GreenWave Routing Trees for Wireless Sensor Networks

Fikret Sivrikaya
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
Email: sivrif@cs.rpi.edu

Bülent Yener
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
Email: yener@cs.rpi.edu

***Abstract*— Contention-free MAC protocols have the desired energy-saving characteristics for wireless sensor networks by avoiding collisions and minimizing retransmissions in the wireless medium [2] [4] [24] [25] [26]. Most of the contention-free MAC protocols split the channel in the time domain, by assigning each node a time slot for use in transmission, which is unique in its neighborhood. In such protocols, the time-multiplexed communication introduces extra delay on the packets when relayed by intermediate nodes. In this paper, we consider the delay optimal routing problem on sensor networks with such MAC protocols. We propose algorithms which construct routing trees rooted at sink nodes to route data to and from sensor nodes. First, we consider routing with data fusion, and present our GreenWave routing idea. We show that our algorithm significantly reduces the end-to-end delay when compared to routing over the shortest-hop paths. We also present a comparison of GreenWave routing over a contention-free MAC protocol to the shortest-path routing over the IEEE 802.11 MAC protocol, accompanied by an end-to-end delay analysis of 802.11. As a side result, we show that there is roughly a two-fold decrease in 802.11 performance when hidden terminals are taken into account. Moreover, we observe that a contention-free MAC protocol may outperform 802.11 with the help of GreenWave routing. Second, we consider routing without data fusion, by taking into account the effect of congestion along the paths on the end-to-end delays. We provide a quadratic integer programming (QIP) formulation of the problem, and present a lower bound and a heuristic algorithm to bound the optimal solution. Numerical results show that the results obtained by the heuristic are close to the lower bounds.**

I. Introduction

Wireless sensor networks receive significant research focus as advances in technology made feasible the design and deployment of tiny sensors with radio communication capabilities. Many application scenarios for sensor networks are projected such as environment monitoring, disaster recovery, emergency and military applications. The characteristics of a sensor network may vary greatly according to the specific application scenario. However most sensor networks share some basic features. First, the lifetime of the network depends on the limited energy available at individual sensor nodes. Second, cheap sensor nodes are usually deployed in large amounts. Third, the sensor devices may fail frequently due to their low-cost nature. From the designer's perspective these features give rise to the issues of energy saving, scalability, and fault-tolerance, respectively.

Medium Access Control (MAC) protocols determine how nodes in a sensor network access the shared wireless medium, and they play a crucial role on the overall energy utilization in the network. Collision of radio packets in the wireless medium has usually a destructive effect on the packets and necessitates retransmission at the senders. Severe energy wastage may be caused by frequent collisions in a sensor network. Many recent work argue that contention-free MAC protocols are more suitable for energy-limited sensor networks [2] [4] [24] [25] [26], and they provide channel access scheduling algorithms, mostly by using a time-multiplexed (TDMA-based) technique. In this approach time is divided into frames and frames are further divided into time slots. Each node is allocated a time slot in its frame that it can use for collision-free transmission.

The collision-free transmission by the use of a TDMA-based MAC protocol comes with the price of increased latency, especially in multi-hop communication environments such as sensor networks. From the source to its destination, a packet suffers a delay at each intermediate node due to the fact that a node has to wait for its time slot before it can relay the packet to the next hop. However if the routing path is carefully chosen for a source-destination pair, the delay can be minimized. The movement of packets in a network could be regarded as vehicle traffic in a city. Then each relaying (intermediate) node along the packet's route might represent an intersection. The packet waits at relaying nodes for the allocated time slot, just as a vehicle waits at intersections for the green light. The *green-wave* idea in traffic regulation is widely used in large cities to allow continuous flow of traffic along the main routes.

Some sensor network applications utilize an energy-saving technique, called *in-network data fusion*, when packets are being relayed (routed) by intermediate nodes. In-network data fusion is the idea of aggregating data from multiple sources/sessions before relaying to the next hop so that the total communication load, hence the energy wastage, is reduced. However, it is not always possible to utilize this in some sensor network applications. In this paper, we consider both cases, and study the problem of minimizing the end-to-end communication delays for each case. Our contributions are two-fold. (i) First we propose *GreenWave routing*, an algorithm to construct end-to-end (sensor-to-sink) routes on a sensor network where data are fused along the routing

paths. In order to assess the performance of this protocol, we present an end-to-end delay analysis of IEEE 802.11 MAC protocol (accompanied by shortest-path routing) by extending and combining the results from previous work. As a side result, we also demonstrate the effect of hidden terminals on the performance of 802.11. (ii) We formulate the problem of delay optimal routing without data fusion as a quadratic integer programming (QIP). Then we provide a lower bound and an efficient heuristic, bounding the optimal solution to the QIP from above and below.

The rest of the paper is organized as follows. In the next section, we present an overview of some previous work related to ours. In section III, the assumptions and notations used in this paper are given. In section IV, we consider routing with in-network data fusion and present the GreenWave routing algorithm. Section V is devoted to end-to-end delay analysis of 802.11 protocol, which we use in section VI to compare the performance of GreenWave routing. We consider the no fusion case in section VII, in which we provide a QIP formulation, a lower bound algorithm, and a heuristic. Section VIII concludes the paper.

II. Related Work

MAC protocols, in general, fall into two broad classes: contention-based and contention-free. *Contention-based* MAC protocols are also known as *random access protocols*, requiring no coordination among the nodes accessing the channel. Colliding nodes back-off for a random duration and try to access the channel later again. IEEE 802.11 [1], the current standard for wireless networks, is also a contention-based MAC protocol. Most of the previous work on MAC protocols for wireless sensor networks are based on contention-based access and many of those propose improvements or modifications on the 802.11 protocol. On the other hand, frequent retransmissions in contention-based protocols may lead to quick depletion of the limited energy available at sensor devices. Hence researches have currently focused on developing distributed algorithms that schedule the channel access among nodes in a sensor network so that collisions are prevented before they occur [2] [4] [24] [25] [26]. These are known as the *contention-free* (CF) MAC protocols, mostly based on a distributed implementation of the TDMA (Time Division Multiple Access) protocol.

In [2] we proposed a scalable, distributed and asynchronous CF MAC protocol for assigning time slots to nodes in a sensor network, such that collisions are eliminated. A randomized protocol is given in which nodes try to negotiate with neighbors for conflict-free time slot allocation in a completely distributed manner. Another recent work, [4], on the other hand, uses a hierarchical structure, and the time slot allocation is done by the *leaders* in clusters. DE-MAC protocol, [24], uses the TDMA technique together with periodic listen and sleep to avoid major sources of energy wastage. In [25], energy-aware routing and MAC protocols are presented, which are cluster based algorithms controlled by the gateway node of each cluster. A self-organizing algorithm is given in [26] to schedule the activation of links in the network, which is a combined approach for constructing a connected network structure and conflict-free communication schedule.

Routing protocols for multi-hop ad-hoc networks can also be classified into two groups: proactive (table-driven) and reactive (demand-driven). Proactive protocols maintain routing tables to keep up-to-date routes from each node to every other node in the network. *Proactive Destination-Sequenced Distance-Vector Routing (DSDV)* [13] is a proactive algorithm based on the Bellman-Ford algorithm [5]. Reactive protocols, on the other hand, do not maintain routing tables, instead routes are discovered when needed by a source node. The two most well-known reactive routing protocols are the *Dynamic Source Routing (DSR)* [14] and *Ad-Hoc On-demand Distance-Vector Routing (AODV)* [15]. Route request and route maintenance are the basic operations in these protocols to explore and maintain routes. A thorough survey of routing protocols for wireless ad-hoc networks can be found in [16].

Finally we cite here some of the existing analysis on the IEEE 802.11 protocol, as we use it as a basis for our performance results. There are innumerable analytic work on the performance of 802.11 in a single-hop environment. Among those, [7] provides a considerably simple, yet accurate model to study the saturation throughput of 802.11 under ideal channel conditions and without hidden terminals. Carvalho et al. [19] generalizes this model to include the physical layer affects, and provides a generic modelling framework for the analytical study of medium access control (MAC) protocols for multihop ad hoc networks, where each node can be modelled and studied individually. The analysis of 802.11 is provided as a case study in this work. Similarly [20] provides the theoretical analysis of a generic Carrier Sense Multiple Access (CSMA) protocol with collision avoidance, and the proposed model is applied to the 802.11 protocol as a case study. In [17], the authors claim that the IEEE 802.11 MAC protocol is inefficient in multi-hop wireless ad-hoc networks, which is validated through simulations. They also consider the key changes required to adapt the IEEE 802.11 MAC protocol for multi-hop wireless networks. Similarly [18] state that the 802.11 protocol is not intended to support wireless mobile ad hoc networks, and by presenting several problems encountered in TCP (transmission control protocol) connections in an IEEE 802.11 based multi-hop network, they show that the current TCP protocol does not work well above the 802.11 MAC layer.

