Randomized Scheduling Algorithm for Data Aggregation in Wireless Sensor Networks

Mario O. Díaz and Kin K. Leung
Electrical & Electronic Engineering Department, Imperial College, London, UK
{orne.diaz06, kin.leung}@imperial.ac.uk

***Abstract*—We consider a wireless sensor network in which a routing tree has been established to transmit the information from a set of source nodes to a data sink. The existing algorithms to schedule the transmission slots in a way that allows the data to be compressed as it moves towards the data sink are centralized or rely on interference models that fail occasionally. We propose a distributed TDMA scheduling protocol specifically designed for data aggregation called RandSched. RandSched tests whether the transmissions of different sets of nodes succeed simultaneously and only assigns the same slot to them if they have been proved to tolerate each other's interference. By constructing and testing the schedule incrementally, RandSched is more likely to obtain a collision-free schedule than existing algorithms, which is particularly important in large networks. This is confirmed by our simulations, which also reveal a low scheduling overhead and a reduced transmission latency.**

***Index Terms*—distributed TDMA scheduling, data aggregation, testing transmissions, sensor networks**

I. Introduction

Wireless Sensor Networks (WSNs) can greatly expand our ability to monitor many kinds of environments by gathering information that was previously too expensive to obtain. One of the many application domains of WSNs is the monitoring of civil-engineering infrastructures, which is an application domain called structural health monitoring. This work is part of the WINES project [1], which investigates the use of WSNs to monitor bridges, tunnels and water supply systems. In this context, one of the most challenging applications of WSNs is to transmit the acoustic emission signals and the vibration measurements detected in bridges. This application requires data rates of several kHz, much higher data rates than existing bridge deployments [2]. When the bridge vibrations become sufficiently big, the civil engineers need to receive periodically a large amount of acceleration measurements gathered by the sensor nodes, but when those vibrations fade, the WSN switches back to a quiet state wherein no data needs to be reported. This kind of operation is called *event-triggered*.

The event-triggered transmission of large amounts of data poses two main challenges. First, in order to prevent the overflow of the small buffers of the sensor nodes, the sensor nodes have to start forwarding their data to their next hop soon after the event is detected. Second, the intensive use of the radio transceivers strains the batteries of the sensor nodes, which typically consist of two AA batteries that need to last for a couple of years. This source of energy consumption can be greatly reduced by compressing the data from neighboring sensors within the network, which is a process referred to as in-network data aggregation [3]. In network data aggregation requires temporal coordination between the nodes, which can be provided by a TDMA schedule. Although TDMA incurs in overhead during the initial scheduling period, if the traffic is stable during a sufficiently long time, the initial overhead is outweighed by the reduction in idle listening, overhearing and the number of packet collisions during the data transmission period.

The existing TDMA scheduling algorithms for data aggregation are inadequate for the considered application for two main reasons. Firstly, they are likely to obtain *infeasible* schedules, which we define as schedules that generate collisions. This is because they use models to determine whether two nodes can be assigned the same transmission slot without suffering a collision, and such models are bound to fail occasionally because the interference range is unpredictable [4]. We quantify the extent of this problem in our simulation results. Secondly, the existing protocols take a long time and energy to setup the transmission schedule because they are centralized and they require updating the topological information periodically. As a result their overhead scales poorly with the network size. In order to overcome the feasibility and scalability limitations of the existing scheduling algorithms for data aggregation, we propose a distributed and randomized scheduling protocol called RandSched.

The central contribution of RandSched is a novel scheduling mechanism that tests different combinations of concurrent transmissions through a lightweight contention process. By not assuming any interference model and only scheduling concurrently sets of transmitters that are proved to be mutually compatible, RandSched obtains a collision-free schedule despite any irregularities in the interference range. In addition, RandSched's scheduling overhead scales better than that of comparable protocols because it is distributed. These properties enable RandSched to extend the scope of application of TDMA and data aggregation to the large networks envisioned in structural health monitoring applications.

This paper is organized as follows. Section II describes the general operation of the WSN in our considered application

This work is funded by UK EPSRC Research Grant EP/D076838/1, entitled: "Smart Infrastructure: Wireless Sensor Network System for Condition Assessment and Monitoring of Infrastructure".

