Dynamic Conflict-free Query Scheduling for Wireless Sensor Networks

Octav Chipara†, Chenyang Lu† and John Stankovic‡
† Department of Computer Science and Engineering, Washington University in St. Louis, {ochipara, lu}@cse.wustl.edu
‡ Department of Computer Science, University of Virginia, stankovic@cs.virginia.edu

***Abstract*— With the emergence of high data rate sensor network applications, there is an increasing demand for high-performance query services in such networks. To meet this challenge, we propose Dynamic Conflict-free Query Scheduling (DCQS), a novel scheduling technique for queries in wireless sensor networks. In contrast to earlier TDMA protocols designed for general-purpose networks and workloads, DCQS is specifically designed for query services supporting in-network data aggregation. DCQS has several important features. First, it optimizes the query performance and energy efficiency by exploiting the temporal properties and precedence constraints introduced by data aggregation. Second, it can efficiently adapt to dynamic workloads and rate changes without explicitly reconstructing the transmission schedule. In addition, we provide an analytical capacity bound for DCQS in terms of query completion rate. This bound enables DCQS to handle overload through rate control. NS2 simulation results demonstrate that DCQS significantly outperforms a representative TDMA protocol (DRAND) and the 802.11 protocol in terms of query latency, throughput, and energy efficiency.**

I. Introduction

Early research on wireless sensor networks (WSNs) has focused on low data rate applications such as habitat monitoring [1][2]. In contrast, recent years have seen the emergence of high data rate applications such as real-time structural health monitoring [3] and preventive equipment maintenance [4]. For instance, a structural health monitoring system may need to sample the acceleration of each sensor at a rate as high as 500 Hz, resulting in high network load when a large number of sensors are deployed for fine-grained monitoring. Moreover, the system may have highly variable workload in response to environmental changes. For example, an earthquake may trigger a large number of new queries in order to assess any potential damage to the structure. Therefore, a key challenge is to provide a high-throughput query service that can collect data from large networks and adapt to dynamic workload changes.

To meet this challenge we present *Dynamic Conflict-free Query Scheduling* (DCQS), a novel query scheduling protocol designed to meet the communication needs of high data rate applications. DCQS can be integrated with query services (e.g., TinyDB [5]) for WSNs. A query service allows an application or user to submit queries that periodically collect data from a number of sources through a WSN. To improve performance and conserve energy, the query service usually performs in-network aggregation [5] as data is routed toward a base station. In contrast to earlier TDMA scheduling techniques designed for general workloads and networks, DCQS is unique in that it takes advantage of the common properties of WSN query services to construct a conflict-free transmission schedule dynamically.

Specifically, DCQS has the following salient features: (1) By tailoring the transmission schedule to the common communication patterns introduced by periodic queries and in-network aggregation, DCQS can achieve the high throughput and low latency required by high data rate applications. (2) DCQS can *dynamically* adapt the transmission schedule in response to workload changes. As a result, queries may be added, removed, or their rates may be changed without the need to recompute the transmission schedule. This property makes DCQS particularly suitable for applications with variable workloads. (3) DCQS provides predictable performance in terms of both the maximum achievable query rate and power consumption. The predictability of DCQS enables it to effectively handle overload through simple rate control techniques and provide predictable network lifetime. (4) DCQS is designed to work on resource constrained devices with limited memory and processing power.

The remainder of the paper is organized as follows. Section II compares our approach to existing work. Section III describes the query and network models we adopt. Section IV details the design and analysis of DCQS. Section V describes how DCQS handles dynamic networks and workloads. Section VI provides simulation results using NS2. Section VII concludes the paper.

II. Related Work

TDMA scheduling is attractive for high data rate sensor networks because it is energy efficient and may provide higher throughput than CSMA/CA protocols under heavy load. Two types of TDMA scheduling problems have been investigated in the literature: node scheduling and link scheduling. In node scheduling, the scheduler assigns slots to nodes whereas, in link scheduling, the scheduler assigns slots to links through which pairs of nodes communicate. In contrast to earlier work, DCQS adopts a novel approach which we call *query scheduling*. Instead of assigning slots to each node or link, we assign slots to *transmissions* based on the specific communication patterns and temporal properties of queries in WSNs. This approach allows DCQS to achieve high throughput and low latency.

Early TDMA scheduling protocols were designed for static or uniform workloads [6][7][8][9]. Such approaches are not

1-4244-0593-9/06/$20.00 ©2006 IEEE

suitable for dynamic applications with variable and non-uniform workloads. Several recent TDMA protocols can adapt to changes in workload. A common method to handle variable workloads is to have nodes periodically exchange traffic statistics and then adjust the TDMA schedule based on the observed workload [6][10][11]. However, exchanging traffic statistics frequently may introduce non-negligible communication overhead. In contrast, DCQS can efficiently adapt to changes in workloads by exploiting explicit query information provided by the query service. Furthermore, it features a local scheduling algorithm that can accommodate changes in query rates and completions without explicitly reconstructing the schedule.

TinyDB [5] is a representative query service that allows a user to collect aggregated data from a sensor network through a routing tree. It employs a coarse-grained scheduling scheme that evenly divides the period of a query into communication slots for nodes at different levels in a routing tree. TinyDB does not address scheduling for multiple queries with different timing properties. Moreover, the schedule of each node is fixed and does not adapt to the workload. DCQS can be integrated with TinyDB to enhance its performance and flexibility in the face of heavy and variable workloads.

III. System Models

A. Query Model

DCQS assumes a common query model in which source nodes produce data reports periodically. This model fits many applications that gather data from the environment at user specified rates. Such applications generally rely on existing query services such as TinyDB [12]. A query is characterized by the following parameters: a set of sources that respond to a query, a function for in-network aggregation [5], the query period P_q, and the start time of the query ϕ_q. Based on the temporal properties of a query, *query instances* are released periodically to gather data from the WSN. We use the notation $I_{q,k}$ to refer to the kth instance of query q. The query instance $I_{q,k}$ is released at time $R_{q,k} = \phi_q + k \cdot P_q$ which we call the release time of $I_{q,k}$.

A query service usually works as follows: a user issues a query to a sensor network through a base station, which disseminates the query description to all nodes. The query description includes all query parameters. To facilitate data aggregation, the query service constructs a *routing tree* rooted at the base station as the query is disseminated. The execution of a query instance entails in-network data aggregation. Accordingly, each non-leaf node waits to receive the data reports from its children, produces a new data report by aggregating its data with the children's data reports, and then sends it to its parent. We assume that there is a single routing tree that spans all nodes and it is used to execute all queries. This assumption is consistent with the approach adopted by existing query services [5]. During the lifetime of the application the user may issue new queries, remove queries from execution, or change the parameters of existing queries. DCQS is designed to support dynamic queries efficiently.

Query services designed for WSNs usually support a wide range of aggregation functions (e.g., min, sum, average, and histogram) to improve the energy efficiency and performance of queries. In spite of the diversity of queries that may be issued, it is often the case that the communication workload induced by different queries may be similar due to in-network aggregation. For example, in TinyDB [5] queries for the maximum temperature and the average humidity in a building induce the same workload in the network: each node receives a packet from every child, and then sends a packet to its parent. For the max query, the outgoing packet includes the maximum value of the data reports from itself and its children. For the average query, the packet includes the sums of the values and the number data sources that contributed to the sum. Therefore, with respect to the communication requirements, these queries are indistinguishable. Each aggregation function has an upper-bound on the number of packets a node transmits. Let $W_{q,a}$ be the number of packets node a must transmit to satisfy the workload demand of a query q. We introduce the concept of *query class* to denote those queries that induce the same workload demand.