III. Model and Assumptions

We study wireless sensor networks in which all sensors have the same transmission range (or radius). The transmission range of a node determines the set of nodes it can communicate directly, which are also called its *neighbors*.

Consider a wireless sensor network with n sensor nodes and m sink nodes. Typically n is much larger than m, and in some cases m may be just 1. We assume that n is known to all nodes in the network, and each node has a unique ID. Let S be the set of all sensor nodes in the network and let S_i denote the sensor i, for $i = 1..n$. Similarly let R be the set

of all sink nodes in the network and let R_j denote the sink node j, for $j = 1..m$. Moreover let $V = S \cup R$ be the set of all nodes in the network. For any node $v \in V$, the set of its neighbors is denoted by $\Delta(v)$. A *2-neighbor* of node v is defined as a node which is at most two hops away from v, i.e. u is a 2-neighbor of v if and only if u and v are neighbors of each other or they have at least one neighbor in common. We denote the set of 2-neighbors of v by $\Delta_2(v)$.

As stated earlier, the channel access is TDMA-based: time is divided into *frames*, which are further divided into *time slots*. Let Λ denote the number of time slots in each frame, also called the *frame size*. We assume that each node has the same frame size. Each node $u \in V$ has a time slot t_u and keeps a local table (local frame), F_u, of its neighbors' time slots relative to its own. If $v \in V$ is a neighbor of u, the entry $F_u(v)$ is the time slot in u's frame that coincides with t_v. This is depicted in figure 1.

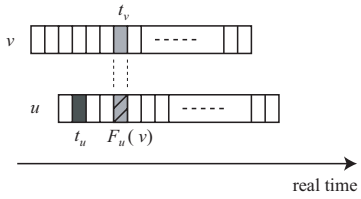


Fig. 1. The time slot of a node relative to another node's time frame. The values t_u and $F_u(v)$ are according to node u's local clock, and t_v is according to v's.

In this paper we consider the problem of finding delay optimal routing paths between sensor and sink node pairs, with the objective of minimizing the total end-to-end delay. The proposed approach could be used for both pull (sensor to sink) and push (sink to sensor) modes of communication [12]. However, for the clarity of presentation, we will only consider constructing routes from sensors to sink nodes, as this is the most common mode of communication in a sensor network. The application of proposed ideas to the other case is quite straightforward.

IV. Routing with Data Fusion

In this section we focus on a sensor network scenario using in-network data fusion, i.e. any incoming data to a node are aggregated in some way and then relayed to the next hop. A typical example of this scenario is a forest fire detection application, in which the sensors report large deviations in temperature readings. A relaying node that receive readings from multiple sensors can then transmit the average of all values received, instead of transmitting the individual readings. In-network data fusion is a method of significant energy-savings and should be employed wherever the application scenario allows. With in-network data fusion, there is no queueing delay since a node may receive multiple packets from different sources in a single time frame, but it suffices to form and send a single packet after processing all received packets so far in the frame. Hence the amount of congestion at relaying nodes, or the number of packets received in a single frame, has no effect on the relaying delays. In this section we formulate and solve this problem.

A. Problem Formulation

We first construct an auxiliary *directed* graph $G(V, E)$, where $V = S \cup R$ is the vertex set corresponding to the set of sensor and sink nodes in the network, and E is the edge set corresponding to the links between pairs of nodes that are within each other's transmission range. We use the same notations for vertices in the graph and nodes in the network to ease the presentation, and it will be clear from the context which one is implied. Since we are interested in finding routes from the sensor nodes to the sink nodes in the network, we adapt the terms *source vertices* and *destination vertices* in the graph corresponding to sensor nodes and sink nodes in the network, respectively. Hence each vertex $S_i \in S$ is called a source vertex, and each $R_j \in R$ is called a destination vertex in graph G. We assume that G is strongly connected. Note that this assumption is equivalent to the basic assumption that the sensor network is connected.

For each link (u, v) in a sensor network, there are two weighted (and directed) edges, (u, v) and (v, u), in the auxiliary graph. The weight of the directed edge (u, v) is denoted by $w(u, v)$ and is assigned as

$$w(u, v) = (F_u(v) - t_u) \bmod \Lambda.$$

Intuitively, weight $w(u, v)$ is the amount of delay observed if node v relays a packet received from node u. By the intrinsic property of contention-free MAC protocols, two neighbor nodes can not have overlapping timeslots, i.e. $t_u \neq F_u(v)$, for any $(u, v) \in E$. Hence we have

$$1 \leq w(u, v) \leq \Lambda - 1$$

Figures 2 and 3 demonstrate a simple example network and the corresponding auxiliary graph. A sensor network with 6 nodes is shown in figure 2. The set of sensor nodes and sink nodes are not identified as they are irrelevant for the construction of the auxiliary graph. The local time frame and allocated time slot of each node is also shown, respecting their alignment in real time. Figure 3 represents the auxiliary graph corresponding to this network. Consider, for example, nodes a and b in figure 2. Suppose that b receives a packet from a which it needs to relay to some other node. After a sends this packet at its timeslot t_a, node b can not relay it immediately after receiving it; it has to wait for its allocated time slot t_b. Note that t_b coincides with the time slot in a's frame which is three slots after t_a. Hence the packet will suffer a delay of 3 time slots, which also includes the transmission time of the packet. Therefore the weight of the directed edge (a, b) is set to 3 in the auxiliary graph. If, on the other hand, a forwards a packet of b, then the delay would be $-3 \bmod \Lambda = 7$, since $\Lambda = 10$ for this example. Hence $w(b, a) = 7$ in figure 3. Also note that $w(u, v) + w(v, u) = \Lambda$ for any two vertices u and v in the auxiliary graph.

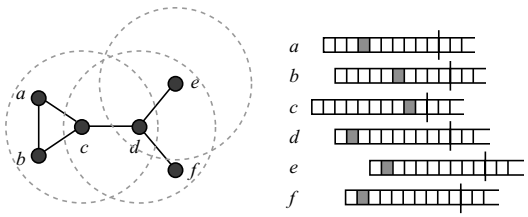


Fig. 2. An example network with 6 nodes. Transmission ranges are shown only for nodes *c*, *d* and *e*, but all links are demonstrated by the lines between neighbor nodes. On the right is the local time frames of each node shown with their relative alignment.

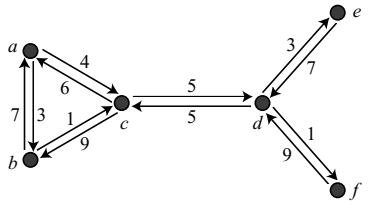


Fig. 3. The auxiliary directed graph G corresponding to the network in Figure 2.

Our objective is to minimize the end-to-end delay from each sensor node to *any* sink node. More precisely, for two nodes u and v, let $\delta(u, v)$ denote the shortest path length between vertices u and v in graph G, where path length is defined as the sum of the weights of all edges on that path. Then for each source vertex S_i, the objective is first to find its *closest* destination vertex R_j, which we define as the vertex satisfying

$$\delta(S_i, R_j) = \min_{k=1..m} \delta(S_i, R_k).$$

Then for each such (S_i, R_j) pair, we want to find the shortest path from S_i to R_j. We now present an integer programming (IP) formulation of the problem based on a variation of the multicommodity flow problem. We define a commodity originating from each sensor node, destined to any one of the sink nodes. Therefore there is a total of n commodities. We then try to minimize the total cost (weight) of the commodity flows. Note that in our case, there is no edge capacity in the auxiliary graph, therefore the flows of the commodities are independent of each other. In other words, one can also define n independent IP formulations to individually find shortest-paths between S_i, R_j pairs. Nevertheless, we formulate a single IP consisting of n commodities, which will also serve as a basis for our QIP model later in section VII-A. Let X_{ij}^k be a decision variable such that

$$X_{ij}^k = \begin{cases} 1 & \text{if } (i,j) \in E \text{ is used for commodity } k \\ 0 & \text{otherwise} \end{cases}$$

Then the optimization problem can be formulated as

$$\text{minimize} \sum_{(i,j)\in E} \sum_{k \in S} X_{ij}^k w(i,j)$$

subject to

$$\sum_{j \in V} X_{ij}^i = 1 \quad , \qquad \forall i \in S \tag{1}$$

$$\sum_{i \in S} \sum_{j \in R} X_{ij}^k = 1 \quad , \qquad \forall k \in S \tag{2}$$

$$\sum_{j \in V} X_{ij}^k = \sum_{j \in S} X_{ji}^k \quad , \qquad \forall k \in S, i \in S, i \neq k \tag{3}$$

$$X_{ij}^k \in \{0, 1\} \quad , \qquad \forall (i,j) \in E, k \in S \tag{4}$$

Here the first constraint states that each commodity i must initiate from source vertex $S_i \in S$. The second constraint forces each commodity to reach a destination vertex $R_j \in R$. The third constraint is the flow conservation equation and must be satisfied for all vertices except the source and destination vertices of the flow. Since no flow can go through a destination vertex as an intermediate vertex, we restrict the incoming flow of a vertex, i.e. the right hand side of the constraint, to the flow coming from the vertices in S. Also note that whenever we use the term X_{ij}^k, there is an implicit constraint on i and j such that $(i, j) \in E$.