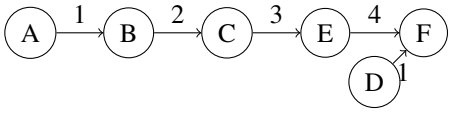

Fig. 1. Example of schedule verifying the precedence requirement. Node F is the data sink and thus the root of the routing tree. Each node is connected to its parent in the tree. The interference relationships are not represented but are fully considered in this work. The slot number assigned to each node is written on top of the arrow that connects it to its parent. The number of slots of the schedule is $M = 4$.

and the properties that it imposes on the TDMA schedule. Section III reviews the related work. Section IV presents the RandSched protocol. Section V proves some results about the feasibility of the schedule computed schedule under certain assumptions. Section VI describes our simulator and sections VII through VIII present and discuss our simulation results. Finally, Section IX concludes the paper.

II. System model

We consider a WSN in which all the relevant information is forwarded to a special node referred to as the *data sink*, possibly across multiple hops. The data sink is typically a gateway connected to the Internet that relays the information it receives to an external server. After the sensor nodes are deployed, the WSN goes through different phases, starting with the *initialization phase*, in which every sensor node discovers the hop distance towards the sink of itself and its neighbors. Then, the network enters the *quiet phase* and stays in that phase as long as there is no data to transmit. During the quiet phase the sensors keep their transceivers off most of the time, but they turn them on periodically to resynchronize themselves with their neighbors, update their neighbor lists, and listen for potential transmissions.

When the sensor nodes decide to report their measurements to the data sink or the data sink commands them to do so, the *setup phase* is executed. The setup phase serves to prepare for the data transmission phase and it has two goals. The first goal is to obtain a *routing tree*. The routing tree is a tree whose nodes are the sensor nodes and whose root is the data sink. Each sensor node in the routing tree will receive packets from its child nodes and transmit packets to its parent node in the routing tree. The routing tree should be adapted to the specific set of data sources in order to compress the data intensively within the network. The FAT protocol [5] can obtain a routing tree quickly and distributedly. The second goal of the setup phase is to obtain a *TDMA schedule*. The TDMA schedule indicates the transmission and reception slots of every node in the routing tree. Let M be the total number of different slots used by the schedule. Every node should have exactly one transmission slot and one reception slot for each of its children in the routing tree. Within each frame, every node should transmit to its parent after it has received the data from all its children, as illustrated in Figure 1. We refer to this requirement as the *precedence requirement*.

The setup phase is followed by the *transmission phase*, which is the only phase in which actual data is transmitted. This phase consists of as many TDMA Frames (TFs) as necessary until all relevant information from the current data stream has been transmitted to the data sink. The transmission phase is typically identical for all the protocols and is represented in Figure 2. Each TF starts with a CSMA period that the nodes can use to request time synchronization signals from their parents or for other purposes. The rest of the TF consists of M Transmission Blocks (TBs), and each TB consists of two slots. The first slot in a TB is to transmit data and the second one is to receive the corresponding ACK. In each TDMA frame, every node receives a packet from each of its children in the routing tree, aggregates their data with its own data to create one compressed packet, and transmits that packet to its parent. Several compression techniques for performing this kind of compression for acoustic emission signals in bridges are presented in [6]. The precedence requirement ensures that the information from every node travels all the way to the data sink within a single TDMA frame, resulting in low latency and minimal buffering requirements.

III. Related work

In-network data aggregation is a popular research topic [3]. In most of the protocols proposed to implement data aggregation, such as TAG [7], the sensor nodes have to contend with their neighbors each time that they have to transmit a packet. Such contention-based MAC protocols are suitable if the traffic pattern is unpredictable or the wireless links change relatively fast. However, this is not usually the case in the WSNs used to monitor civil-engineering infrastructures. Typically, the sensor nodes have to transmit periodic data flows subject to latency constraints. In addition, the wireless links between the sensor nodes change slowly. Under these circumstances, using a TDMA transmission schedule is more energy and bandwidth efficient than using contention-based MAC protocol because the initial scheduling overhead of the TDMA protocols is outweighed by the avoidance of idle listening, overhearing and collisions during the data transmission phase.

TDMA requires time synchronization [8], which is necessary anyway in order to sample the data at the appropriate times. Accurate time synchronization is particularly important in applications using very high sampling rates, such as acoustic emission monitoring [6]. Time synchronization is also used by the FAT protocol [5] to avoid idle listening. When a node is engaged in frequent communication with its parent in the data aggregation tree, the node can receive synchronization information piggybacked in the acknowledgments from its parent at very little additional cost.

