A Distributed Protocol for Wireless Sensor Networks Based on Multiple-Leader Stackelberg Network Games

Gautam S. Raj and Volkan Rodoplu
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
gautamraj@gmail.com, vrodoplu@ece.ucsb.edu

***Abstract*—While the past literature on game theory rarely addresses the implementation of protocols that converge to Nash equilibria, practical networks must be designed with protocols that correctly address issues of information asymmetries, hysteresis effects due to these asymmetries, and the fact that information can propagate only locally on networks. To this end, we develop a distributed protocol based on multiple-leader Stackelberg network games to efficiently utilize the localized energy resources of a sensor network. Our protocol arrives at a Nash equilibrium of the multiple-leader Stackelberg network game. We demonstrate the performance of our protocol under a large-scale fading model for the internodal links, and quantify its control overhead. Through this work, we find that there are considerable differences between centralized implementations of algorithms that find Nash equilibria on networks, and distributed protocols that can converge to those equilibria in practice.**

***Index Terms*—network, Stackelberg, game, pricing, distributed, protocol**

I. INTRODUCTION

Wireless sensor networks are typically conceived of as collective entities that collect information from an area and relay it to a collection site (a.k.a. base station) via a multi-hop network between the sensor nodes. However, decisions in sensor networks have to be made in a localized and distributed manner. Each node usually has a limited battery supply that it has to conserve, and sees different Joules-per-bit costs along the different links to its neighbors, assuming that each node is able to dynamically adjust its transmit power level to reach its neighbors [1]. In addition, sensor networks typically operate in the low traffic load regime; that is, the key measure is not the achieved bits-per-second throughput, but rather the bits-per-Joule capacity [2] of the network, namely the number of bits that can be sent per Joule of energy to the destination. As a result, in sensor networks, most of the bandwidth goes unused, and end-to-end data transmissions typically occur in an on-demand fashion, initiated either by the source that has just collected some important data, or "pulled" from the source by the destination [3].

Game theory models each node as a selfish, autonomous entity that aims to maximize its own utility. Even though the nodes in a sensor network have as their common objective, the reliable end-to-end delivery of sensor data, from the perspective of distributed, localized protocol design, each node can be modeled as an entity that also aims to preserve its battery resources by reflecting the energy costs it is incurring to transfer the information. Then, the most natural setting in which this locally available information is made visible to the network is through pricing variables that are locally determined by the relay nodes.

The application of game theory to routing problems in wireless sensor and ad hoc networks is not new. Sadagopan et al. [4] show the construction of an energy-balanced tree of sensor nodes, modeling each sensor as a selfish entity. Nurmi [5] models energy-constrained routing in ad hoc networks made up of selfish nodes, and lets the source send along the best path based on its subjective beliefs about the amount of remaining energy at the relay nodes. Liu et al. [6] assume that each node forwards with some probability, and take the end-to-end reliability to be the product of these. Under a single-source, single-destination model, they develop a polynomial-time method for deriving a Nash equilibrium routing path. Sengupta et al. [7] apply non-cooperative game theory to power control problems in wireless sensor networks, taking each sensor as a selfish entity. Similarly, Campos-Nanez et al. [8] develop a game-theoretic approach to power management in sensor networks, and Kannan et al. [9] model wireless sensors in a routing game to achieve reliable, energy-constrained routes through the sensor network. Felegyhazi et al. [10] examine cooperative packet forwarding in a game-theoretic framework in multi-domain sensor networks.

None of the above works consider the incorporation of the source's utility function into the decisions of a relay node, as in a Stackelberg framework. In the past, Stackelberg games have been to applied to wired networks with a single Internet Service Provider (ISP) [11]–[13], where the ISP (or a group of multiple ISPs [14]) is the Stackelberg leader, and the network users are the followers. More recently, this model has also been applied to wireless networks [15], again under a single-leader setting. In contrast to these single-leader settings, this paper utilizes the framework of a "multiple-leader Stackelberg game", introduced in [16], in which each relay node acts as a leader that anticipates the response of the source node that is currently initiating traffic to the base station.