B. Network Model

DCQS works by constructing a conflict-free schedule for query execution. To facilitate this we introduce the Interference-Communication (IC) graph. The IC graph, IC(E,V), has all nodes as vertices and has two types of directed edges: *communication* and *interference* edges. A communication edge $\overrightarrow{ab}$ indicates that the packets transmitted by a may be received by b. A subset of the communication edges forms the routing tree that is used for data aggregation. An interference edge $\overrightarrow{ab}$ indicates that a's transmission interferes with any transmission intended for b even though a's transmission may not be correctly received by b. The IC graph is used to determine if two transmissions may be scheduled concurrently. We say that two transmissions, $\overrightarrow{ab}$ and $\overrightarrow{cd}$ are *conflict-free* ($\overrightarrow{ab} \parallel \overrightarrow{cd}$) and may be scheduled concurrently if (1) a, b, c, and d are distinct and (2) $\overrightarrow{ad}$ and $\overrightarrow{cb}$ are not communication or interference edges in E.

The IC graph accounts for link asymmetry and for the irregular communication and interference ranges observed in WSN [13]. The IC graph may be stored in a distributed fashion: each node *only* needs to know its incoming/outgoing communication and interference edges. It is feasible for a node to determine its own communication and interference edges. A practical solution for constructing the IC graph is presented in [13].

IV. Protocol Design

DCQS has two core components: a *planner* and a *scheduler*. First, for each query class, the planner constructs a *transmission plan* according to which query instances of that class are executed. A transmission plan is an ordered sequence of steps, each comprised of a set of conflict-free transmissions. To reduce the query latency, the planner minimizes the length

of the transmission plan while enforcing the precedence constraints required by data aggregation. Second, the scheduler dynamically determines in what time slot each step in the transmission plan should be executed. The scheduler executes a step by executing the set of transmissions it contains. To maximize the query completion rate, the scheduler may overlap the execution of multiple query instances (of one or multiple queries) by executing a step in each of their transmission plans in the same slot. The scheduler ensures that the transmissions executed in a slot are conflict-free by enforcing a minimum inter-release time between the times when a node starts the execution of two consecutive query instances. This is a key feature of DCQS.

In presenting DCQS, we assume that clocks are synchronized and the slot size is sufficiently large to transmit a single packet. Clock synchronization is a fundamental service in WSN as many applications must time-stamp their sensor readings to infer meaningful information about the observed events. Several time synchronization protocols for WSNs have been proposed [14][15]. We also assume that the routing tree and the IC graph are constructed in a bootstrapping phase. In the rest of this paper, we first present a centralized planner, which serves as a starting point for the design of the distributed protocol. We then describe the local scheduler and the distributed planner.

A. The Centralized Planner

In this section we present a centralized version of the planner. In presenting the centralized planner we assume the node executing the planner knows the entire IC graph. The decentralized planner (see Section IV-C) removes this assumption.

Definitions. A *transmission plan* is an ordered sequence of *steps* that executes a query instance. Instances of queries belonging to the same query class have the same transmission plan. This property allows DCQS to amortize the cost of constructing a query plan over many queries and hence it effectively reduces its overhead. A transmission plan has the following properties: (1) In each step a set of conflict-free transmissions are assigned. (2) The transmission plan respects the precedence constraints introduced by data aggregation: a node is assigned to transmit in a later step than any of its children. (3) Each node is assigned in sufficient steps to meet its workload demand. We use $T_c[s]$ to denote the set of transmissions assigned to step s in the transmission plan of query class c. L_c is the length of the transmission plan.

Since the execution of a query instance entails a node performing a data aggregation operation, a node must wait to receive the reports from all its children before it may transmit the aggregated data report to its parent. Therefore, to minimize the query latency, the planner assigns the transmissions of a node with a larger depth in the routing tree to an earlier step in the transmission plan. This strategy reduces query latency because it reduces the time a node waits to receive the data reports from all its children.

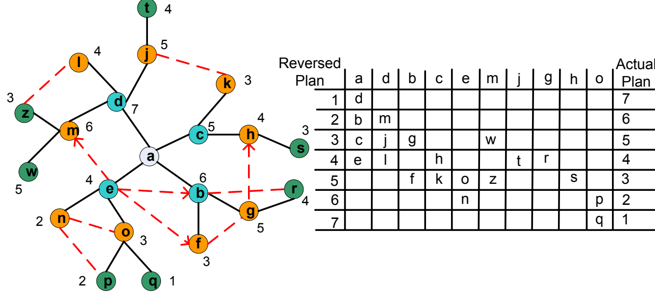
The pseudo-code of the centralized planner is shown in Fig. 1. The centralized planner works in two stages. In the first stage the planner constructs a *reversed* transmission plan (R_c) in which a node's transmission is assigned to an *earlier* step than its children. In the second stage it constructs the actual plan (T_c) by reversing the order of the steps to enforce the precedence constraints. The planner maintains two sets of nodes: *completed* and *eligible*. Node n is a member of the *completed* set if the planner already assigned sufficient steps to meet n's workload demand. The set *eligible* contains nodes whose parents are in the *completed* set. Initially, the *completed* set contains the root and the *eligible* set the children of the root. The planner considers the eligible nodes in order of their *priority* and assigns steps in which they transmit to their parents. The priority of a node depends on its depth, number of children, and ID. Nodes with smaller depth have a higher priority. Among the nodes with the same depth the ones with more children have higher priority. Node IDs are used to break ties. After the planner assigns steps for n to transmit to its parent, it moves n from the *eligible* set to the completed set, and adds n's children to the *eligible* set. The first stage is completed when the *completed* set contains all the nodes in the network. In the second stage, the planner reverses the order of the steps in the reversed transmission plan to obtain the actual plan (see line 7).

Let us consider how the scheduler assigns n's transmissions to its parent p in the reversed transmission plan. The planner associates with each node two pieces information. $n.minStep$ is the step number in which the planner attempts to assign n's transmission to p. In the field $n.assignedSteps$ the planner maintains the number of steps in which n is assigned to transmit. Since nodes with smaller depth have a higher priority, p's transmissions to its parent has already been assigned to enough steps. Let s be the last step in the reversed plan R_c in which p transmits to its parent. In the reversed plan the earliest step in which n may transmit its own data report to p is $R_c[n.minStep]$, where $n.minStep = s + 1$. This means that, in the actual plan, p must transmit its data report to its parent at least one step before the parent transmits its data report. To determine if the transmission $\overrightarrow{np}$ may be assigned to $R_c[n.minStep]$ without conflict, n must verify that all transmission pairs that involve $\overrightarrow{np}$ and any transmission already assigned to $R_c[n.minStep]$ are conflict free. The planner assigns node n to transmit in multiple steps until its workload demand $W_{q,n}$ is met.