Obtaining a schedule with the shortest possible length M is an NP-complete problem [9], so any practical algorithm seeks an approximation of the optimal schedule. Most existing TDMA scheduling algorithms use the k-hop interference model, which considers a transmission successful if the minimum hop distance between the receiver and any interferer exceeds k. This model occasionally fails, causing the protocols that rely upon it to obtain an infeasible schedule. This problem is introduced in [10] and quantified in our simulation results. A

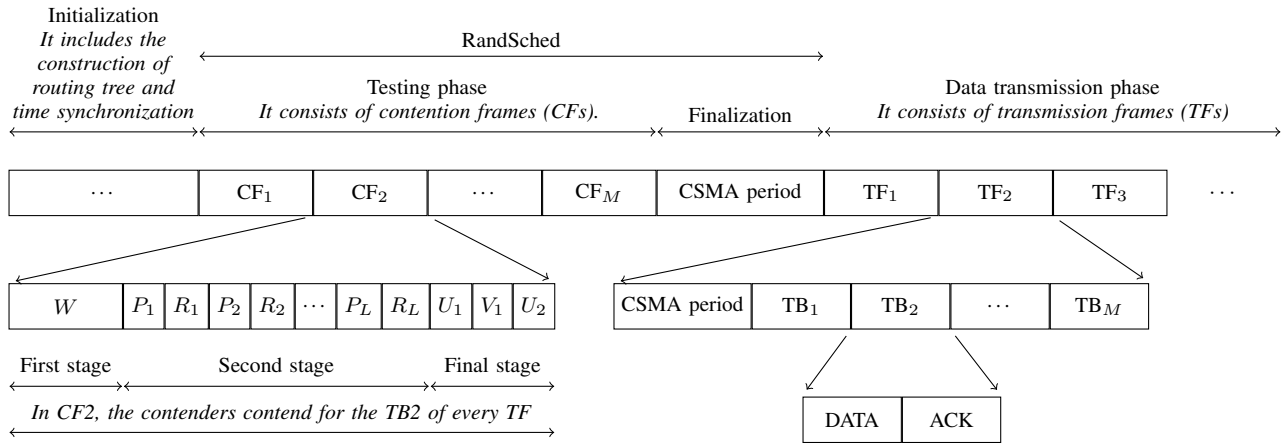

Fig. 2. Time slots used by the RandSched protocol to obtain a TDMA transmission schedule. The testing phase consists of contention frames (CFs), and it concludes when all the nodes have gained a transmission slot in the schedule. Each CF consists of a contention window W and $2L + 3$ slots. The P_i slots are used by the contenders and the R_i slots by the servers.

group of protocols, called adaptive scheduling protocols, solve this problem by rescheduling the unduly scheduled nodes during the transmission phase. FlexiTP [10] is one such protocol. However, FlexiTP does not satisfy the precedence requirement. To the best of our knowledge, no adaptive scheduling protocols with that property have been published and it would be difficult to design one such protocol.

Algorithms [11] and [9] verify the precedence requirement. We refer to both of them as BF2 because they are very similar, they assign slots by traversing the routing tree in Breadth-First order, and they use the 2-hop interference model. Similarly, we refer to their direct extension using the 3-hop interference model as BF3. Our simulation results in Section VIII-B show that BF2 is likely to obtain an infeasible schedule and that BF3 performs better in this respect. Both BF2 and BF3 scale poorly with the network size because they are centralized, which means that the sink needs to receive updates on each node's neighbors and to inform every node of its receiving and transmitting slots. The EMAC protocol [12] is distributed and incurs in very little overhead. However, EMAC's interference model is extremely poor because it only considers the hop distance in the routing tree as opposed to the hop distance in the connectivity graph. To sum up, the existing scheduling algorithms for data aggregation are not simultaneously scalable and robust to the interference-irregularity problem.

IV. The RandSched Protocol

To overcome the limitations of the existing protocols, we propose a distributed scheduling protocol for data aggregation that satisfies the precedence requirement. We call our protocol RandSched. In contrast to existing protocols, RandSched does not use a model to determine whether the transmissions of two nodes will succeed simultaneously. RandSched's novel approach consists of performing tests to determine whether a set of nodes can be scheduled concurrently without collisions. RandSched only schedules a set of transmitters concurrently if one of the tests confirms that each node in the set can tolerate the joint interference from all the other nodes in the set. RandSched consists of a *testing phase* and a *finalization phase*, as depicted in Figure 2.