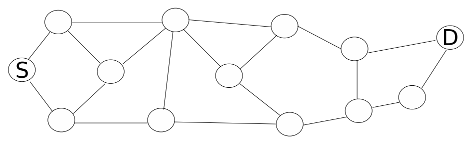


Fig. 1. General network topology.

The main contribution of the current paper is the design of a distributed protocol that converges to a Nash equilibrium of the multiple-leader Stackelberg game. In fact, the Nash equilibrium to which our protocol converges is a Pareto-optimal Nash equilibrium; that is, it is not possible to increase the utility of any one of the users without decreasing the utility of another user. (This fact cannot be proved within the scope of this paper.) Pareto-optimality generalizes scalar capacity to multiple dimensions. Nash equilibria are desirable because they constitute stable operation points from the perspectives of the users in a system. A Nash equilibrium implies that a user cannot benefit from unilateral deviations (i.e., deviations by itself) from the operation point of the network. While this is clearly desirable when users have competing objectives (as in non-cooperative games), it is also desirable in the implementation of *distributed* protocols on sensor networks. In this paper, we model each relay node as an autonomous entity that aims to maximize its own utility. However, the Stackelberg formulation by which each relay node takes into account the utility of the source node creates the link that ties that objectives of the nodes together. Hence, the Stackelberg feature is what makes the protocol useful for application to sensor networks.

While the past literature on the application of game theory to networks rarely addresses the *implementation* of protocols that converge to Nash equilibria, practical networks must be designed with protocols that correctly address issues of information asymmetries between the nodes, hysteresis effects due to these asymmetries, and the fact that information can propagate only locally in these networks. This paper makes an important contribution in this regard in that it explicitly addresses the design of a distributed protocol that overcomes these effects, and further quantifies the overhead of this design in order to show that it is viable. Through this work, we show that there are considerable differences between centralized implementations of algorithms that find Nash equilibria on networks, and those that design distributed solutions.

The rest of this paper is organized as follows: In Section II, we describe our utility and network models. In Section III, we design a distributed network protocol that converges to a Nash equilibrium of the multiple-leader Stackelberg game on this network. In Section IV, we analyze the performance of this protocol when it is applied to energy-limited, ad hoc wireless networks. In Section V, we present our conclusions.

II. Models and Assumptions

In this paper, we consider a sensor network of N nodes on an arbitrary topology G. Since the data generation rates in typical sensor networks (e.g., for habitat monitoring and sensing) can be as low as a few kilobytes per hour, and commercial modems (such as Crossbow [17]) typically operate at relatively high data rates such as 250 Kbits per second, bandwidth usage is not an issue; however, energy consumption is of paramount importance. Due to the low data generation rates, we can focus on the data from a single source to the destination at a time, as shown in Fig. 1. On this topology, the Joules-per-bit cost from each node to any one of its neighbors is fixed, and is independent of the concurrent transmissions on the other links. The constant Joules-per-bit costs (rather than e.g., convex costs) hold if the modulation scheme is fixed (e.g., adaptive modulation is not used). The non-interference assumption is well-justified in the low data rate sensor network scenario (see [18] for a full justification, with numerical examples from sensor network applications.)

Throughout the paper, we focus on a single source at a time. This is possible because the ratio of the total average generated data rate to the available bandwidth is very low in sensor networks. We label the source node as node 1, and the destination node as node N. We let $\aleph$ denote the node set, and $\mathcal{R}$ denote the set of relay nodes; that is, $\mathcal{R} = \{2, \ldots, N-1\}$. In order to reflect back to the source the average energy costs that it bears, each relay node i charges a price p_i per bit, independently of which outgoing link is used for the transmission. Each p_i is a variable under the control of the relay node i. The total price along any path h from the source to the destination is the sum of the prices on h, and this price is assumed to be borne by the source node. These prices merely constitute a distributed scheme to organize and efficiently utilize the network's energy resources.