Fig. 2 shows an example topology and the transmission plan generated by the central planner. All nodes have a workload demand of one packet. Initially, the children of the root a are eligible. The planner starts by scheduling d since it has the highest-priority among the eligible nodes (i.e., a's children). The planner assign $\overrightarrow{da}$ to step 1 since $R_c[1] = \emptyset$. Next, b becomes the highest-priority eligible node. The first step in which $\overrightarrow{ba}$ may be assigned is step 1. However, since $\overrightarrow{ba} \nparallel \overrightarrow{da}$, $\overrightarrow{ba}$ cannot be assigned to that step. We assign $\overrightarrow{ba}$ to step 2 since $R_c[2] = \emptyset$. Similarly, $\overrightarrow{ca}$ and $\overrightarrow{ea}$ are assigned to steps

centralized-planner:
1: $completed = \{root\}$; $eligible = children(root)$;
2: while ($completed \neq V$)
3: Let n be the highest-priority node in $eligible$
4: invoke assign-steps(n)
5: $completed = completed \cup \{n\}$
6: $eligible = eligible \cup$ children(n)
7: reverse plan: $T_c[s] = R_c[L - s]$
assign-steps(n):
9: Let p be n's parent and assigned.
10: Let $R_c[s]$ be the last step in which a transmission $\overrightarrow{np}$ is assigned
11: n.minStep = s + 1; n.assignedSteps = 0
12: while ($n.assignedSteps < W_{q,n}$)
13: if $\overrightarrow{np}$ does not conflict with any transmission $\overrightarrow{ab} \in R_c[n.minStep]$
14: R_c[n.minStep] $= R_c$[n.minStep] $\cup \{\overrightarrow{np}\}$;
15: n.assignedSteps = n.assignedSteps + 1;
16: else n.minStep = n.minStep + 1

Fig. 1. The centralized planner.



Reversed Plan	a	d	b	c	e	m	j	g	h	o	Actual Plan
1	d										7
2	b	m									6
3	c	j	g			w					5
4	e	l		h			t	r			4
5			f	k	o	z			s		3
6					n					p	2
7										q	1

Fig. 2. Example transmission plan. The edges without arrows are bi-directional. The solid lines denote communication edges and the dotted lines interference edges.

3 and 4, respectively. When the planner completes assigning e's transmission to its parent ($\overrightarrow{ea}$), m becomes the highest-priority eligible node. Since $\overrightarrow{da}$ is assigned to step 1, the first step to which $\overrightarrow{md}$ may be assigned is 2. Since in $R_c[2]$ only $\overrightarrow{ba}$ is assigned and $\overrightarrow{md} \parallel \overrightarrow{ba}$, we assign $\overrightarrow{md}$ to step 2. A more interesting case occurs when f becomes the highest-priority eligible node. The earliest step to which $\overrightarrow{fb}$ may be assigned is 3, since the transmission of its parent's transmission $\overrightarrow{ba}$ is assigned to step 2. The planner first attempts to assign $\overrightarrow{fb}$ to steps 3 and 4, but fails. $\overrightarrow{fb}$ cannot be assigned to step 3 due to $\overrightarrow{gb}$. $\overrightarrow{fb}$ cannot be assigned to step 4 because $\overrightarrow{ea} \nparallel \overrightarrow{fb}$ due to the interference edge $\overrightarrow{eb}$. Since no transmission is currently assigned to $R_c[5]$, $\overrightarrow{fb}$ is assigned to it. The first stage of the planner continues to produce the transmission plan shown in the table. In the second stage, the planner reverses the order in which the steps are executed. Accordingly, the last step in the reversed transmission plan ($R_c[7]$) is the first step in the transmission plan ($T_c[1]$), the second to last step in the reversed transmission plan ($R_c[6]$) is the second step in the transmission plan ($T_c[2]$), and so on. The rightmost column of the table shows the step assignment in the actual transmission plan T_c.

B. The Scheduler

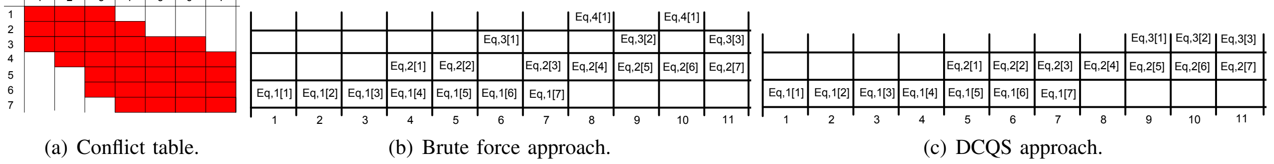
In this subsection, we first describe how to construct a global conflict-free schedule. We then present an efficient local scheduling algorithm. For clarity, we initially assume that all queries belong to a single query class. Consequently, all query instances are executed according to the same transmission plan. Next, we extend our solution to handle multiple query classes.

Definitions and notation. Each query instance executes an *instance* of the transmission plan. We use $E_{q,k}[s]$ to denote the set of transmissions assigned to step s of $I_{q,k}$'s instance of a transmission plan. We say that two steps of query instances $I_{q,k}$ and $I_{q',k'}$ are *conflict free* $E_{q,k}[s] \parallel E_{q',k'}[s']$ if all pairs of transmissions in $T_c[s] \cup T_{c'}[s']$ are conflict free. We also use the notation $E_{q,k}[s] \nparallel E_{q',k'}[s']$ to denote that the two steps conflict with each other. The schedule should have the following properties: (1) All steps scheduled in a slot are conflict-free. (2) The relative order of the steps of the same query instance is preserved: if step $E_{q,k}[s]$ is scheduled in time slot i, step $E_{q,k}[s']$ is scheduled in slot i' and $s > s'$ then $i > i'$. This ensures that the precedence constraints required by aggregation are enforced.

The Brute Force Approach. Let us consider a brute-force way to dynamically determining what steps should be scheduled in the same slot. We say step $E_{q,k}[s]$ is *ready* if $E_{q,k}[s-1]$ has been executed. The first step $E_{q,k}[1]$ is ready when the query instance $I_{q,k}$ is released at time $R_{q,k}$. Intuitively, the brute force approach schedules in each slot multiple conflict-free and ready steps. Priority is given to executing steps in the transmissions plans of query instances with earlier release times. To determine what steps may be scheduled in a slot, we need to know if any two steps in the transmission plan conflict. To facilitate this we construct a conflict table of size $L_q \times L_q$ that stores the conflicts between any pairs of steps in the transmission plan of the query class. Fig. 3(a) shows the conflict table of the transmission plan presented in Fig. 2. Fig. 3(b) shows the transmission schedule constructed using the brute force approach under saturation conditions when a query instance is released after the first step in the previous query instance was executed.

The brute force approach constructs the schedule as follows. Initially, $E_{q,1}[1]$ is the only step ready and it is scheduled in slot 1. In slot 2, the steps $E_{q,1}[2]$ and $E_{q,2}[1]$ are ready. However, the earliest slot when $E_{q,2}[1]$ may be scheduled is slot 4 since according to the conflict table $E_{q,2}[1] \nparallel E_{q,1}[1..3]$. So, in slot 4 we schedule $E_{q,1}[4]$ and $E_{q,2}[1]$. A more interesting case occurs when scheduling the steps in slot 6. In slot 6, $E_{q,1}[6]$ is scheduled since it has the earliest release time. $E_{q,2}[3]$ cannot be executed in slot 6 since $E_{q,2}[3] \nparallel E_{q,1}[6]$. However, $E_{q,3}[1]$ is ready and its execution does not conflict with $E_{q,1}[6]$. Therefore, it is also scheduled in slot 6. The process continues to construct the schedule presented in Fig. 3(c).

Unfortunately, the brute force approach is impractical due to its high computation and storage costs. The computation time for determining what steps to schedule in a slot is quadratic in the number of ready steps in all query instances that have been released. The memory cost for storing the conflict table is quadratic in the length of the transmission plan. As a result, the brute force approach cannot scale effectively



(a) Conflict table. (b) Brute force approach. (c) DCQS approach.