The testing phase consists of Contention Frames (CFs). There are as many CFs as necessary until every node in the routing tree has obtained one Transmission Block (TB) that it can use in every frame of the data transmission phase to communicate with its parent. In each CF, some nodes contend to obtain one TB from their respective parents. Those nodes are called *contenders*, and those of them that succeed in their attempts to gain a TB are referred to as *winners* of the current CF. The winners of CF_i are assigned TB_i. In order to satisfy the precedence requirement, a node only contends for a TB when all its children have gained a slot. An exception to this rule occurs when a node detects that one of its child nodes has not obtained a slot or contended for one for a certain period, in which case the node ignores the child because the child is probably dead.

When every child node of the data sink has obtained a slot or a certain sufficiently long period has elapsed, the testing phase concludes. Then, the data sink starts the finalization phase by creating a packet called *finalization packet*, which is a packet containing information about the aggregation process such as the start time of the first TDMA frame and the frame period. The purpose of the finalization packet is to propagate the finalization packet to every node in the network. To propagate the finalization packet, every node forwards the packet to each one of its children using CSMA. In order to ensure the correct reception of the finalization packet, every node retransmits this packet up to a certain maximum number of times until it receives an ACK from each one of its child nodes.

A. Contention process during a Contention Frame (CF)

We explain now the mechanism whereby the sensor nodes can gain a TB during one of the CFs of the testing phase. In each CF, any node some of whose children have not gained a TB is called a *server*, and any node that has not won a TB

but all of whose children have already won a TB is called a *contender*. During a CF, every server listens for REQ packets from its child nodes in the routing tree and every contender contends for a TB by transmitting REQ packets to its parent. The contenders use the REQ packets to test whether they can communicate with their respective parents despite the interference from other contenders. Figure 2 shows the structure of a CF. The contention process during a CF is distributed and consists of three different stages that every contender needs to pass in order to become a winner. The contention process aims to maximize the number of winners per CF in order to reduce the length of the schedule M. Next, we detail the algorithms executed by contenders and servers during a CF.

1) Algorithm executed by each contender during a CF: At the beginning of the CF, each contender chooses a random time within the W slot and checks the channel at that time. If the contender senses the channel busy, it withdraws from the contention in the current CF. Otherwise, the contender succeeds in the first stage of the contention and starts transmitting rubbish from that moment until the end of W in order to try to prevent other contenders incompatible with itself from becoming winners of the current CF.

At the end of the contention window W, each remaining contender chooses a random integer r between 1 and M and sleeps until slot P_r. In slot P_r, the contender sends a REQ packet to its parent, and if it fails to receive an ACK in slot R_r it withdraws from the contention in the current CF. On the other hand, if it receives an ACK, the contender succeeds in the second stage of the contention and we call it a *finalist*. Each finalist transmits rubbish in slots $P_{r+1}, \ldots, P_M$ without listening for replies. This is to increase the probability that other contenders incompatible with itself will fail to gain a TB in the current CF.

In the third stage of the contention every finalist transmits a REQ packet in slot U_1, and if it fails to receive an ACK in slot V_1, it withdraws from the contention. Otherwise, the contender succeeds in the third and last stage of the contention and has become a winner of TB. Each winner declares its victory to its parent by transmitting a final REQ packet in slot U_2. Then, the winner listens continuously until it receives a finalization packet. Before receiving the finalization packet, the winner may receive a packet asking the winner the number of the TB that it obtained, in which case the winner replies with that information.

2) Algorithm executed by each server during a CF: The server listens in all the P_i slots, and if it receives a packet from a child X in slot P_j, it transmits an ACK in slot R_j and rubbish in slots $R_{j+1}, R_{j+2}, \ldots, R_L$ in order to try to prevent the success of the contenders incompatible with X. In slot U_1, if the server receives a REQ packet from X, it replies with an ACK in slot V_1. Finally, if it receives another REQ from X in slot U_2, the server records X as the winner of the TB that is disputed in the current CF.

When all the child nodes of the contender have obtained a TB, the server becomes a contender. However, if one of its child nodes, which we call X, has not obtained a slot for a long time, it may have been because X's declaration of victory in one of the U_2 slots was lost. To account for this situation, the server transmits a packet in any of the P_i slots asking X to reply with a packet indicating the number of the TB that it obtained. The contender may repeat this step several times, and if it does not receive any response, the contender removes X from its children list.