B. GreenWave Routing

As stated earlier, underlying this IP formulation is n independent network flow problems (one for each source node). By the *Unimodularity Theorem* [10], the LP relaxation of each of those IPs yields an integral optimal solution, which is also the optimal solution of that IP. Therefore each individual flow problem can be efficiently solved. However, global information of the sensor network is required in this approach, and this is impractical for sensor networks.

On the other hand, given the auxiliary graph G, we observe that the problem is actually a generalization of the single-destination shortest paths problem such that there are *multiple copies of the destination*. Here we use the phrase "single-destination shortest paths", noting that single-source and single-destination shortest paths problems are equivalent since one can be converted to the other by reversing all edge directions in the graph. We claim that we can use a generalized version of the Bellman-Ford algorithm [5] to construct shortest-path trees, each rooted at a destination vertex. In this approach each sink node (destination vertex) can be treated as a copy of the single destination. Then the only modification needed is to allow multiple destinations in Bellman-Ford algorithm by initializing the d values of all destination vertices to 0, where d is the local shortest-path estimate.

Although the only difference is in the initialization section of Bellman-Ford algorithm, we present the complete centralized algorithm here for completeness (see Algorithms 1, 2, 3). In the next section we present the correctness proof for the *generalized Bellman-Ford algorithm*.

Each vertex u has a shortest-path estimate $d[u]$, which eventually converges to the actual length of the shortest path from u to its closest destination vertex. Also associated with each vertex u is the variable $\pi[u]$, which keeps the parent of

Algorithm 1 INITIALIZE (S, R)

1: **for** $i = 1..n$ **do**
2: $d[S_i] \leftarrow \infty$;
3: $\pi[S_i] \leftarrow$ NIL;
4: **for** $j = 1..m$ **do**
5: $d[R_j] = 0$;
6: $\pi[R_i] \leftarrow$ NIL;

Algorithm 2 RELAX $(u, v, w(u, v))$

1: **if** $d[u] > d[v] + w(u, v)$ **then**
2: $d[u] \leftarrow d[v] + w(u, v)$;
3: $\pi[u] \leftarrow v$;

u in the shortest path from u to its closest destination vertex. The BELLMAN-FORD algorithm starts with initializing local variables d and π for all vertices, and executes its main loop n times.

1) Correctness: Although the modification in the algorithm is brief, its effect is significant and needs a formal analysis. In this section, we basically prove the following theorem, which verifies the correctness of the generalized Bellman-Ford algorithm.

Theorem 1: At the termination of the generalized Bellman-Ford algorithm,

(i) m shortest-path trees emerge, each rooted at a distinct destination vertex $R_j \in R$, for $j = 1..m$,
(ii) all shortest-path trees are node-disjoint, hence each $S_i \in S$ is part of a single tree, for $i = 1..n$,
(iii) if $S_i \in S$ is part of the tree rooted at $R_j \in R$, then

$$d[S_i] = \delta(S_i, R_j) = \min_{k=1..m} \delta(S_i, R_k).$$

We assume that the vertices and edges of the graph are static during the execution time of the algorithm. Then we can show that the d values of each node converges to the shortest-path lengths after the main loop of Algorithm 3 is executed n times. We will utilize an adaptation of the path relaxation property of [5], which is presented below.

Lemma 1: **Path-relaxation Property** *[5]***:** If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from v_0 to v_k, and the edges of p are relaxed in the order $(v_{k-1}, v_k), (v_{k-2}, v_{k-1}), \ldots, (v_0, v_1)$, then $d[v_0] = \delta(v_0, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with the relaxations of the edges of p.

We first show that the number of edges in a shortest path in G from any source vertex S_i to its closest destination R_j can be at most n. It is easy to see that a shortest path from S_i to R_j can not go through another destination vertex $R_k \in R$, simply because R_k would then be the closest destination vertex for S_i. On the other hand, a shortest path can not have cycles as all edge weights in G are positive and removing the cycle from the path would yield a shorter path. Thus, each of n vertices in S can be used at most once in a shortest path.

Algorithm 3 BELLMAN-FORD $(G = (S \cup R, E), w)$

1: INITIALIZE$(S[G], R[G])$
2: **for** $i = 1..n$ **do**
3: **for** each edge $(u, v) \in E[G]$ **do**
4: RELAX$(u, v, w(u, v))$;

Now consider an (S_i, R_j) pair such that R_j is the closest destination for S_i, and let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be the shortest path from $v_0 = S_i$ to $v_k = R_j$. Then at the first iteration of the main loop of Algorithm 3, the edge (v_{k-1}, v_k) will be relaxed among all other edges. At the second iteration of the algorithm, the edge (v_{k-2}, v_{k-1}) will be relaxed, and so on until the edge (v_0, v_1) is relaxed at the kth iteration. Hence by lemma 1, all edges of p will be relaxed in order after k iterations of the algorithm and $d[v_0]$ will converge to $\delta(S_i, R_j)$. Since we have $k \leq n$ for all $R_i \in R$, all d values will converge to the desired shortest-path lengths at the termination of the algorithm.

We now prove that the algorithm splits the graph into m partitions, each partition being a tree rooted at a destination vertex. First we construct a spanning subgraph $G'(V, E')$ of G, where $E' = \{(u, \pi[u]) \mid u \in S\}$. Clearly G' has no cycles: each edge on a cycle must satisfy the relaxation condition of algorithm 2, and this yields a contradiction since all edge weights are positive. Hence the graph G' must be a forest, i.e. a collection of trees.

Lemma 2: Each destination vertex R_j has $\pi[R_j]$ =NIL after the algorithm termination, and hence is the root of a tree in G'.

Proof: For any vertex u in the graph, $\pi[u]$ is updated only if line 1 in the RELAX operation, $d[u] > d[v] + w(u, v)$, evaluates to true. Since all edge weights in the graph are positive, this cannot evaluate to true for any outgoing edge of the destination vertex R_j, as $d[R_j]$ is initialized to 0 -the minimum possible value- prior to the algorithm. ■

Since graph G is strongly connected, there is a path in G from any $S_i \in S$ to any $R_j \in R$. Then by the path-relaxation property, each source vertex S_i will have a destination vertex R_j as one of its ancestors in G'. Combined with lemma 2, this states that each tree in G' is rooted at a destination vertex.

Lemma 3: For each destination vertex R_j, let T_j denote the tree rooted at R_j. Then all trees $T_1, T_2, \ldots, T_m$ are node-disjoint.

Proof: Suppose that the trees T_i and T_j share a common node u. Then there are two cases for node u: (i) it has two parents -one for each tree- or (ii) it has a single parent -shared by both trees. The first case contradicts with the fact that the algorithm keeps only a single parent, π, for each node, hence can not happen. Consider the second case, and let $\pi[u] = v$. Then there are two same cases for v: it has either two parents or just one. The first case is again a contradiction and can not happen. Continuing recursively in this manner, we must reach an ancestor of u, call it z, which is the root of one of the trees, say T_i. Since each tree has a unique root, we have that z, a destination vertex, is an intermediate node in T_j, which contradicts with lemma 2. Therefore T_i and T_j are

node-disjoint. ■

Theorem 1 follows from Lemmas 1, 2 and 3.

2) Distributed Implementation: By implementing GreenWave routing as an extension of Bellman-Ford algorithm, we can take advantage of its well studied distributed implementation [6]. The idea is to have all nodes execute the algorithm on its incident edges in parallel and communicate their computations to their neighbors at some intervals. We now present precisely how nodes in a sensor network would apply the algorithm locally and in a distributed manner. Consider node u, and assume that it has a virtual clock $C_u(t)$, which is simply a counter incremented at each time slot. Then at real time t, node u executes the following:

Algorithm 4 GREENWAVE(node u, time t)

```
1: if C_u(t) mod Λ = t_u then
2:    broadcast d[u];
3: else
4:    if received d[v] from neighbor v, such that
      d[v] + w(u,v) < d[u] then
5:       d[u] ← d[v] + w(u,v);
6:       π[u] ← v;
```

Each node can execute this algorithm locally in every time slot, after initializing its local variables as in the GBF algorithm. Having a time-slotted communication paradigm actually provides a perfect environment for the synchronous execution of the algorithm: each node relaxes its outgoing edges and communicates its updated d value exactly once in every time frame. Hence during each time period of Λ time slots, all links in the network would be relaxed, corresponding to one iteration of the main loop of the centralized algorithm. Then it would suffice for each node to execute Algorithm 4 for $\tau = n\Lambda$ time slots, given that they start executing the algorithm at the same time. We assume that the nodes in the sensor network are already synchronized by the underlying MAC protocol, either by a physical synchronization method such as using GPS [11], or by a virtual synchronization method embedded in the contention-free MAC protocol, such as in [2]. Even if there is a drift between the times the nodes start the algorithm, each node may execute the algorithm for an increased amount of time to offset the different starting times at individual nodes. The value of τ could easily be adjusted by achieving an upper bound on the maximum difference between the times each node starts the algorithm. We can safely assume that $\tau = O(n)$, and the message complexity of the algorithm is then $O(n)$ for each node, since each node sends only one message for each of the τ time frames.