Fig. 3. Scheduling approaches.

in large networks with multiple concurrent queries running on resource constrained devices. To alleviate these problems we may trade some of the throughput in favor of reduced computational and storage costs. To this end, we impose the additional constraint that the execution of a query instance cannot be *preempted*. The execution of a query instance is not preempted if, once its first step is executed, the subsequent steps of its transmission plan are executed *without gaps* in the following slots. For example, in Fig. 3(b), the schedule constructed by the brute force approach does not meet this constraint because the execution of $I_{q,2}$ is preempted in slot 6.

Minimum inter-release time. We define the *minimum inter-release time*, Δ, as the minimum number of slots the execution of $I_{q,k}$ must be delayed after another query instance $I_{q',k'}$ starts executing such that the execution of $I_{q,k}$ and $I_{q',k'}$ are conflict-free. In other words, any two query instances are conflict free as long as their inter-release time is no lower than Δ.

Consider the execution of two consecutive query instances $I_{q',k'}$ and $I_{q,k}$ (from one or two queries). If the inter-release time between $I_{q,k}$ and $I_{q',k'}$ is δ and the execution of query instances cannot be preempted, then the steps $E_{q,k}[1]$ and $E_{q',k'}[\delta+1]$ are scheduled in the same slot of the transmission schedule. Hence, δ must be selected to ensure that $E_{q,k}[1] \parallel E_{q',k'}[\delta+1]$. However, the execution of $I_{q,k}$ may start in any slot after δ steps in the transmission plan of $I_{q',k'}$ are executed. Therefore, we must guarantee that $E_{q,k}[1]$ does not conflict with $E_{q',k'}[\delta+1]$ *and* any of the subsequent slot executions i.e., $E_{q,k}[1] \parallel E_{q',k'}[\delta+i+1]$ for all $i \in [0, L_c-\delta-1]$, where L_c if the length of the transmission plan of query class c. Δ is the smallest number such that the execution of any step $E_{q,k}[s]$ does not conflict with $E_{q',k'}[s+\delta+i+1]$ where $s \leq L_c$ and $i \in [0, L_c - s - \delta - 1]$. Thus, the minimum inter-release time is:

$$\Delta = min_{\delta \in [1, L_c]}(E_{q,k}[s] \parallel E_{q',k'}[s+\delta+i+1]) \\ \forall i \in [0, L_c - s - \delta - 1], s \leq L_c \quad (1)$$

The Scheduler. Each node employs a *local* scheduler that schedules the transmissions of all query instances. The scheduler maintains a queue of all query instances that have been released and not completed. The query instances are ordered by their release times. The scheduler starts executing the first step of the first query instance $I_{q,k}$ in the queue if the minimum inter-release time constraint is satisfied: if $I_{q',k'}$ is the last query instance executed and its first step was scheduled in slot i', then the scheduler executes the first step of $I_{q,k}$, $E_{q,k}[1]$, in slot $i' + \Delta$. The execution of a query instance cannot be preempted: the step $E_{q,k}[s]$ is executed in the next slot after $E_{q,k}[s-1]$ was executed until the query instance is completed.

The scheduler executes the current step as follows: (i) If there exists a transmission in the step in which the local node is the sender, the scheduler submits the packet of the query instance to the MAC layer for transmission. (ii) If there exists a transmission in the step for which the local node is the receiver, it keeps the radio active to receive the incoming packet. If the local node is neither a sender nor a receiver in any of the transmissions in the current step, it turns the radio off until the next slot in which the local node is a receiver or sender. As a result, the radio is active only when it is needed for sending or receiving packets resulting in maximum energy savings.

The scheduler is simple and efficient making it feasible to run it on resource-constrained devices. The time to determine what steps are scheduled in a slot is $O(1)$. Consequently, a node may construct the transmission schedule dynamically. Second, the memory cost of the algorithm is also significantly lower than the brute force approach. The scheduler maintains only the minimum inter-release time and a queue of query instances.

Fig. 3(c) presents the schedule constructed when the minimum inter-release time Δ is 4 slots. The constructed schedule has slightly lower throughput than the one constructed using the brute force approach. This illustrates our decision to trade-off throughput to reduce the memory and processing costs. However, our simulation results show that DCQS still achieves significantly higher throughput than existing solutions. (see Section VI).

Analysis. In the following we prove three properties of the DCQS scheduler. First, we prove that the scheduler never schedules conflicting transmissions in the same slot. Second, we analyze the network capacity in terms of query completion rate under DCQS. This result is important because it enables us to prevent drastic performance degradation using rate control (as described in Section V-B). Finally, we characterize the energy consumption of a node.

Theorem 1: The scheduler executes conflict-free transmissions in all slots.

Proof: Consider the scheduler constructing a schedule for the following sequence of query instances $I_{q_1,k_1}, I_{q_2,k_2}, I_{q_2,k_2} \cdots$. We will prove that the scheduler does not execute in a slot conflicting steps in the execution of query instances I_{q_1,k_1} and I_{q_i,k_i}. Consider the case when I_{q_1,k_1} and I_{q_i,k_i} overlap. Let s_1 and s_i be the steps in the plans of I_{q_1,k_1} and I_{q_i,k_i} that the scheduler assigns in the same slot. Since the scheduler enforces a minimum

inter-release time of Δ between consecutive query instances then $s_1 - s_i \geq (i-1) \cdot \Delta \geq \Delta$ because $i \geq 2$ Thus, the scheduler executes conflict-free transmission in any slot.

Theorem 2: The maximum query rate of DCQS is $\frac{1}{\Delta \cdot slotSize}$ where $slotSize$ is the size of a slot in seconds.

Proof: A query instance can be released every Δ slots. Therefore the maximum query completion rate that can be achieved is $\frac{1}{\Delta \cdot slotSize}$.

A network running DCQS has predictable power consumption. DCQS keeps a node n awake only when it or one of its children are scheduled to transmit a data report. Otherwise, node n is scheduled to sleep. Therefore, the power consumed by n to execute a query q is:

$$Pwr_n(q) = \frac{1}{P_q} \cdot (Pwr_{recv} \cdot \sum_{c \in child(n)} W_q[c] + Pwr_{send} \cdot W_q[n]) \quad (2)$$

The rate of query q is $\frac{1}{P_q}$. $W_q[c]$ is the maximum number of packets a child c transmits to n to satisfy the workload demand of q. $W_q[n]$ is the maximum number of packets transmitted by n to its parent. Pwr_{recv} and Pwr_{send} is the power consumed in receiving and transmitting a packet, respectively. Based on Equation 2 the network lifetime may be computed.

Handling Multiple Query Classes. We now extend our scheduler to the case when there are multiple query classes. To this end, we must refine the definition of minimum inter-release time to accommodate the case when query instances have different transmission plans. We define $\Delta(c, c')$ as the minimum number of slots a query instance of class c must wait after a query instance of class c' started its execution such that there are no conflicts. Note that Δ is not commutative.

Given the minimum inter-release times between any ordered pairs of query classes, the scheduler needs to control the inter-release times of two consecutive query instances based on their query classes. We note that the storage cost of multiple class scheduler is quadratic in the number of query *classes*, since we must store the minimum inter-release time of each ordered pair of query classes. However, as discussed in Section III-A, usually only a small number of query classes are used in practice.