V. Analysis of the feasibility computed schedule

Although it is common that the wireless links between the sensor nodes change slowly over time and that the amount of external interference is relatively small, at some points in time these conditions may not be true. When these conditions do not hold, we say that *temporal anomalies* have occurred. If these anomalies are sufficiently frequent, they can disrupt any scheduling protocol, and this applies also to RandSched. However, RandSched is designed to handle several of these anomalies. For example, RandSched is able to recover from losses in the U_2 slot, which is very important in order to avoid a deadlock. RandSched also avoids other deadlocks by using timers and setting a maximum number of repetitions in several occasions.

In this section we assume that the wireless links do not change over time and that there is no external interference. These assumptions are not strictly true in practice and RandSched does not require them in order to be useful, but, if they hold, RandSched verifies the important property of being able to guarantee a collision-free schedule. This property justifies why in our simulations RandSched always obtains a collision-free schedule. The existing protocols do not verify this property because the models that they use to determine which nodes can be scheduled concurrently fail occasionally. Because of this, in our simulations BF2 and BF3 obtain schedules that suffer packet collisions despite the fact that BF2 and BF3 are simulated under the same conditions as RandSched. The proof about the collision-free nature of RandSched is interesting because it helps understand the way RandSched works and the rationale of its design.

We now introduce some definitions. We say that a set of nodes form a *feasible set* if two conditions hold. First, if they transmit simultaneously and there are no other simultaneous transmissions, their parents receive their respective transmissions successfully. Second, if their parents transmit simultaneously and there are no other simultaneous transmitters, the concerned children receive their respective transmissions successfully. We define a *collision-free schedule* as a schedule that only schedules simultaneously feasible sets of nodes. In our following proof, we also assume that RandSched does not use timers or a maximum number of repetitions, since this mechanisms are only designed to cope with external interference and node failures.

Lemma 1: If RandSched finds a schedule, it is collision-free

Proof: RandSched only schedules simultaneously the winners of the same CF, and thus we have to prove that the winners of every CF form a feasible set. Let us see that this is the case. In each CF, every finalist transmits a REQ packet to its

parent in slot U_1. Some of the finalists may not receive an ACK in slot V_1, thereby failing to become winners. However, we know that those that become winners were able to communicate with their parents and receive their respective replies despite the interference of the unsuccessful finalists. Therefore, the winners will also be able to communicate with their parents in slot U_2 because there is less interference then. In other words, when a node considers itself a winner, its parent will be aware of that, ensuring that the schedule is consistent. In addition, the transmitters were proved to be a feasible set in slots U_1 and V_1. ■

Theorem 2: If $L \geq 2$ and a feasible schedule exists, RandSched always obtains a collision-free schedule.

Proof: We have to prove that the algorithm does not reach a deadlock. We saw that every node is aware of the victories of its children, which means that no server never waits unnecessarily for its children and there is at least one contender in every CF. Since we also have $L \geq 2$, there is a nonzero probability that exactly one contender X will transmit a REQ packet in slot P_1. When this happens, X receives an ACK because there are no other interferers and a schedule exists, and thus X could become a finalist. In slots P_2 to P_L, X transmits rubbish, and other contenders may transmit their first REQ. If in slot P_2 a contender Y transmits a REQ, Y only becomes a finalist if tolerated the interference of X and other contenders in slot P_2. If Y becomes a finalist, X may not be able to tolerate Y, in which case X would not become a winner in the current CF, but at least one node, Y, would become a winner if there were no more contenders. Repeating this argument in every P_i slot for $i \in \{3, \dots L\}$, we obtain that there will be at least one finalist that can tolerate the interference of all the other finalists, and thus there will be at least a winner in the CF under our initial assumptions. Therefore, the probability of there being some winners in every CF is strictly positive. This property, together with the fact that there are as many CFs as necessary, means that a schedule will be found eventually, and due to the previous lemma, it will be feasible. ■

VI. Simulator description

We wrote a packet-level simulator in Python whose code is available in [13]. The simulator decides that a node receives a packet correctly if the SINR at its transceiver exceeds 20 dB. The noise figure of the transceivers is 5. The wireless channel is static, with attenuation exponent $\alpha = 3.5$, path loss at 100 m of 80 dB, and log-normal shadow fading with standard deviation \$$\sigma_f$=8\$ dB. Although there is no common transmission range for all the network, we define t as a node's hypothetical transmission range if there were no fading and interference; in our case, t is 48 m.