We focus on the utility of the source node each time that it has accumulated sufficient sensor data to be sent to the destination. However, as mentioned earlier, since the prices reflect the energy costs of transmission to the relay nodes, the source node may end up sending a variable number of bits to the destination, trading off the energy cost of transmission in the relay nodes against the utility gained by sending sensor data to the destination. As in [16], we model the utility of the source node as

$$u_1(b_1) = M_1 b_1^{1/\alpha} - p_{best}^{total}(\mathbf{p}) b_1, \alpha > 1, b_1 \geq 0 \qquad (1)$$

In the above equation, the variable b_1 is the number of bits that the source node chooses to send through the network to the destination node. We call parameter $M_1 > 0$ the "amplitude" of the utility function. The physical meaning of this parameter is that it determines how much value we place on sending sensor data end-to-end. Making this parameter larger means that we place more value on this, relative to the prices that are charged (which reflect the relay nodes' energy costs). The parameter $\alpha > 1$ governs the curvature of the source's utility function. Physically, a small α means that the benefits of sending more bits continue to remain large even as b_1 increases.

The source node always chooses the path with the lowest price to the destination. Above, $p_{best}^{total}(\mathbf{p})$ is the total price of the lowest price path from the source to the destination. It is a function of the vector $\mathbf{p}$, which is the vector of all of the prices of the relay nodes in the network. Node 1 always acts to maximize its own utility.

The second part of our model focuses on the utility function of each relay node. We assume that a relay node i gains a utility of p_i for each bit that it transmits along one of its outgoing links, and loses a utility equal to the cost of transmitting along that link. Let h_i denote a path of links from the source to the destination, such that the path goes through the node i. Let $c_{i,next}(h_i)$ denote the cost per bit incurred by i to transmit on the link that goes out from i and that falls on the path h_i. Note that this cost may depend on both the Joules-per-bit link cost as well as the remaining battery energy of the relay node. Hence, the notion of cost c_{ij} is general. Then, the utility of relay node i is

$$u_i(p_i) = (p_i - c_{i,\mathit{next}}(h_i))b[h_i] \tag{2}$$

where $b[h_i]$ is the number of bits that the source node chooses to send through node i, along the path h_i. We see that in order to achieve a positive utility, it is necessary that node i set $p_i > c_{i,\mathit{next}}(h_i)$.

Now, one of the key assumptions in our framework is that even though prices are used to arrive at a distributed management of localized resources in a sensor network, overall, the sensor network represents a collective effort to send sensor data end-to-end from the source to the destination. Hence, it is to the network's advantage to incorporate the form of the utility function of the source node into the relay node's decisions. Such schemes are generally referred to as "Stackelberg games" where the leader incorporates into its own utility function, the form of the utility function of the follower. Here, the source node is the follower, and each of the relay nodes acts as a leader, hence, resulting in the novel form of a multiple-leader Stackelberg game. Based on this discussion, the utility model of a relay node i is given by

$$u_i(p_i; p_{-i}) = (p_i - c_{i,\mathit{next}}(h_i))b_1[h_i](p_i; p_{-i}) \tag{3}$$

where p_{-i} denotes the set of prices of all of the relay nodes besides i, and $b_1[h_i](p_i; p_{-i})$ is the number of bits that Node 1 sends through node i via path h_i, after it has chosen the lowest price path and the number of bits to send through that path, via its maximization of its own utility in (1). (Note that if node i is not on the lowest price path, then $b_1[h_i](p_i; p_{-i}) = 0$; that is, no bits are sent through node i.)