C. Distributed Planner

In this subsection we present a distributed planner which uses only neighborhood information in constructing transmission plans. Specifically, a node knows only its adjacent communication and interference edges (e.g., by executing the RID protocol [13]). We say that a node is in n's *one-hop neighborhood* if there is a communication or interference edge between it and n. n's two hop neighborhood includes n's one-hop neighbors and their one-hop neighbors. After running the decentralized planner a node knows its *local plan* which contains the step assignments for its two-hop neighbors.

To construct a local transmission plan, a node communicates only with its one-hop neighbors. However, some of the neighbors may lie outside the node's communication range. A routing algorithm or limited flooding may be used to communicate with these nodes over multiple hops. Alternatively, the transmission power of the sender may be increased to reach the one-hop neighbors in a single hop. Like RID [13], our implementation in the simulations adopted the latter approach.

A node n constructs a transmission plan in three stages: plan formulation, plan dissemination, and plan reversal. The formulation stage starts when a node n becomes the highest-priority eligible node in its one-hop neighborhood. When this occurs, n broadcasts a *Plan Request* packet to gather information about transmissions which have already been assigned steps. To construct a conflict-free plan, n must know the steps in which its two-hop neighbors with higher priorities were assigned. Upon receiving the *Plan Request* from n, each one-hop neighbor checks if there is a node in its own one-hop neighborhood that has a higher priority than n. If no such node exists, the receiver responds with a *Plan Feedback* packet containing its local plan. Otherwise, the node does not reply. After a time-out, node n will retransmit the *Plan Request* to get any missing *Plan Feedback* from its one-hop neighbors. Since all *Plan Feedback* are destined for n, to reduce the probability of packet collisions, nodes randomize their transmissions in a small window. Once n receives the *Plan Feedback*, it has sufficient information to assign its transmissions to its parent using the same method as the centralized planner. In the second stage, n disseminates its local plan to its one-hop neighbors via a *Plan Send* packet. Upon receiving a *Plan Send*, a node updates its plan accordingly and acknowledges its action via a *Plan Commit* message.

To ensure that DCQS constructs a conflict-free schedule, neighboring nodes must have consistent transmission plans. We note that the distributed planner achieves this objective through retransmission when needed. If a *Plan Feedback* message from some neighbors are lost, node n assumes that a higher priority node has not yet been scheduled and retransmits the *Plan Request* until it has received Plan Feedback from each neighbor or reached the maximum number of re-transmissions. Similarly, during the plan dissemination stage, node n retransmits the plan until all its neighbors acknowledge the correct reception of its *Plan Send* via the *Plan Commit* message.

Finally, the planner reverses the transmission plan. To do this, a node must know the length of the global transmission plan. We take advantage of the routing tree and data aggregation to compute the length of the transmission plan as follows. A node computes the length of its local transmission plan length based on the maximum step number in which a transmission/reception is assigned. The node aggregates its local length of the transmission plan with that of its children by taking the maximum of the two. The result is sent to its parent. At the base-station, the plan length may be computed. The root then uses the routing tree to disseminate this value to each node. Upon receiving the plan length a node reverses its transmission plans.

Distributed computation of minimum inter-release times. An important feature of DCQS scheduler is that they

are localized. Node will independently construct the same transmission schedule as long as they have a consistent view of the query parameters, local transmission plans and the minimum inter-release time. The query descriptions are disseminated to all nodes in the network so that they can determine whether they should respond to a query. We now enhance the distributed planner to compute the minimum inter-release times.

The key to compute the minimum inter-release time in a distributed manner lies in the observation that a node may compute its local value for the minimum inter-release time based on the its local transmission plan and its local knowledge of the IC graph according to (1). The minimum inter-release time of the global plan is the maximum of the minimum inter-release times of the local plans. This suggests that, similar to the length of the transmission plan, the global minimum inter-release time can be computed using in-network aggregation. In fact, the two may be computed concurrently. Once the aggregation process is complete, the root can compute the length, and minimum inter-release time of the transmission plan and then disseminate them to all nodes in the network.

V. Handling Dynamics

A. Dynamic Workload

DCQS can efficiently adapt to changes in the workload including arrival, deletion, and rate change of queries. Consider the case where a user issues a new query. The query service disseminates the query type and parameters to all nodes in the network. Next, DCQS checks if a transmission plan for the issued query was previously constructed. If no transmission plan was constructed, DCQS uses the decentralized planner to compute a new transmission plan and the minimum inter-release times. DCQS isolates the execution of current queries from the setup of new queries by periodically reserving slots for protocol maintenance. During the protocol maintenance slots, the planner computes the transmission plan and minimum inter-release times. Once they are computed, the scheduler has sufficient information to construct a conflict-free schedule which accounts for execution of the new query.

If a query from the same class was previously issued, a transmission plan for that class has already been constructed. As previously mentioned, queries from the same class share the same transmission plans and minimum inter-release time. Since usually there are only a small number of query classes, it is likely that DCQS already computed the transmission plan and minimum inter-release times of a class. In this case, DCQS can handle the new query without any additional overhead. Similarly, DCQS can also handle query deletions and rate changes of existing queries without any overhead.

B. Preventing Overload

A key advantage of DCQS is that it has a known capacity bound in terms of the maximum query completion rate, as shown in Theorem 2. This bound enables DCQS to easily detect overload conditions which obviates the need for complex congestion control mechanisms. When the user issues a query, DCQS computes the total query rate $\sum_q \frac{1}{P_q}$ (including the query to be issued). If the total query rate is smaller than the network capacity, then the query can be admitted to the network. Otherwise, we consider the following two options. First, the user may be notified that the query will not be executed because the network capacity would be exceeded. Second, DCQS may reduce the rates of existing queries to allow the new query to be executed. For example, a simple rate control policy is to reduce the rates of all queries proportionally by multiplying their rates by $\alpha = (\frac{\sum_q P_q}{\Delta \cdot slotSize})$. This rate control policy is used in our simulations. As discussed in the previous section, DCQS may modify the period of a query without recomputing the transmission plan or minimum inter-release times. Therefore, the only overhead is to disseminate the new rates of the existing queries to the network.

C. Handling Topology Changes

We now describe how DCQS handle topology changes due to node or link failures. For DCQS to detect topology changes, we increase the slot size to allow a parent to acknowledge the correct reception of a data report from its child. A child can detect the failure of its parent or their link if it does not receive ACKs from its parent for several consecutive transmissions. A parent detects a child failure if it does not receive any data reports from that child for a number of query periods.

For all nodes to maintain a consistent schedule, DCQS must ensure the following: (1) the *two-hop neighbors* of a node have a consistent view of its local transmission plan, which dictates when the node transmits and receives data reports; (2) *all nodes* have consistent information about the length of the transmission plans and the minimum inter-release times. In response to topology changes, the routing tree must be adjusted. Consider the case when a node n detects that its parent p failed and, as a result, it must select a new parent p'. This entails the planner assigning a step in the transmission plan for $\overrightarrow{np'}$, while the step in which the transmission $\overrightarrow{np}$ is scheduled must be reclaimed. If $\overrightarrow{np'}$ can be assigned to the step in which $\overrightarrow{np}$ was scheduled or a different step *without conflicts* then DCQS only needs to update the local transmission plan. This involves node n disseminating its updated transmission plan to its two-hop neighbors. If this is not possible, then DCQS must start recomputing a new transmission plan. We note that the computation of the new plan affects only nodes with lower priority than n. If during the computation of the plan either the minimum inter-release times or the transmission plan lengths are modified, this information must be disseminated to all nodes in the network. Consider the case when a child node c of n failed. In this case, the step in which c is assigned should be reclaimed. To reduce the overhead, DCQS reclaims such slots only when other topology changes occur.