The sensor nodes are deployed randomly within an $8t \times 3t$ rectangle, and the sink is in the middle of the left border of that rectangle. We define A as the area of the rectangle, and the node density ρ as $N(\pi t^2)/A$, where N is the number of nodes. We say that a random node deployment yields a connected network if at least 90 % of the nodes can reach the gateway directly or through multiple hops. In our simulations, we discard any deployment that yields an unconnected network, which is rarely necessary for $\rho \geq 7$. We set all the nodes as data sources and use the shortest path tree as the routing tree.

VII. Properties of the scheduling process

An important performance metric of RandSched is the duration of the testing phase. Since according to Figure 2 this duration is $M \times L$, we now examine L and M and how they scales with the network size and the node density

A. Choosing the optimal L

In order to reduce the duration of the scheduling operation, we should choose the smallest value of L that makes the protocol work properly. In particular, we need a value of L that yields a sufficiently small M. To see how a larger L reduces M, consider the operation of RandSched in the case where two neighboring contenders contend to obtain a slot during the same CF. Also assume that either each contender's transmission to its parent cannot succeed if the other contender transmits simultaneously, or the ACKs of their respective parents collide if they are transmitted simultaneously. Now let us compare two alternative scenarios. The first scenario is that during the second stage of the contention the two contenders transmit their first packet in different slots, and the second scenario is that they transmit in the same slot. In the first scenario, one of the contenders may obtain a slot in the current CF, whereas in the second scenario no contender will succeed in the current CF. Therefore, if we increase L we increase the probability of the first scenario and thus we reduce M.

We average 500 simulation runs to evaluate the influence of L on M and present the results in Figure 3. The figure reveals that a value of L as small as 4 is sufficient to achieve a sufficiently small M across all node densities. In particular, $L = 4$ yields a schedule less than 6 % longer than the schedule obtained for $L = 12$. Choosing $L = 4$ implies that each CF in the testing frame consists of twelve slots, which may represent a significant overhead for small networks. However, we are mostly interested in how our protocol scales with the network size. We know that larger networks do not require bigger values of L because the purpose of a large L is to resolve the collisions within a local area. In addition, Figure 3 shows that the value of L required to obtain a small M does not depend on the node density either. Since the overhead of RandSched is dominated by the factor $M \times L$ and L is constant, the scalability of RandSched is determined by the growth of M with the network size, which we study next. In the rest of our simulations we set $L = 12$.

B. Growth of M with the network size and the node density

We evaluate the dependence of M with the network size with another set of simulations. We set $\rho = 7$ and consider a monitored area with the shape of a square with side s. We perform simulations for values of s of up to $12t$. For the biggest simulated area, the number of nodes in the network is $N = 321$. Figure 4 presents the average of 100 simulations

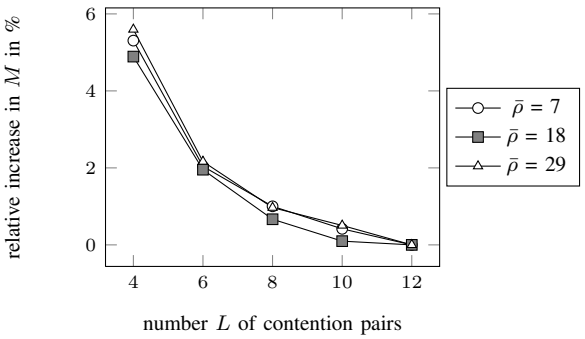

Fig. 3. Influence of the number L of pairs of contention slots per CF on the size M of the computed schedule. The schedule size M decreases with L. The graph compares the value of M for different values of L with the value of M for the largest tested value of L.

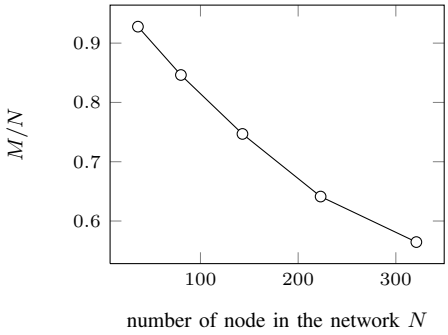

Fig. 4. The length M of the schedule computed by RandSched grows more slowly than the number of nodes in the network N.

and shows that M, and thus RandSched's overhead, grows less than linearly with N. This is because in large networks childless nodes are approximately uniformly distributed in the network and the number of winners per contention frame is approximately proportional to the monitored area, particularly in the first few CFs. Therefore, as the network size grows, more nodes can be assigned the same TB.