After node u is done with the GREENWAVE algorithm by executing it for τ time slots, it forwards all incoming packets to $\pi[u]$. It has no information as to which sink node the packets are forwarded to, and it does not need to. With the local information stored at each node, the packets eventually reach the closest sink node. Here we restate our assumption on the scenario that each sensor can report its data to any sink node in the network.

V. IEEE 802.11 Delay Analysis

To the best of our knowledge, there is no existing study comparing the performance of a contention-free and a contention-based MAC protocol. Recall that the contention-free MAC protocols have the desirable properties for an energy-constrained sensor network, since there is no energy waste due to collisions and retransmissions. Their drawback, on the other hand, is increased latency and lower throughput. We proposed GreenWave routing to alleviate this drawback, and it is a natural step to compare the performance of the resulting contention-free protocol suit with that of a contention-based protocol, such as 802.11, and see how much we gain by routing data along *green-wave paths*.

To assess the delay analysis of the IEEE 802.11 MAC protocol, we combine and extend the two analytic models by Bianchi [7] and Wang et al. [20]. Bianchi model presents accurate results for the 802.11 protocol DCF (distributed coordination function) scheme in saturation conditions, verified by simulation results. As in most of the existing analytic models of 802.11, the Bianchi model considers the single hop communication, where all considered stations are assumed to contend against each other for channel access, i.e. there are no hidden terminals. The Wang model, on the other hand, considers a generic Carrier Sense Multiple Access (CSMA) protocol with collision avoidance, and also takes the hidden terminals into account. Both of these work provide throughput analysis. In this section, we first extract the delay information for both models, and then combine the results of [7] and [20] to obtain the end-to-end delay performance of the 802.11 protocol. The results also present the significant effect of hidden terminals on the protocol performance.

A. Bianchi Model

When a node u wants to access the channel, it contends with its immediate neighbors. Moreover, if v is a 2-hop neighbor of u (i.e. v is a neighbor of one of u's neighbors), then u may also contend with v if the intended receiver of u or v receives signals from both of them. This is called the hidden terminal problem, and is handled in 802.11 protocol by the use of RTS/CTS (request to send/confirm to send) message exchange.

The Bianchi model assumes that there are N contending stations for the channel, where there are no hidden terminals. In a multi-hop environment, if we consider the communication link from u to v, there are $|\Delta(u)|$ stations *directly* contending with the sender u, where $\Delta(u)$ is the set of neighbors of u. Thus we can simply set the number of contending stations as $N = |\Delta(u)|$ in Bianchi model to obtain a lower bound for the one-hop delay in 802.11. Note that in reality the delay would be higher due to hidden terminals. We will investigate this issue using the Wang model later in this section.

In the Bianchi model, the average amount of time spent on the channel in order to observe a successful packet transmission is given as

$$T_{cs} + \tau\frac{1-P_{tr}}{P_s P_{tr}} + T_{cc}\left(\frac{1}{P_s} - 1\right) \tag{5}$$

where T_{cs} is the average time the channel is sensed busy because of a successful transmission, T_{cc} is the average time the channel is sensed busy by each station during a collision, τ is the duration of an empty slot time, P_{tr} is the probability that there is at least one transmission in the considered time slot, and P_s is the probability that a transmission on the channel is successful, i.e. the probability that exactly one station transmits, conditioned on the fact that at least one station transmits. More intuitively, the first term in (5) is the time spent on the channel in order to successfully transmit a packet. The second term is the amount of time the channel is idle, per successful transmission. The third term represents the time wasted on the channel because of collisions, per successful transmission.

Given the number of contenders N, let $D(N)$ denote the average delay to observe a successful transmission. We can then expand (5) to obtain

$$D(N) = T_{cs} + \tau\frac{1-p}{Np} + T_{cc}\left(\frac{1-(1-p)^N}{Np(1-p)^{N-1}} - 1\right) \quad (6)$$

where p is the probability that a station transmits in a randomly chosen slot, and can be approximated as [7]

$$p \approx \frac{1}{N\sqrt{T_{cc}/2\tau}}.$$

Note that p has nothing to do with the traffic generation at a node. The Bianchi model assumes saturation conditions, i.e. each node has always data to transmit, but has to remain silent during back-off slots. T_{cs} and T_{cc} are constant values dependent only on protocol specific settings, such as the length of DIFS (distributed inter-frame space) and SIFS (short inter-frame space) durations, and τ is the constant slot duration. Therefore we can express the single-hop transmission delay as a function of N.

An important observation is that $D(N)$ is the average delay until *any* successful transmission is observed among N contenders. Hence if we consider a specific node u among these contenders, it has an expected delay of $N/2 \cdot D(N)$ before it can transmit successfully. Since we set $N = |\Delta(u)|$, we obtain the following estimate D_P for the end-to-end delay of a path P, ignoring hidden terminals.

$$D_P = \sum_{(u,v)\in P}\left[\frac{|\Delta(u)|}{2} \cdot D(|\Delta(u)|)\right]$$

B. Wang Model

The delay bound provided by the Bianchi model may be loose since the hidden terminals are not considered. In order to investigate the effect of hidden terminals on the performance of 802.11 protocol, we use a recent work by Wang et al., [20], which provides an analysis of generic collision avoidance protocols in multihop wireless networks. Unlike most of the previous work, the authors take into account the hidden terminals in their throughput analysis. They use two separate Markov chains, one for modelling the state of the channel around a node, and the other for the state of the node itself.

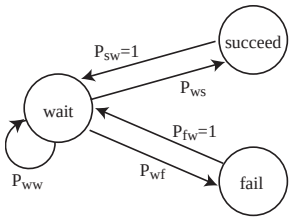


Fig. 4. Markov chain model for a node [20]

The channel is viewed as a circular region around the node with a set of other contending nodes. The *channel Markov chain* is used to compute the probability of the channel around the node being idle, and then the *node Markov chain* is used to compute the steady state probabilities of the node being in each of the states, *wait*, *fail*, and *succeed*. The two Markov chains together generate a set of closed form expressions that can be solved using numerical methods.

Figure 4 shows the Markov chain for a node, which is used in [20] to obtain the throughput of a single node. Using this Markov chain, we can extract the delay that a packet suffers when being sent by this node.

The steady state probabilities of *wait*, *succeed* and *fail* states are given as π_w, π_s and π_f and can be computed using the methodology of [20]. Also, let T_w, T_s and T_f be the durations of *wait*, *succeed* and *fail* states, respectively. The duration of a state is the amount of time spent in that state before a transition occurs (possibly back to itself).

Let T_{ws} denote the time spent until the Markov chain first visits the *succeed* state starting from the *wait* state. Then T_{ws} is the expected amount of delay observed at the node before a successful transmission starts, which is our desired value. Similarly let T_{fs} be the time spent before the Markov chain first visits the *succeed* state starting from the *fail* state. We have

$$T_{ws} = P_{ws}T_w + P_{ww}(T_w + T_{ws}) + P_{wf}(T_w + T_{fs}) \quad (7)$$
$$T_{fs} = P_{fw}(T_f + T_{ws}) = T_f + T_{ws} \quad (8)$$

Plugging (8) into (7) and solving for T_{ws}, we obtain

$$\begin{aligned} T_{ws} &= P_{ws}T_w + P_{ww}(T_w + T_{ws}) \\ &\quad + P_{wf}(T_w + T_f + T_{ws}) \\ T_{ws}(1 - P_{ww} - P_{wf}) &= P_{ws}T_w + P_{ww}T_w \\ &\quad + P_{wf}T_w + P_{wf}T_f \\ T_{ws}P_{ws} &= T_w(P_{ws} + P_{ww} + P_{wf}) + P_{wf}T_f \\ T_{ws} &= \frac{T_w + P_{wf}T_f}{P_{ws}} \quad (9) \end{aligned}$$

Noting that $\pi_f = \pi_w P_{wf}$ and $\pi_s = \pi_w P_{ws}$ from the Markov chain, we can rewrite (9) as

$$\begin{aligned} T_{ws} &= \frac{T_w + \frac{\pi_f}{\pi_w}T_f}{\frac{\pi_s}{\pi_w}} \\ &= \frac{\pi_w T_w + \pi_f T_f}{\pi_s} \end{aligned}$$

The reader is referred to [20] for the detailed expressions of π_w, π_s, π_f and T_w, T_s, T_f. However we stress here that there is a strong dependency of these quantities on the probability that a node transmits in a slot, p', and the number of contenders in the channel, N. p' is a protocol-specific parameter and its value depends on how the MAC protocol grants nodes to access the channel. Note that [20] considers the carrier-sense multiple access protocols in general, not specifically the 802.11 protocol.