To reduce the cost of handling topology changes, we now describe an approach to constructing robust transmission plans that can tolerate some topology changes. To handle this we change the mechanism used to adapt the routing tree in response to link or node failure. We allow a node to change

its parent in the routing tree as long as the new parent is selected from a predefined *set of potential parents*. Our goal is to construct transmission plans that are insensitive to a node changing its parent under the constraint that the new parent is in the set of potential parents. To this end, we introduce the concept of *virtual transmissions*. Although node n actually transmits to a single potential parent, we construct the transmission plan and compute the minimum inter-release times as if n transmits to *all* potential parents. We trade-off some of the throughput in favor of better tolerating topology changes. This trade-off is similar to other TDMA algorithms designed to handle topology changes [16][17]. To implement this strategy in DCQS, we refine the definition of conflict free transmissions. We define a *virtual transmission* of node n, $v(n)$, as the set of transmissions between n and its potential parents. We say that two virtual transmissions $v(n)$ and $v(m)$ are conflict free if there is no pair of transmissions, $\overrightarrow{ab} \in v(n)$ and $\overrightarrow{cd} \in v(m)$, such that $\overrightarrow{ab}$ and $\overrightarrow{cd}$ conflict.

VI. Experiments

We implemented the distributed version of DCQS in NS2. Our simulation settings are similar to 802.11b radios. This is because we are interested in high data rate applications such as structural monitoring and preventive maintenance. Sensor nodes used for such applications often adopt high-bandwidth radios. For example, the DuraNode [4] designed for structural health monitoring is equipped with an 802.11b WLAN card. Some applications may adopt hierarchical network architectures that integrate high-bandwidth wireless networks with traditional low-bandwidth sensor networks. For instance, Intel's sensor network deployed for preventive maintenance employs an 802.11 ad hoc network with a lower-tier mote network [4].

In our simulations, the network bandwidth is 2Mbps. The communication range is 125m. The power consumed for transmitting and receiving a packet is 1.6W and 1.4W, respectively. The size of a packet is 2040 bytes, of which 20 bytes are used for packet headers. Based on packet size and bandwidth we computed the slot size to be 8.16ms. The queue size is 10 packets. Each experimental run takes 200s.

In the beginning of the simulation we construct the IC graph and the routing tree. The IC graph is constructed similarly to the method described in [13]. The routing tree is constructed as follows. The node closest to the center of the topology is selected as the base-station and is the root of the routing tree. The base station initiates the construction of the routing tree by flooding setup requests. A node may receive setup requests from multiple nodes and selects the node with the latest depth as its parent. Each node in the routing tree performs in-network aggregation when executing a query. We assume that each aggregated data report fits in a single packet. The queries issued involve all nodes in the network. In all experiments the queries belong to the same query class.

For performance comparison we ran two baselines: 802.11b[18] and DRAND[19]. 802.11b is representative CSMA/CA-based protocol, while DRAND is a representative node-scheduling TDMA protocol. Unlike DCQS, DRAND does not account for the interference relationships among nodes. Hence, the schedule it constructs may still result in collisions. To avoid this problem, we modified DRAND to treat the interference edges in the IC graph as communication edges. We augmented DRAND with a sleep-scheduling policy: a node is kept awake if DRAND schedules it or one of its children to transmit; otherwise, the node is put to sleep to conserve energy.

We evaluated the performance of DCQS and the baselines according to four metrics: query completion rate, query fidelity, query latency, and energy efficiency. The query completion rate is defined as the number of query instances completed per second during a run. A query instance is complete if the base station received at least a data report from its children. During the simulations data reports may be dropped preventing some sources from contributing to the query result. We quantify the quality of a query result using the query fidelity metric. The query fidelity is the ratio of the number of sources that contributed to the query result received by the base station and the total number of sources. We measure the energy efficiency by dividing the total energy consumed in a run by the total number of data reports that contributed to the query results.

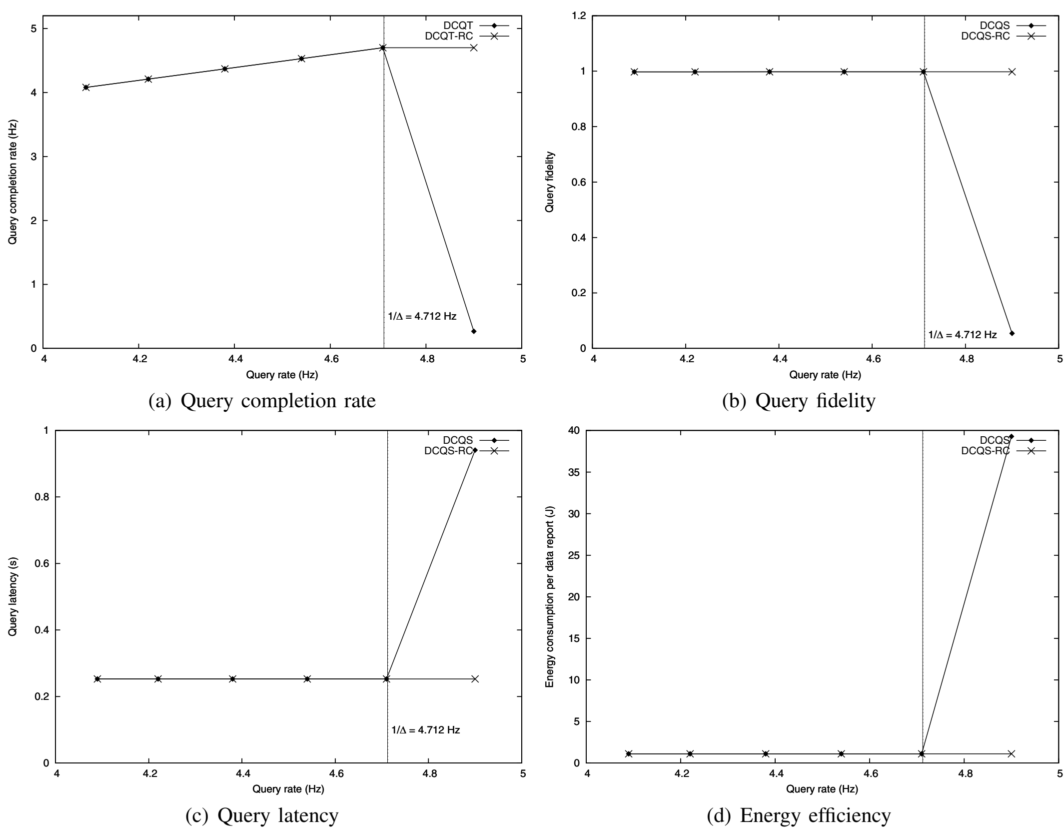
We start by evaluating the performance of DCQS when there is a single query executed in the network. This experiment is designed to validate the analytical results on network capacity and power efficiency presented Section V. The next experiment compares the performance of DCQS to that of the baselines when multiple queries are executed in the network and the workload is varied by changing the period of the queries. The last experiment shows the scalability of DCQS compared to that of DRAND when the number of nodes in the network is varied.

A. Single Query

The first experiment is designed to validate our capacity result and to show the effectiveness of our rate control policy. We ran DCQS with both the rate control policy and without it. DCQS-RC denotes DCQS running in conjunction the rate control policy.