Figure 5, which results from averaging 3000 simulation runs, shows that RandSched obtains requires a small number M of scheduling operations independently of the node density ρ.

C. Scheduling overhead of the three protocols

The total overhead of RandSched includes the overhead of both the testing and the finalization phase. However, the finalization phase is much shorter than the one of the testing phase. In fact, it is possible to use during the finalization phase the inverse of the schedule computed during the testing phase, with the difference that instead of using full CFs it is sufficient to transmit a short packet. Therefore, the overhead of RandSched is given by the overhead of the testing phase, that we have shown to grow at a lower rate than N. In contrast, since BF2 and BF3 are centralized, they require that the data sink informs every node of its transmission and reception slots. As a result, the nodes near the sink have to relay an amount of traffic proportional to N, and thus the execution time of BF2 and BF3 grows at least linearly with the number of nodes in the network N. Therefore, the *execution time* of RandSched grows with the network size at a lower rate than BF2 and BF3 do.

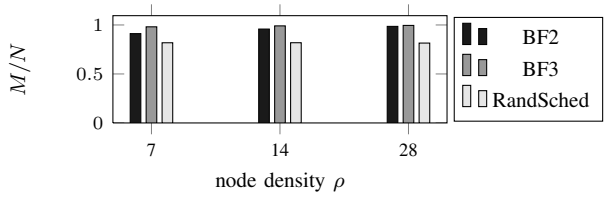

Fig. 5. RandSched outperforms BF2 and BF3 in terms of latency for different node densities ρ. Here, M is the schedule length and N is the number of nodes in the network. The latency increases with M.

In all the considered scheduling protocols, the sensor nodes are listening during most part of the scheduling process without receiving anything. This idle listening is the major source of energy consumption of the three protocols, which is very directly connected to the their execution time. Therefore, the *energy consumed* by RandSched also scales better with the network size than that of BF2 and BF3.

VIII. Quality of the computed schedule

A. Upstream latency

We define *upstream latency* as the maximum interval required by any packet to travel from any source node to the data sink. In order to reduce the upstream latency we have to reduce M, which is shown in Figure 5 as a function of the node density. According to this figure, RandSched enjoys a smaller latency than BF2 and BF3 do for all the simulated node densities. This can be explained by the fact that the BF algorithms tend to assign the last slots in the schedule to the nodes close to the sink independently of their number of children, whereas RandSched tends to assign the first slots to childless nodes regardless of their depth in the tree. Figure 5 also shows that BF2 slightly outperforms BF3 in terms of M. This is because it imposes fewer conditions on a set of transmitters to be considered feasible.

B. Feasibility of the computed schedule

We define P_u as the probability that a node in the routing tree is unable to reach the data sink due to an infeasible schedule. We estimate P_u by averaging the results of 3000 simulation runs and present the results in Table I. Considering the number of simulations and that the number N of simulated nodes varies between 53 and 214 for different values of the node density ρ, our simulations cannot estimate P_u when P_u is smaller than 10^{-3}, but such small values of P_u are acceptable in many applications. Therefore, using only our simulations results it would be incorrect for Table I to show $P_u = 0$ for RandSched, since the best we can claim based on simulations only is $P_u < 10^{-3}$. The reason why we write $P_u = 0$ and not $P_u < 10^{-3}$ is that based on the previous section we know that under our simulation settings, which are the same for the three protocols, RandSched always schedules all the nodes correctly.

	protocol		
ρ	BF2	BF3	RandSched
7	0.0796	$\approx 10^{-3}$	0 (theoretical)
14	0.0321	$< 10^{-3}$	0 (theoretical)
28	0.0098	$< 10^{-3}$	0 (theoretical)

TABLE I
PROBABILITY P_u THAT A NODE CANNOT REACH ITS PARENT DUE TO AN ERROR IN THE COMPUTED SCHEDULE. THIS PROBABILITY IS SHOWN FOR DIFFERENT PROTOCOLS AND VALUES OF THE NODE DENSITY ρ.

This can obviously not be guaranteed if the channel properties vary during the scheduling process.

Table I shows that BF2 suffers high P_u, particularly for sparse deployments, whereas BF3 performs significantly better. We see that P_u decreases with the node density ρ. This is because as ρ grows, the number of hops between two nodes becomes more strongly correlated with the distance between them, and that distance is an important factor in the attenuation between them. Although for small networks BF3 may obtain a value of P_u sufficiently small, this may not be the case for very large networks, since in large networks some packets carry the information from many nodes. RandSched is unique in that it combines a low P_u with a distributed operation.