III. Distributed Protocol Design

In [16], we described a centralized algorithm to find a Nash equilibrium of the network over an arbitrary topology G of relay nodes. In this paper, we describe a distributed network protocol that the nodes can use in practice to converge to a Nash equilibrium. The distributed protocol that we present converges to a Nash equilibrium because it implements the centralized algorithm of [16] whose convergence to a Nash equilibrium was proved. The main idea behind the convergence of this protocol is as follows: we designed an algorithm on a general topology that converged to a Nash equilibrium by solving a set of price equations via the Jacobi method. The same Jacobi method is implemented via the distributed protocol, as will be seen shortly for the serial network. After this, the competition that occurs between parallel paths within the network is modeled by the protocol's dynamically placing caps on the prices on the currently winning path via the constraints that occur due to the competition paths. The main challenge that we have to address in this setting is that each node can communicate only with its neighbors on the topology G; hence, no global information channels that can announce all of the prices of the relay nodes to each other exist. Further, the source node can become aware of the relay node prices only through its own links on G, and any announcements by the source node, of the best current bid (that is, the best current lowest price path) must propagate to all of the relay nodes via the links on G.

Finally, the control overhead of the resulting network protocol must be small enough to justify its use in sensor networks. We shall demonstrate this in the next section.

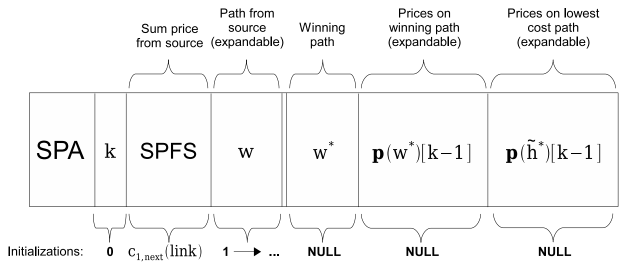


Fig. 2. SPA Packet

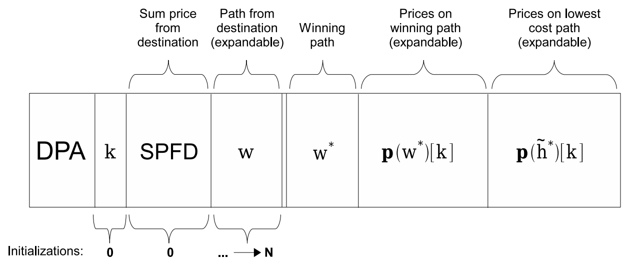

Sum price from destination
Path from destination (expandable)
Winning path
Prices on winning path (expandable)
Prices on lowest cost path (expandable)
DPA
k
SPFD
w
w*
p(w*)[k]
p(h̃*)[k]
Initializations:
0
0
... ⟶ N

Fig. 3. DPA Packet

A. Distributed Protocol on a Serial Network

For ease of exposition, we begin by developing the protocol first for the serial network of Fig. 4. In a serial network, every relay node needs to update its price p_i according to (see (11) in [16]):

$$\forall i \in \mathcal{R} : p_i^* = (\alpha - 1)\Big[\sum_{j \in \mathcal{R}\setminus\{i\}} p_j^* + c_{12}\Big] + \alpha c_{i,i+1} \tag{4}$$

The sum of the prices of all of the other relay nodes can be efficiently accumulated as follows: The source node initiates a "Source Price Accumulate" (SPA) packet, and sends it on its link to Node 2 in Fig. 4. The ultimate destination of this packet is Node N. The structure of this packet is shown in Fig. 2. The second field is the iteration number k of the protocol, which is set to 0 for the first SPA that the source ever sends out. The third field of this packet is the "Sum Price From

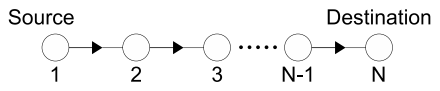


Fig. 4. An N-node serial network.

Source" (SPFS), which accumulates the total sum price from the source node thus far. Node 1 initializes this field to c_{12}, which is the cost of the link from 1 to the next node on which this SPA is being sent. (This is denoted by $c_{1,next}(w)$ in the figure, where w is the path that this SPA travels on.) We shall now describe the events for the kth iteration. When Node 2 receives the SPA, it records the price accumulated from the source thus far, and adds its own current price $p_2[k]$ to this field. In the 0th iteration, $p_2[0] = c_{23}$, which is the cost of the link to the next node. The fourth field w is an expandable list, that contains the path through which the SPA has travelled thus far. Hence, Node 2 also adds its node ID to the path w in the fourth field, and sends the SPA to Node 3. Node 3, and all of the relay nodes in this sequence continue in a similar fashion.