A single query is executed in the network. The results are obtained from a topology of size 675m×675m. The topology is divided into grids of 75m×75m. In each grid a node is placed at random. Under these settings, DCQS constructed a transmission plan with $\Delta = 26$ slots. According to Theorem 2 the maximum query rate that DCQS may support is $\frac{1}{26 \cdot 8.16ms} = 4.7Hz$. The vertical lines in Fig. 4 indicates the network capacity. To validate the capacity bound the query rate is varied between 4.1Hz to 4.9Hz in increments of 0.1Hz. Each result obtained in this experiment is from a single run. We chose to present results from a single run, because for different topologies DCQS constructs transmission plans with different Δ values.

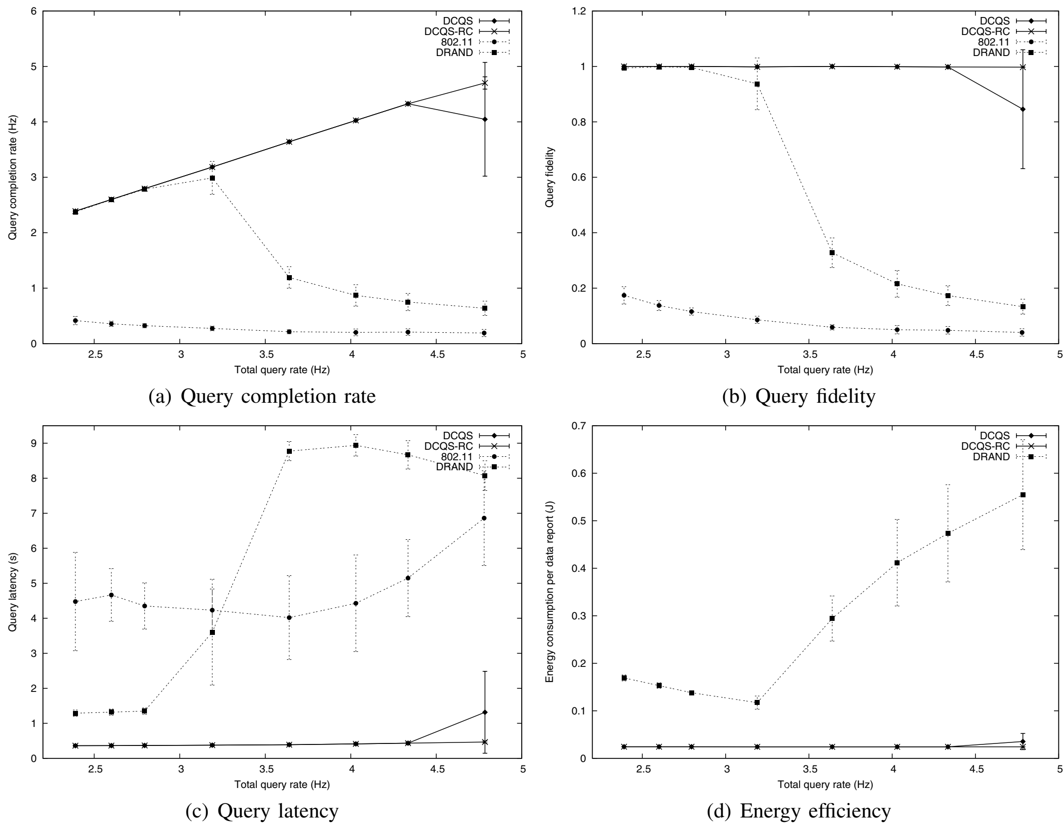
Fig. 4(a) shows the query completion rate. We observe that the increase in query rate is matched by an increase in the query completion rate until the network capacity is reached.



(a) Query completion rate (b) Query fidelity

(c) Query latency (d) Energy efficiency

Fig. 4. Throughput, fidelity, delay and energy for a single query when the query rate is varied.



(a) Query completion rate (b) Query fidelity

(c) Query latency (d) Energy efficiency

Fig. 5. Throughput, fidelity, delay and energy for a four queries are executed and their base rate is varied.

When workload exceeds the network capacity, the performance of DCQS degrades drastically. As discussed in Section V-B rate control may be used to avoid triggering the capacity bottlenecks. As shown in the figure, DCQS-RC which uses rate control maintains its good performance even when the offered load exceeds the network capacity.

Fig. 4(b) shows the data fidelity. DCQS provides 100% query fidelity up to the maximum query rate. This result shows that the schedule constructed by DCQS is conflict-free, validating the correctness of our algorithms. In addition, DCQS-RC, which uses the rate control policy, avoids the drop in query fidelity under overload conditions. A similar pattern may be observed in terms of delay as shown in Fig. 4(c). Both DCQS and DCQS-RC have similar latencies up to the network capacity. If the capacity is exceeded and the rate control is not used, the query latency increases sharply. In contrast, DCQS-RC is unaffected by the overload conditions.

Fig 4(d) shows the energy consumed per data packet. As predicted by Equation 2, when the data fidelity is 100%, the energy consumed per data packet is constant. This result validates our analysis of the network lifetime. This figure also indicates the advantage of rate control: DCQS-RC avoids the decrease in energy efficiency in overload conditions.

B. Multiple Queries

This set of experiments is designed to compare the performance of DCQS to that of the baselines under different workloads. The workload is generated by running four queries with different rates. The ratio of the rates of the four queries $Q_1 : Q_2 : Q_3 : Q_4$ is 8:4:2:1. We refer to Q_1's as the base-rate. We vary the workload by changing the base rate. The start time of the queries is spread evenly in an interval of 81.6ms. The topology setup is identical to the previous experiment. Each data point is the average of five runs. We also plot the 90% confidence interval for each point.

Fig. 5(a) shows the query completion rate when the total query rate is varied. A common trend may be observed: the protocols match the increase in the total query rate up to their respective maximum capacity and then their performance plummets. The lowest throughput is obtained by 802.11 protocol. The reason for this outcome is that the capacity of 802.11 is exceeded in all tested settings. This is because contention based protocols perform poorly under heavy workloads. DRAND outperforms 802.11. It achieves a maximum query completion rate of 3.11Hz when the total query rate is 3.19Hz. DCQS and DCQS-RC clearly outperform DRAND, DCQS-RC achieving maximum query rate of 4.7Hz which is about 47% higher than DRAND. This result is attributed to the fact that DRAND assigns slots to nodes fairly. Fair allocation is unsuitable for queries in WSNs because different nodes may have different communication load. For example, a node with more children need to receive more messages per each query instance. As in the previous experiment, DCQS-RC maintains its good performance even under overload conditions. This shows that our rate control policy works not only in the single-query case, but also in the multi-query case.

Fig. 5(b) shows the query fidelity of the protocols. As expected, 802.11 has poor query fidelity, whereas the TDMA protocols perform much better. DRAND maintains its high query fidelity up to its maximum query completion rate of 3.11Hz after which it plummets. The reason for this is that the transmission queues fill-up and packets are dropped. In contrast, DCQS-RC maintains 100% fidelity for all tested query rates.

Fig. 5(c) shows the query latency of the presented protocols. Even when the query rate is low, DCQS has significantly better query latency than DRAND. For example, when the query rate is 2.39Hz, DRAND has a query latency of 1.31s. In contrast, DCQS has a latency of 0.38s which is 70% lower than that of DRAND. DRAND has a long query latency because at each hop a node may need to wait for the duration of an entire frame before it may transmit its packet to the parent. In contrast, DCQS accounts for the precedence constraints introduced by data aggregation when constructing the transmission plans. This results in a significant reduction in the query latency.