IX. CONCLUSION AND FUTURE WORK

In many WSNs used in structural health monitoring, the wireless links change slowly with time and external interference occurs sporadically. In such WSNs, TDMA protocols offer important advantages over contention-based protocols. However, in this paper we have shown that the existing TDMA scheduling algorithms for data aggregation do not simultaneously obtain a collision-free schedule with high probability and incur in an overhead that scales adequately with the number of nodes in the network. Obtaining a collision-free schedule is particularly important for data aggregation since some packets contain the information from many nodes and losing those packets would lose a great amount information. The existing scheduling algorithms are unlikely to obtain to a collision-free schedule because they use models to determine which nodes can be assigned the same slot and those models are bound to fail.

We have proposed a TDMA scheduling protocol for data aggregation called RandSched. RandSched builds the transmission schedule distributedly and incrementally. In each scheduling step, RandSched performs a test to check whether the set of nodes to which it plans to assign the same slot can tolerate the interference from the other nodes in the set and whether the ACKs are also successfully received. In this way, RandSched is very likely to obtain a collision-free schedule.

Our simulations show that RandSched obtains shorter schedules and thus shorter latencies than the existing algorithms. RandSched's overhead is relatively small since twelve slots are sufficient to determine a set of concurrent transmitters independently of the node density and the network size. In addition, each node does not need to incur the overhead of monitoring its neighbors and reporting any changes to the to the data sink, possibly across multiple hops. More importantly, RandSched's overhead grows more slowly than that of existing protocols. This property and the better feasibility guarantees make RandSched more scalable than the existing protocols.

In our future work, we will explore other applications of our contention mechanism. Furthermore, we will modify our protocol so that once the data aggregation schedule is computed it can be modified on demand with as few changes as possible.

REFERENCES

[1] WINES Consortium, "Wired and Wireless Intelligent Networked Systems (WINES) – Smart Infrastructure Project," *[Online]. Available: www.winesinfrastructure.org*, 2008.

[2] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. D. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proc. ACM/IEEE Conf. Information Processing in Sensor Networks (IPSN)*, 2007, pp. 254–263.

[3] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Commun. Mag.*, vol. 14, no. 2, pp. 70–87, 2007.

[4] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Impact of radio irregularity on wireless sensor networks," in *Proc. ACM Int'l Conf. Mobile Systems, Applications, and Services (MobiSys)*, Boston, MA, USA, Jun. 2004, pp. 125–138.

[5] M. O. Díaz and K. K. Leung, "Efficient data aggregation and transport in wireless sensor networks," *Wiley J. Wireless Comm. and Mobile Computing*, 2009.

[6] C. U. Grosse, S. D. Glaser, and M. Krüger, "Condition monitoring of concrete structures using wireless sensor networks and MEMS," in *Proc. Society for Photo-optical Instrumentation Engineers (SPIE)*, vol. 6174, 2006, pp. 407–418.

[7] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation service for ad-hoc sensor networks," in *Proc. ACM SIGOPS Symp. Operating System Design and Implementation (OSDI)*. Boston, MA, USA: ACM Press, Dec. 2002, pp. 131–146.

[8] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Elsevier J. Computer Networks*, vol. 52, no. 12, pp. 2292 – 2330, 2008.

[9] R. Mangharam, A. Rowe, and R. Rajkumar, "FireFly: a cross-layer platform for real-time embedded wireless networks," *Springer J. Real-Time Systems*, vol. 37, no. 3, pp. 183–231, Dec. 2007.

[10] W. L. Lee, A. Datta, and R. Cardell-Oliver, "FlexiTP: A flexible-schedule-based TDMA protocol for fault-tolerant and energy-efficient wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 6, pp. 851–864, Jun. 2008.

[11] V. Annamalai and S. K. S. Gupta, "On tree-based convergecasting in wireless sensor networks," in *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC)*, S. Gupta, Ed., vol. 3, 2003, pp. 1942–1947.

[12] X. Liang, W. Li, and T. A. Gulliver, "An energy-efficient MAC protocol exploiting the tree structure in wireless sensor networks," in *Proc. IEEE Military Comm. Conf.* , W. Li, Ed., 2007, pp. 1–7.

[13] M. O. Díaz, "Randsched implementation," *http://github.com/ornediaz/wsnpy.git*, 2009.