Recall that using the Bianchi model, we could compute p, the probability that a node transmits in a randomly chosen slot in the 802.11 protocol. Hence we can simply incorporate this result into the Wang model to obtain the results for 802.11, by setting $p' = p$. The definition of p in [7] exactly matches that of p' in [20], and its value is derived by the behavior of 802.11's exponential back-off algorithm.

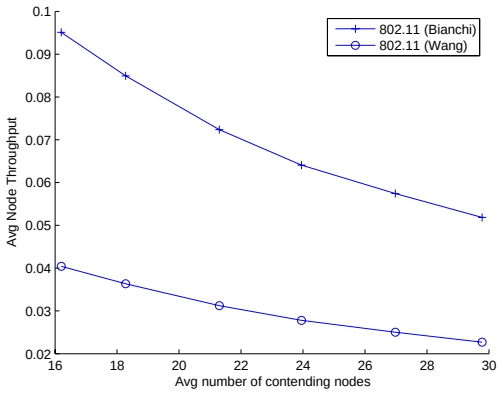
Since [20] models the channel as a circular region around the sender node, we can easily extend the analysis to find the expected end-to-end delay between two nodes. We consider each hop as an instance of this model in which the intermediate sender node and its neighbors compose the nodes in the channel.

C. Numerical Results

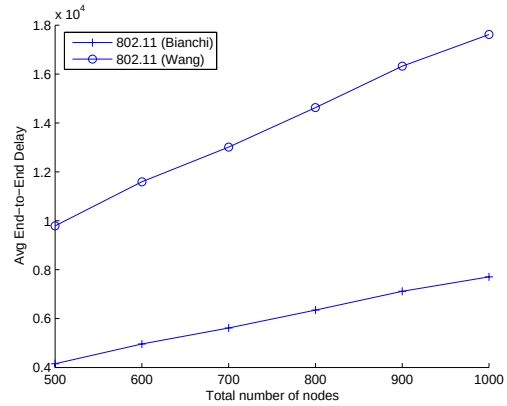
We now provide some numerical results to demonstrate the effect of hidden terminals on the 802.11 protocol performance. We randomly generate networks of different sizes, $n = 500$ to $n = 1000$ nodes. Nodes are distributed uniformly at random in a unit area and the transmission radius is constant, $r = 0.1$, for all networks. Hence the node density (a.k.a. average number of neighbors of a node) increases as the network size becomes larger. We present two figures showing the average node throughput and the average end-to-end delay. Each data point in the figures represent an averaged value over 100 random networks of the given size. In order to investigate the end-to-end delay values, shortest-path routing is applied to route packets from every sensor node to the closest sink node. The number of sink nodes is set as constant, $m = 3$. Other protocol specific parameters used conform to the settings given in [20]; data packet size is 100τ, and ACK, RTS, CTS packet sizes are 5τ.

Figure 5(a) shows the average node throughput versus the average number of contending nodes. The number of contending nodes is essentially the average node degree in the graph corresponding to the network of given size, varied from 500 to 1000 nodes. We observe that the throughput is reduced by more than half with the hidden terminals taken into account.

Similarly in Figure 5(b), there is more than two folds increase in the end-to-end delay when hidden terminals are considered. The effect of hidden terminals on the end-to-end delay also increases as the network gets denser. The values on the y-axis are expressed in τ units, i.e. the number of slot times elapsed before the packet arrives from source to destination.



(a) Node throughput vs. Number of contending nodes.



(b) End-to-end delay vs. Network size.

Fig. 5. The effect of hidden terminals on the 802.11 protocol. *Bianchi* represents the analysis ignoring the hidden terminals, and *Wang* represents the analysis with hidden terminals.

VI. GreenWave Routing Performance Analysis

In this section we present the performance effect of routing over the *green-wave paths* from sensors to sinks. We also compare contention-free protocols to the contention-based 802.11 protocol using the results from the previous section. We observe that by the existence of GreenWave routing, contention-free MAC protocols may surpass the 802.11 MAC protocol with respect to end-to-end delay and throughput.

Let us first introduce some notations and assumptions that we will make use of in this section. We denote by SH the shortest-hop routing algorithm, which simply routes packets over minimum-hop paths. Note that this can be achieved by simply setting each edge weight to 1 in the generalized Bellman-Ford algorithm. We let CF represent some given contention-free MAC protocol. We are not interested in how this underlying MAC protocol works, i.e. how it allocates a time slot for each node. Instead, given a frame size, we randomly generate time slots for all nodes in the network respecting the no-conflict requirement. The frame size and the allocation of time slots to the nodes collectively represent a given CF MAC protocol.

Note that contention-free time slot allocation in a sensor network is equivalent to the distance-2 vertex coloring in a graph. Because of the hidden terminal problem, nodes in a network that are within two hops distance can not be assigned the same time slot, just as 2-distant vertices in a graph can not be assigned the same color. Hence the smallest possible frame size can not be determined efficiently, since the coloring problem is NP-hard. However, a simple upper bound on the minimum frame size is the maximum 2-neighborhood size, Δ_2, in the network, meaning that any frame size larger than Δ_2 is feasible. In obtaining the results of this section, we calculate Δ_2 in the given graph, and set this value as the frame size in the CF MAC protocol. Hence our basic aim is to investigate the CF MAC protocols at their limits.

We compare three different approaches in this section, each of which represents a combination of a MAC and a routing protocol.

- *CF + GW* — Our GreenWave routing protocol on top of a contention-free MAC protocol.
- *CF + SH* — The shortest-hop routing protocol on top of a contention-free MAC protocol.
- *IEEE 802.11 + SH* — The shortest-hop routing protocol on top of the contention-based IEEE 802.11 MAC protocol.

Performance metric — Given a sensor network, for each protocol suit we compute the end-to-end delay of the path from each sensor to its closest sink. We then average the delay over all such paths to get the performance of the protocol on the given network. Throughout this section, each data point (the delay or throughput value) is an averaged value over 100 random networks. We assume saturation conditions, i.e. each node has always data to send.

We present performance results calculated on random sensor networks. The random networks are generated by a random geometric graph model: nodes are scattered in a unit area, uniformly at random, and a pair of nodes that are at most r units apart from each other are connected (set as neighbors in the network), where $0 < r < 1$ is the transmission radius. We also ensure that each randomly generated network is connected. For all networks studied in this section, we set the number of sink nodes as three. Hence n and r uniquely define a class of network topologies on which we present the results.

An extra parameter needed for the contention-free MAC protocols is the *slot time*, or the duration of each time slot. Note that the *slot time* in 802.11 is very small as its only role is to transform the continuous time line into discrete time steps. For a CF MAC protocol, each time slot should be long enough for a packet to be transmitted over the channel. Since we use the packet size 100τ in our numerical results, the slot size for the CF MAC protocol is set as 100τ.

Figure 6 demonstrates the average end-to-end delay figures for all three approaches. In order to see the effect of node density, we keep the transmission radius fixed, $r = 0.1$, and increase the network size from 500 nodes to 1000 nodes. We first observe that GreenWave routing significantly decreases

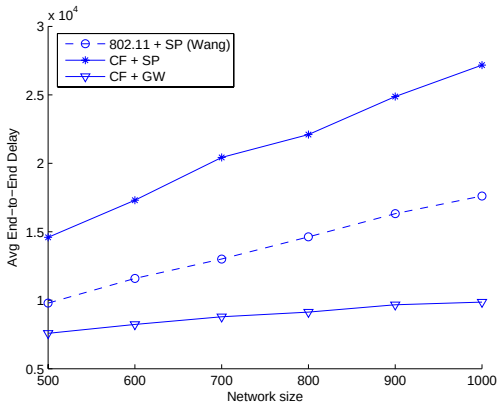


Fig. 6. Average end-to-end delay versus network size. Transmission radius is fixed, $r = 0.1$, for all network sizes.

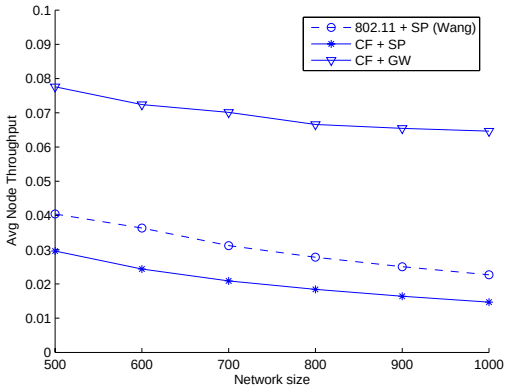


Fig. 7. Average node throughput versus network size. Transmission radius is fixed, $r = 0.1$, for all network sizes.

the average end-to-end delay for a contention-free MAC protocol, more than half, when compared to using shortest-hop routing on top of the CF MAC. More interestingly, the performance of CF+GW is better than 802.11+SH although the CF+SP scheme is always surpassed by 802.11+SH. The end-to-end-delay savings by the GreenWave routing increases with higher density of nodes in the network.