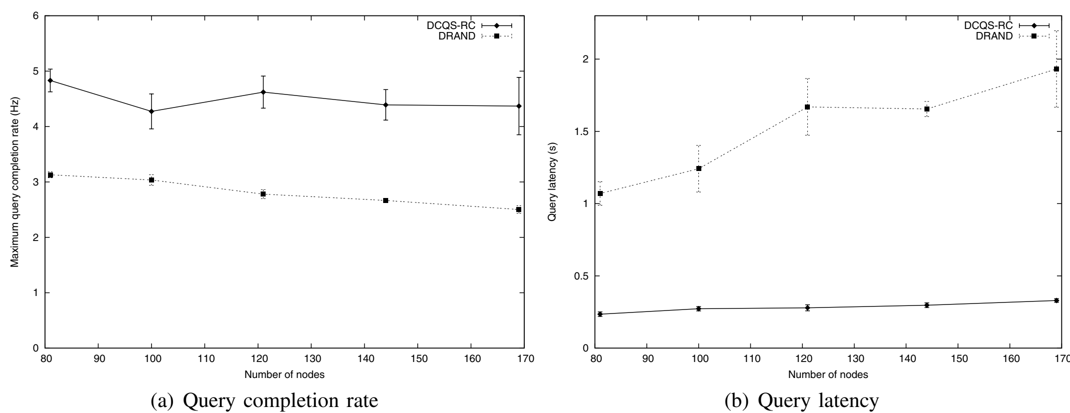
Fig. 5(d) presents the energy consumed per data report. We observe an improvement in the energy consumed by DRAND with the query rate up to 3.19Hz. The performance drastically degrades after this point due to packet loses. Even under light loads, DCQS performs better than DRAND in terms of energy. The reason is that DRAND must remain awake when a child is scheduled to transmit even if the child node has no packets to transmit. In contrast, DCQS takes advantage of temporal properties of the workload to wake-up nodes only when necessary. As observed in the previous set of experiments, the energy consumption per packet of DCQS is constant.

This set of experiments indicates that DCQS significantly outperforms both 802.11 and DRAND in all the considered metrics. Two factors contribute DCQS's high performance. First, the planner constructs transmission plans based on a heuristic that accounts for the precedence constraints introduced by data aggregation. This is highly effective in reducing message latency. Second, the scheduler overlaps the execution of multiple query instances to increase the query completion rate.

C. Scalability

The last set of experiments is designed to evaluate the scalability of DCQS and DRAND. To this end we constructed five topologies with an increasing number of nodes by increasing the deployment area and keeping the node density constant. All topologies are squares with edges of size 675m, 750m, 825m, and 900m. Each area is divided into grids of size 75m$\times$75m. In each grid, a node is placed at random. Each data point is the average of five runs and plot the 90% confidence intervals.

Fig. 6(a) shows the maximum completion rate that may be achieved by DCQS and DRAND for each topology. The maximum query completion rate of DCQS was computed theoretically and then verified experimentally. To determine the maximum query completion of DRAND we increased, the query rate until the query fidelity dropped below 90%. This is



(a) Query completion rate (b) Query latency

Fig. 6. Maximum query completion rates and query latency obtained when the number of nodes is increased but the density is kept constant.

reasonable since the DRAND drops packets only if the queue of a node fills up. When the topology has 81 nodes, DCQS outperforms DRAND by 54%. When the topology contains 169 nodes, the performance gap between the two protocols increases, DCQS outperforming DRAND by 74%.

Fig. 6(b) shows the query latency at the maximum query rate supported by each protocol. The query latency of DRAND increases with the number of nodes. In contrast, the query latency of DCQS increases only slightly. The key to understanding this result is that the one-hop delay of DRAND is significantly larger than that of DCQS. The one-hop delay corresponds to the slope of the shown curves. When using DRAND, a node often needs to wait for the entire length of a frame before it may transmit its packet. In contrast, DCQS has low one-hop delays. Two factors contribute to this. First, DCQS organizes its transmissions to account for the precedence constraints introduced by data aggregation. Second, DCQS executes multiple query instances concurrently. As such, the time until the query instance starts being executed is reduced.

VII. Conclusions

This paper presents DCQS, a novel query scheduling protocol specifically designed for query services in wireless sensor networks. DCQS features a planner and a scheduler. The planner reduces query latency by constructing transmission plans based on the precedence constraints for in-network aggregation. The scheduler improves throughput by overlapping the transmissions of multiple query instances concurrently while enforcing a conflict-free schedule. Our simulation results show that DCQS achieves query completion rates at least 47% higher than DRAND, and query latencies at least 70% lower than DRAND. Furthermore, DCQS is designed to handle dynamic changes in workloads and network topologies efficiently. Finally, DCQS provides an analytical network capacity bound that enables it to prevent network overhead through rate control. In the future we plan to integrate DCQS with a query service to support high data rate applications in wireless sensor networks.

VIII. Acknowledgement

This work is funded in part by NSF under ITR grants CCR-0325529 and CCR-0325197.

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 *SenSys*, 2005.

[2] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, “Wireless sensor networks for habitat monitoring,” in *WSNA*, 2002.

[3] K. Chintalapudi, J. Paek, O. Gnawali, T. Fu, K. Dantu, J. Caffrey, R. Govindan, and E. Johnson, “Structural damage detection and localization using netshm,” in *IPSN*, 2006.

[4] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis, “Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea,” in *SenSys*, 2005.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TAG: a tiny aggregation service for ad-hoc sensor networks,” in *OSDI*, 2002.

[6] I. Cidon and M. Sidi, “Distributed assignment algorithms for multihop packet radio networks,” *IEEE Trans. Comput.*, vol. 38, no. 10, 1989.

[7] A. Ephremides and T. Truong, “Scheduling broadcasts in multihop radio networks,” *IEEE Transactions on Communications*, vol. 38, no. 4, 1990.

[8] R. Ramaswami and K. K. Parhi, “Distributed scheduling of broadcasts in a radio network,” in *INFOCOM*, 1989.

[9] E. Arikan, “Some complexity results about packet radio networks,” *NASA STI/Recon Technical Report N*, vol. 83, Mar. 1983.

[10] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, “Energy-efficient collision-free medium access control for wireless sensor networks,” in *SenSys*, 2003.

[11] L. Bao and J. J. Garcia-Luna-Aceves, “A new approach to channel access scheduling for ad hoc networks,” in *MobiCom ’01*, 2001.

[12] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “Tinydb: an acquisitional query processing system for sensor networks,” *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[13] G. Zhou, T. He, J. A. Stankovic, and T. F. Abdelzaher, “RID: radio interference detection in wireless sensor networks,” in *INFOCOM*, 2005.

[14] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks,” in *SenSys*, 2003.

[15] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, “The flooding time synchronization protocol,” in *SenSys*, 2004.

[16] I. Chlamtac and A. Farago, “Making transmission schedules immune to topology changes in multi-hop packet radio networks,” *IEEE/ACM Trans. Netw.*, vol. 2, no. 1, 1994.

[17] J.-H. Ju and V. O. K. Li, “An optimal topology-transparent scheduling method in multihop packet radio networks,” *IEEE/ACM Trans. Netw.*, vol. 6, no. 3, 1998.

[18] “Wireless LAN medium access control (MAC) and physical layer (PHY) specifications,” *IEEE Standard 802.11*.

[19] I. Rhee, A. Warrior, J. Min, and L. Xu, “DRAND: Distributed randomized TDMA scheduling for wireless ad hoc networks,” in *MobiHoc*, 2006.