When the SPA packet has reached the destination node N, all of the relay nodes have accumulated the sum of the prices of the relay nodes to their left on the serial network, via the SPFS field. When Node N receives the SPA, it records the path w which contains all of the nodes through which the SPA travelled, as well as the total price, namely the value of the SPFS field, accumulated through the entire path. Then, Node N initiates a "Destination Price Accumulate" (DPA) packet toward the source node. The structure of the DPA packet is shown in Fig. 3. The third field in the DPA is the "Sum Price From Destination" (SPFD), which accumulates the sum of the prices on the *forward* links toward the destination. Because there is only one path from the source to the destination in a serial network, Node N sets w^*, the "winning path" field of the DPA, to the path w of the SPA that it has just received. Note that the sixth field of the DPA, $\mathbf{p}(w^*)$ is an expandable vector of prices. This field is initially empty, and will be updated by the relay nodes on the winning path w^*, as the DPA travels back.

When a relay node i receives the DPA, it records the value of the SPFD field. At this point, the relay node i has the sum of the prices of all of the other relay nodes, obtained from the SPFS field of the SPA, and the SPFD field of the DPA. Thus, it now updates its price $p_i[k+1]$ according to (4). Then, it adds its current price, $p_i[k+1]$ to the SPFD field, and sends it toward the source node. Hence, when the DPA has reached the source node, all of the relay nodes have updated their prices. Further, the source has just received the winning path w^* from the fifth field of the DPA, as well as all of the *new* prices on the winning path, namely $\mathbf{p}(w^*)$ from the fifth and sixth fields of the DPA.

After the source has received the DPA of the kth iteration, it first checks whether the prices on the winning path w^* have converged. It does this by taking the norm of the difference between the prices on w^* in the current and the previous iterations. If the prices on w^* have converged, then

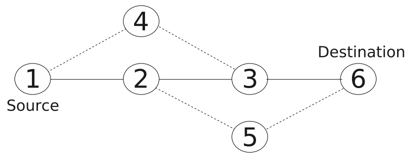


Fig. 5. Example topology.

it terminates the SPA-DPA exchanges, and announces the final path w^{**} via a separate packet to all of the nodes. Otherwise, it prepares a new SPA packet, with iteration number $k+1$ in the second field, and with the currently winning path w^* that it has copied from the DPA into the fifth field of the SPA, and sends this SPA toward the destination.

B. Distributed Protocol on the General Topology

The distributed protocol on the general topology will implement a distributed version of the centralized algorithm in Fig. 3 of [16]. The centralized algorithm has access to all of the constraint equations on the least cost path $\tilde{h}^*$. In contrast, in the distributed protocol, the nodes do not know the $\tilde{h}^*$ path a priori, and have no global picture of the constraints on the $\tilde{h}^*$ path. Hence, both $\tilde{h}^*$ and the price constraints have to be discovered dynamically, while the price competition takes place among the relay nodes. A part of the main structure of the protocol is the SPA-DPA exchanges described for the serial network in the previous subsection. The pseudocode for the distributed protocol is shown in Figs. 6-9. On a general topology, on line 12 of Fig. 6, the source initiates an SPA along each of the paths that emanate from it. On lines 16-26 of Fig. 7, when a relay node i receives an SPA, it first checks the accumulated price thus far (namely the value of the SPFS field) against *minSPFS* which is an internal variable of i, set to the minimum SPFS value observed so far. If the value of the SPFS field is greater than the value of *minSPFS*, then it discards the SPFS field of this SPA. Otherwise, a better total price from the source up to this relay node has been discovered; hence, *minSPFS* is updated to the value of the SPFS of this SPA.