We also investigate the average throughput of nodes in the network for each approach under discussion. For this purpose, instead of the end-to-end delays, we compute the average single-hop delay for the same set of random networks. Let D denote the average single-hop delay, and χ the packet size in the network. Then, under saturation conditions, the average throughput of a node can be stated as χ/D. Figure 7 demonstrates the results for the same set of networks used for Figure 6.

As expected, a contention-free protocol by itself always performs poorly against the contention-based protocol with respect to communication delays or throughput regardless of the network size and density. However, by the help of GreenWave routing, contention-free protocols may become

preferable especially as the network size and density grows. Given that many sensor network scenarios project thousands of densely deployed nodes, CF+GW protocol suits seem more suitable for sensor networks with in-network data fusion.

VII. Routing Without Data Fusion

In some scenarios, in-network data fusion may not be available such that the relaying node forwards each single packet it receives as is. Then the delay observed on a packet is greatly affected by how many other packets the relaying node receives in that time frame. In other words, the flow of different commodities may have strong effects on each other's end-to-end delays. Let us first demonstrate this effect by a simple example.

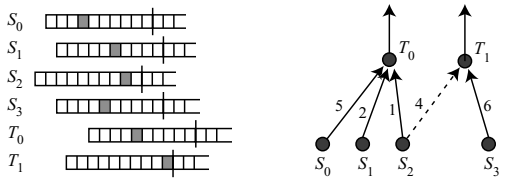


Fig. 8. A small sample network. Local time slots of nodes are shown on the left. The delay on each link (u, v) is calculated by $w(u, v) = (F_u(v) - t_u) \bmod \Lambda$, and the resulting routing tree is shown on the right. The dashed arrow only shows that S_2 and T_1 are neighbors.

Consider the simple example in figure 8, where there are four sensor nodes and two relaying nodes. To illustrate the idea, assume for now that the four sensors are the only data producers and the relaying nodes just forward their data. The local allocated time slots for each node are also shown on the left. If the link weights are assigned simply by the formula $w(u, v) = (F_u(v) - t_u) \bmod \Lambda$, then three nodes report their data through T_0 and only one node to T_1 as a result of the algorithm. However, if in-network data fusion is not applicable and if all three nodes S_0, S_1, S_2 have packets to transmit in the same time frame, then T_0 will have three packets to relay, where it can only send one at a time. Therefore the effective delay on these packets will be higher than the ones shown on the right in figure 8. If we assume that each node transmits a packet in every time frame, and that T_0 transmits the packets one by one in the order they were received, then in the steady state the delays for S_0, S_1, S_2 will all go up by 2Λ as shown on the left in figure 9. On the other hand, S_2 is also a neighbor of T_1 and thus T_1 could relay its packets as well. The graph on the right hand side of figure 9 shows the effective delays if S_2 joins the tree of T_1 instead of T_0. Note that the total delay -total link cost- on the right is less than the one on the left, hence is preferable. In this section, we try to model this modified problem by taking into account the interaction among different flows in the network.

A. Problem Formulation

We now formalize the interaction among different flows and formally define the modified problem. Suppose that we

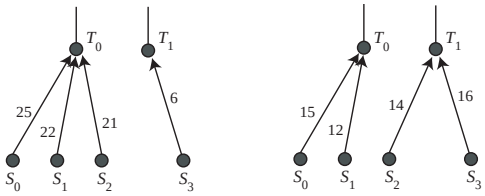


Fig. 9. The two possible solutions to the example of figure 8, this time taking into account the interaction among different flows for calculating delays. The routing tree on the right yields a lower total delay and hence is preferable.

are given a *routing graph* which is formed by the union of n paths, from each source vertex to some destination vertex. Each path in the routing graph defines the flow of the commodity originating from a unique source vertex. We define the *congestion* on vertex v as the number of different commodities (paths) going through v, and denote it by $\mathcal{C}_v$. Hence the corresponding node v relays the packets of $\mathcal{C}_v$ different nodes, which we call the *descendants* of v. Note that v also "relays" its own traffic, hence a total of $\mathcal{C}_v + 1$ distinct commodities are leaving out of v, i.e. v has $\mathcal{C}_v + 1$ descendants. Assume that each node senses an event (hence transmits) in each time frame with probability p. The fixed probability p may be considered as an averaged quantity that reflects a possibly bursty traffic generation in the long term. This approach has been extensively applied before to make the theoretical modeling tractable [20] [21] [22] [23]. p also serves as a tuning parameter in our model to represent sensor network scenarios with different event frequencies –hence different traffic loads.

Consider an edge (u, v) on the routing graph. Assume that some commodity k is active on (u, v) at some instant. We want to find the *effective delay* on commodity k incurred by the link (u, v) by taking into account (i) the time slots of u and v, and (ii) other commodities that are relayed by v. With probability $(1-p)^{\mathcal{C}_v}$, none of the other descendants of v transmits in this frame, and the delay incurred by the link (u, v) is then just $w(u, v)$. Similarly, with probability $\mathcal{C}_v p(1-p)^{\mathcal{C}_v - 1}$, only one of the other descendants of v transmits in this frame, and in this case the delay for u goes up to $w(u, v) + \Lambda$. Note that the packet of commodity k may be the first one to be relayed by v, but in the steady state, if two nodes are sending packets and v is relaying them alternatingly, then both of them suffer Λ amount of extra delay. Considering all possible cases, we can calculate the expected value of the delay when v relays a packet from u, for some given commodity. Let $w'(u, v)$ be the effective delay on edge (u, v). Then –using $\mathrm{E}[X] = \sum x P\{X = x\}$– we can compute the expected value of $w'(u, v)$ as follows

$$\mathrm{E}[w'(u,v)] \quad = \quad \sum_{i=0}^{\mathcal{C}_v} \left[(w(u,v) + i\Lambda) \binom{\mathcal{C}_v}{i} p^i (1-p)^{\mathcal{C}_v - i} \right]$$

$$
\begin{aligned}
&= w(u,v) \cdot \sum_{i=0}^{\mathcal{C}_v} \left[\binom{\mathcal{C}_v}{i} p^i (1-p)^{\mathcal{C}_v - i} \right] \\
&\quad + \Lambda \cdot \sum_{i=0}^{\mathcal{C}_v} \left[i \binom{\mathcal{C}_v}{i} p^i (1-p)^{\mathcal{C}_v - i} \right] \\
&= w(u,v) + p\mathcal{C}_v\Lambda \qquad (10)
\end{aligned}
$$

The last step follows from the fact that the first summation is the sum of all probabilities for a random variable $\mathbf{X}$ that follows a binomial distribution, $b(i; \mathcal{C}_v, p)$, and the second summation is the expected value of $\mathbf{X}$, which equals $p\mathcal{C}_v$.

For convenience of presentation, we call the first term in (10), $w(u,v)$, the *static cost* and the second term, $p\mathcal{C}_v\Lambda$, the *congestion cost*. Instead of assigning this cumulative cost as a whole to the edge, we attribute the static cost to the edge, (u,v), and the congestion cost to vertex v. Hence the cost of vertex v is $p\mathcal{C}_v\Lambda$. This approach will allow us to model the problem more neatly and help us to derive lower bounds.

We use the multi-commodity flow approach as in section IV-A to mathematically express the congestion on a vertex and the total cost of a path, and then formulate the modified problem as a minimization problem using the same notation. Recall that X_{uv}^k is the variable that equals 1 when commodity k goes through edge (u,v). Then we can express $\mathcal{C}_v$ as

$$\mathcal{C}_v = \sum_{k \in S} \sum_{u \in S} X_{uv}^k.$$

The cost of a path is the sum of the costs of all edges and vertices along the path. We can then express the cost of a path P_k, corresponding to commodity k as

$$Z_k = \sum_{(u,v) \in E} \left[X_{uv}^k \left(w(u,v) + p\Lambda\mathcal{C}_v \right) \right]. \qquad (11)$$

The objective is to minimize the total cost of all flow paths, each corresponding to a commodity originating from a source vertex, i.e. to minimize $\sum_{k \in S} Z_k$. In its open form, we have the optimization problem

$$\text{minimize} \sum_{k \in S} \sum_{(u,v) \in E} \left[X_{uv}^k \left(w(u,v) + p\Lambda \sum_{l \in S} \sum_{u \in S} X_{uv}^l \right) \right]$$

subject to constraints (1), (2), (3) and (4). This is a *quadratic integer programming* (QIP) problem, with a quadratic objective function and linear constraints. We can solve this optimization problem using the QIP solver of the commercially available CPLEX optimizer, but only trivially small networks can be efficiently solved with this approach.

B. Lower Bound

We now try to derive a lower bound for this minimization problem. Our approach is based on considering the static costs and the congestion costs independently. We first derive a lower bound, $L_{\mathcal{C}}$, on the summation of the congestion costs over all paths, then a lower bound L_W on the sum of the static costs over all paths. Note that minimizing the static costs can be efficiently solved as we showed earlier. Hence we just need to find $L_{\mathcal{C}}$. Then the total cost for all paths, the objective function value, can not be lower than $L_W + L_{\mathcal{C}}$.

The congestion cost of path P_k can be extracted from (11) –by setting all edge weights to 0– as

$$p\Lambda \sum_{(u,v) \in E} (X_{uv}^k \mathcal{C}_v).$$

We can safely ignore the constant multiplier $p\Lambda$ in deriving a lower bound on the overall *congestion cost*, $L_{\mathcal{C}}$, in order to make the presentation clearer. Then the overall lower bound can be expressed as $L_W + p\Lambda L_{\mathcal{C}}$. In the rest of this section the congestion cost of a path should be considered this way –omitting the constant term $p\Lambda$.

First we observe that each commodity has to terminate at one of m destination vertices. Hence there is always a total of n flows coming into the set of destination vertices. Let n_i denote the number of sensors reporting to sink i, hence the *congestion* on sink i is n_i. Then the congestion cost of each path coming into i is at least n_i, and the total congestion cost of these n_i paths is at least n_i^2. Summing over all sink nodes, we get $\sum_{i=1}^{m} n_i^2$. Clearly, this summation is minimized when $n_i = n/m$. Hence a simple lower bound on the overall congestion cost is $L_{\mathcal{C}} = n^2/m$. This bound holds regardless of the network topology, and is tight when the network graph is complete. However we can efficiently improve it considering the given topology of a network.

Given the graph G corresponding to a network topology, we first find the shortest path lengths from all sources to their closest destinations, using the hop count as the distance metric. Let $S^{(i)}$ denote the set of vertices with shortest hop distance i to any one of the destination vertices, and let $n^{(i)} = |S^{(i)}|$. Hence $S^{(i)} = \{u \in S \mid \delta_u = i\}$, where δ_u is the shortest hop distance from u to the closest destination vertex. Also let δ be the largest of these shortest hop distances, i.e. $\delta = \max_{u \in S} \delta_u$. Initially, we apply the same strategy on the destination vertices as we did for the general lower bound. Then we remove all destination vertices and their incident edges from G. We then set all vertices in $S^{(1)}$ as the new destination vertices. Note that all paths corresponding to the $n - n^{(1)}$ remaining nodes (those that are at least two hops away from any sink node) have to terminate at one of these new destination vertices. Following our original idea, the additional congestion cost introduced at this level is at least $(n - n^{(1)})^2/n^{(1)}$. We then remove all vertices in $S^{(1)}$ from the graph and apply the same idea for nodes in $S^{(2)}$ and so on until all vertices in the graph are removed. Adding the cost values at each level, we obtain the overall congestion cost

$$L_{\mathcal{C}} = \sum_{i=0}^{\delta} \frac{\left(\sum_{j=i+1}^{\delta} n^{(i)} \right)^2}{n^{(i)}}. \qquad (12)$$

Next we consider only the static costs of paths by setting the congestion on each node to 0. This can be efficiently solved by the generalized Bellman-Ford algorithm to obtain L_W. Then the overall lower bound is simply $L_W + p\Lambda L_{\mathcal{C}}$.

The lower bound algorithm is neatly presented as a pseudo-code in Algorithm 5.

Algorithm 5 LBOUND $(G = (S \cup R, E), w, p, \Lambda)$

$\forall u \in S$, find δ_u = shortest hop distance from u to the closest destination vertex.
$\delta = \max_{u \in S} \delta_u$
$n^{(0)} = m$;
for $i = 1..\delta$ **do**
  $S^{(i)} = \{u \in S \mid \delta_u = i\}$
  $n^{(i)} = |S^{(i)}|$
$L_{\mathcal{C}} = 0$
$n_{rest} = n$
for $i = 1..\delta$ **do**
  $L_{\mathcal{C}} = L_{\mathcal{C}} + \lfloor n_{rest}^2 / n^{(i-1)} \rfloor$
  $n_{rest} = n_{rest} - n^{(i)}$
find L_W by applying generalized Bellman-Ford on G with static costs as edge weights.
return $L_W + p\Lambda L_{\mathcal{C}}$

C. Heuristics

In this section, we present a novel heuristic to find good suboptimal solutions to the QIP problem presented in section VII-A. Given an instance of the problem, let OPT denote the optimal solution to this QIP. Note that any valid routing scheme (that connects each sensor node to some sink node) is a heuristic, and the result of the heuristic provides an upper bound on OPT. Hence, as a first step, our original GW routing algorithm can be considered as a simple heuristic for this problem. However GW does not take congestion into account and hence may perform very poorly –especially when the traffic is heavy (p is large) and thus congestion has larger effects on the delay.

Recall that the end-to-end delay of a communication path has two main factors (i) static costs and (ii) congestion cost. Minimizing the congestion cost requires (a) distributing/balancing the load among nearby nodes and (b) minimizing the hop distance from sensors to the sinks. The intuition behind these observations were presented when obtaining the lower bound. Our objective is to devise a heuristic algorithm which is extends the idea of GreenWave routing so that it takes these observations into account.

It is easiest to explain our new heuristic as an extension of the distributed Bellman-Ford (DBF) algorithm, although it can be implemented more efficiently otherwise. Recall that each node has a shortest distance estimate to some sink node, d, and a parent node, π, as the next hop on its routing path. Initially each sink v has $d_v = 0, \pi_v = v$ and each sensor u has $d_u = \infty, \pi_u = NIL$. In addition, we introduce a *congestion counter* associated with each node, which is a measure of the congestion on this node. Note that it's value does not reflect the exact quantity of the congestion at a node as we defined earlier, but it serves as our heuristic tool to minimize congestion costs along constructed paths in the routing graph.

Nodes communicate their shortest path estimates iteratively to update their paths to the sinks. In the original DBF algorithm, node v broadcasts a message to announce its current shortest path estimate. All neighbors of v then receive the same value d_v and each neighbor u such that $d_u > d_v + w(u, v)$ updates d_u and sets $\pi_u \leftarrow v$. We modify this part of the algorithm as follows. Node u broadcasts a message requesting the shortest path estimates from its neighbors. Upon receiving this message v replies u with the updated cost $d_v + (c_v + 1) * p * \Lambda$, where c_v is the *congestion counter* of v, initialized to 0. If u indeed sets $\pi_u \leftarrow v$ then v increments c_v by 1. It is worth noting that even if u updates its parent to another relaying node later in the execution of the algorithm, the congestion counter at v is never decremented. This ensures that the cost are monotonically increasing and the algorithm is guaranteed to converge, with loop-free paths. Adding the term $(c_v + 1) * p * \Lambda$ when reporting the shortest path estimate has the effect of balancing the congestion among nearby nodes; each time a node selects v as its parent, the cost (shortest path estimate) of v effectively increases and future requests from other nodes to route over node v is discouraged. The multiplier $p\Lambda$ serves to adjust the trade-off between static and congestion costs.

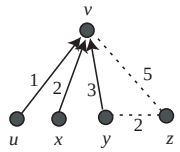


Fig. 10. Demonstration of a problematic case mentioned in the heuristic. At this snapshot of the scenario, $c_v = 3$ and $c_y = 0$.

On the other hand, consider the scenario in Figure 10. There are 3 nodes u,x and y reporting to v, and a fourth node, z, is about to decide between y and v. Suppose that y has selected v as its parent in the first order, hence its local estimate is $d_y = 3 + d_v + p\Lambda$. Since $c_v = 3$ and $c_y = 0$ in the snapshot of Figure 10, z computes the distance $5 + d_v + 4p\Lambda$ through v and $2 + d_y + p\Lambda = 5 + d_v + 2p\Lambda$ through y, thus selects y to report its data. However this introduces extra congestion on node y, and increases the overall cost. Note that data flow from z will eventually reach v and the congestion on v is the same in any case. Hence we need to remedy the heuristic to favor shorter hop paths following the observation (ii-b) we made in the beginning of this section. To achieve this effect, when node v reports its d_v value it also considers the parent's current congestion counter as well as its own counter. This way, the congestion information disseminates to the lower levels along the routing paths. In effect, every time the congestion counter of node v is incremented, the *children* of v increment their congestion counters as well[1]. Hence with this modification, in Figure 10, node z computes the cost of path through y as $2 + d_y + 4p\Lambda = 5 + d_v + 5p\Lambda$, and select v as its parent instead of u.

At each round of the algorithm, each node requests distance

[1]In a practical distributed implementation, nodes synchronize the congestion value with the parent only when they report their distance estimate, hence a single increment operation never needs to spread to all descendants.

TABLE I

COMPARISON OF THE OPTIMAL QIP SOLUTION THE RESULTS OF THE HEURISTICS AND LOWER BOUND (LB).

p	LB	OPT	HR1	HR0
0	42.76	42.76	42.76	42.76
0.1	79.44	88.42	94.64	97.06
0.3	152.80	171.42	190.12	205.66
0.5	226.16	253.06	287.64	314.26
0.7	299.52	334.54	384.70	422.86
1.0	409.56	456.70	536.54	585.76

estimates from neighbors and reports its local distance estimate as described above. The algorithm terminates after a round in which there is no update on the distance estimate of any node. The upper bound on the number of rounds is n, as in the Bellman-Ford algorithm, since no shortest path length is larger than n. The computational complexity of the heuristic differs from that of Bellman-Ford by just a constant factor.

D. Numerical Results

In this section we test the quality of the lower bound and the heuristic algorithm. We refer to our original GW routing algorithm (which assumes data fusion, hence ignores contention) as *Heuristic0*, or HR0, and the new heuristic proposed in the previous section as *Heuristic1*, or HR1.

For the results of the *heuristic0* and *heuristic1*, we compute the objective function value of the QIP over the paths returned by these algorithms. Each data point in Table I and Figure 11 represents the results averaged over 100 random networks.

For very small networks, with 10 sensor nodes and single sink node, we find the optimal solutions to the QIP using the CPLEX optimization software. Table I presents the results obtained, by varying the probability p. Note that when $p = 0$ the problem reduces to the weighted shortest path routing problem and all algorithms return the same result. We observe that the lower bound (LB) is close to the optimal solution, and the heuristic1 performs better than heuristic0, as desired.

For larger networks, we compare the solutions returned by the lower bound, *heuristic0* and *heuristic1*. It is also clear from this figure that *heuristic1* (improved GreenWave routing) performs quite better than *heuristic0* (original GreenWave routing), reducing the gap between the lower and upper bounds for the optimal solution. The lower bound and heuristic (upper bound) algorithms may be used to efficiently bound the optimal solution and measure the performance of a routing protocol proposed for sensor networks without in-network data fusion[2].

VIII. CONCLUSION

We presented efficient routing schemes for wireless sensor networks with contention-free MAC protocols. Two cases were considered; routing with in-network data fusion, and without data fusion. The case with fusion makes the problem easier since there is no queueing delays, hence the intersection of paths (congestion) does not affect the end-to-end delays observed. We presented *GreenWave routing*, a distributed and scalable protocol for wireless sensor networks that forms end-to-end routes from sensors to sinks, based on an underlying TDMA-based MAC protocol. By computational results on random sensor networks, we showed that GreenWave routing significantly decreases the communication delay, and thus increases the throughput of the network under saturation conditions. Moreover, we present an important result that contention-free MAC protocols may perform better than the contention-based 802.11 protocol, when accompanied by our GreenWave routing algorithm. Recall that the contention-free MAC protocols have already the desirable properties for an energy-constrained sensor network, since there is no energy waste due to collisions and retransmissions. Hence having their delay and throughput performance even just as comparable to the 802.11 protocol makes them much more attractive for use in sensor networks. We believe this work will encourage further study on designing efficient contention-free MAC protocols for multi-hop wireless networks.

On the other hand, if in-network data fusion is not available, then the problem becomes harder since we should take into account the interaction among the flows from different sources. We formulated this problem as a QIP, which is expressed as a combined minimization of the *static costs*, as defined for the fusion case, and the *congestion costs*, which depends on the interaction among different flows (routes). Since the QIP can not be efficiently solve for large networks, we provided a lower bound and a new heuristic algorithm to bound the optimal solution from above and below.

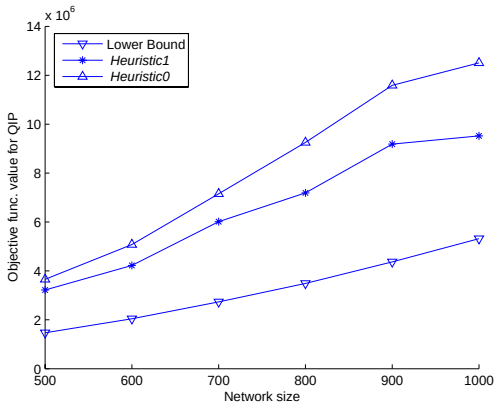


Fig. 11. Comparisons of the solutions by the heuristic and the GreenWave algorithm to the lower bound on larger networks. Transmission radius r and the probability p are both fixed at 0.1 for all network sizes.

ACKNOWLEDGMENT

The authors would like to thank Malik Magdon Ismail for his valuable insights, and Yu Wang and J.J. Garcia-Luna-Aceves for sharing the source codes underlying their model.

[2]All Matlab source codes used for this work are available online at the first author's web site.

REFERENCES

[1] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE standards 802.11, January 1997.

[2] C. Busch, M. Magdon-Ismail, F. Sivrikaya and B. Yener, *Contention-Free MAC protocols for Wireless Sensor Networks*, In proceedings of the 18th International Conference on Distributed Computing (DISC 2004), Amsterdam, The Netherlands, pp. 245–259, October 2004.

[3] F. Sivrikaya and B. Yener, *GreenWave Routing Trees for Wireless Sensor Networks*, Technical Report, Rensselaer Polytechnic Institute, 2005.

[4] T. Herman and S. Tixeuil, *A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks*, ALGOSENSORS 2004, pp. 45–58.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms, Second Edition*, MIT Press, September, 2001, pp. 588–592.

[6] D. Bertsekas and R. Gallager, *Data Networks, Second Edition*, Prentice Hall, Inc., 1992, pp. 404–410.

[7] G. Bianchi, *Performance Analysis of the IEEE 802.11 Distributed Coordination Function*, IEEE Journal on Selected Areas in Communications, Vol.18, No.3, pp. 535–547, March 2000.

[8] J. L. Sobrinho, A. S. Krishnakumar, *Quality-of-service in ad hoc carrier sense multiple access wireless networks*, IEEE JSAC Vol.17, No.8, pp. 1353-1368, 1999.

[9] USC/ISI, *Network Simulator 2 (NS2)*, http://www.isi.edu/nsnam/ns/.

[10] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network Flows (Theory, Algorithms, and Applications)*, Prentice Hall Inc, New Jersey, 1993 (p. 448).

[11] E. D. Kaplan, editor, *Understanding GPS: Principles and Applications*, Artech House, 1996.

[12] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, *A Survey on Sensor Networks*, IEEE Communications Magazine, August 2002.

[13] C. E. Perkins, and P. Bhagwat, *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers*, Computer Communications, pp.234-244, October 1994.

[14] D. B. Johnson, and D. A. Maltz, *Dynamic Source Routing in Ad-Hoc Wireless Networks*, Mobile Computing, T.Imielinski and H. Korth, Eds., Kluwer, pp. 153–181, 1996.

[15] C. E. Perkins, and E. M. Royer, *Ad-Hoc On-Demand Distance Vector Routing*, Proceedings of the 2nd IEEE Workshop on Mobile Comp. Sys. And Apps., pp. 90–100, February 1999.

[16] E. M. Royer, C. Toh, *A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks*, IEEE Personal Communications, pp. 46–55, April 1999.

[17] K. Sundaresan, H. Hsieh, and R. Sivakumar, *IEEE 802.11 over Multi-hop Wireless Networks: Problems and New Perspectives*, Ad Hoc Networks, Vol. 2, No. 2, pp. 109–132, April 2004.

[18] S. Xu, and T. Saadawi, *Revealing the problems with 802.11 medium access control protocol in multi-hop wireless ad hoc networks*, Computer Networks, Vol. 38, No.4, pp. 531–548, 2002.

[19] M. M. Carvalho, and J. J. Garcia-Luna-Aceves, *A scalable model for channel access protocols in multihop ad hoc networks*, Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom '04), pp. 330–344, 2004.

[20] Y. Wang, J. J. Garcia-Luna-Aceves, *Modeling of Collision Avoidance Protocols in Single-Channel Multihop Wireless Networks*, Wireless Networks, Volume 10 , Issue 5, pp. 495–506, September 2004.

[21] F. Cali, M. Conti, E. Gregori, *Dynamic Tuning of the IEEE 802.11 protocol to achieve a theoretical throughput limit*, IEEE/ACM Transactions on Networking, Vol. 8, No. 6, pp. 785–799, December 2000.

[22] H. Takagi and L. Kleinrock, *Optimal Transmission Range for Randomly Distributed Packet Radio Terminals*, IEEE Transactions on Communications Vol. 32 No. 3, pp. 246-257, March 1984.

[23] L. Wu and P. Varshney, *Performance Analysis of CSMA and BTMA Protocols in Multihop Networks: (i). Single Channel Case*, Information Sciences, Vol. 120 pp. 159177, 1999.

[24] R. Kalidindi, L. Ray, R. Kannan, and S. S. Iyengar, *Distributed Energy-Aware MAC Protocol for Wireless Sensor Networks*, International Conference on Wireless Networks, Las Vegas, Nevada, June 2003.

[25] K. A. Arisha, M. A. Youssef, and M. F. Younis, *Energy-Aware TDMA-Based MAC for Sensor Networks*, IEEE Workshop on Integrated Management of Power Aware Communications, Computing and Networking (IMPACCT 2002), New York City, New York, May 2002.

[26] K. Sohrabi, and G. Pottie, *Performance Of A Novel Self-Organization Protocol For Wireless Ad-HocSensor Networks*, IEEE Vehicular Technology Conference, Amsterdam, Netherlands, September 1999.