We shall now describe the protocol for the example network in Fig. 5. We define an iteration as one SPA-DPA exchange from the source back to the source. In iteration 0, the source starts the competition round by sending an SPA to Nodes 2 and 4. Nodes 2 and 4 record the current path w, and the SPFS from the SPA. After storing these fields, they append their node IDs to the w field, and add their link costs to the SPFS fields of their respective SPAs. In this example, Node 4 will add its current price (which is c_{43} in this iteration), while Node 2 will add its current price (c_{23} in this iteration) when sending to node 3, and c_{25} when sending to Node 5. Then, they forward the SPAs to their next nodes, who repeat the process, until all of the SPAs reach the destination. The set of "next nodes" of any node i is defined as the set of all of its neighbor nodes except the ones in the w field of the SPA. (This prevents loops.) These actions are shown on lines 16-26 of Fig. 7.

For the destination node, the *Receive* function on line 8 of

```
1  // Internal Variables
2  int K_MAX=_K_MAX; // max iterations
3  int J_MAX=_J_MAX; // max nodes
4  minSPFD[K_MAX] = [LARGE,...,LARGE]; //min sum price from dest.
5  minW = NULL; //min price route from destination
6  $k = 0$; //sequence number of SPA
7  p[J_MAX][K_MAX];
8  $\delta$ = DELTA;
9  $p[*][-1]$ = LARGE;
10 links = this→allOutgoingLinks;
11 // Main Function
12 send_SPA(links,$k$,$c_{1,next}$(links),1,NULL,NULL,NULL);
13 $k$++;
14 while (TRUE) {
15   while (Receive(DPA[$k$])) { //Receive all DPA's for kth iteration
16     [spfd, $w$; $w^*$, $p[w^*][k]$, $p[\tilde{h}^*][k]$] = ExtractFields(DPA[$k$]);
17     if (spfd < minSPFD[$k$]) {
18       minSPFD[$k$]=spfd;
19       minW = $w$;
20     }
21   } // Got all DPAs for iteration $k$
22   $w^* = w$;
23   if ($k == 0$)
24     $\tilde{h}^* = w$;
25   else if ($||\mathbf{p}[w^*][k] - \mathbf{p}[w^*][k-1]|| < \delta$ | $k ==$ K_MAX) {
26     send($w^{**}$, links); //announce final path
27     break;
28   }
29   send_SPA(links,$k$, $c_{1,next}$(links), 1, $w^*$, $p[w^*][k]$, $p[\tilde{h}^*][k]$);
30   $k$++;
31 }
```

Fig. 6. Distributed protocol executed by the source node.

Fig. 9 has a built-in timer that stops listening for new SPA packets after a timeout has been reached. The timeout must be as long as needed to receive the SPA packets on all of the closest competition paths, and depends on the network size N. If all of the nodes were in a linear arrangement, then the timeout must grow linearly with N. If the nodes are randomly deployed on a square region, then, since the average number of hops from a source at one end to a destination at another grows as $\mathcal{O}(\sqrt{N})$, the timeout must grow in this fashion with respect to N, with the coefficient of the growth larger for a smaller probability that the SPA's from not all of the paths may have arrived by that time. (A design that aims at no loss from optimality uses a design that grows linearly with N in all cases.)

In iteration 0, once the destination receives all the SPAs, it computes the lowest-cost path $\tilde{h}^*$, by finding the minimum SPFS, and using the corresponding w. Then, the destination creates a DPA and sends it toward the source, on nodes 3 and 5 in Fig. 5, as shown on line 15 of Fig. 9. In iteration 0, each relay node that receives the DPA records the lowest-cost path $\tilde{h}^*$, from the w^* field of the DPA, as shown on lines 29-31 of Fig. 7. Each relay node also adds its node ID to the w field, and its current price (which is its link cost in iteration 0) to the SPFD field (lines 59-66). At this point, the node can compute its price for the next iteration, using (4). If the node sees from the DPA that it is on w^* for the 0th iteration, it also appends its price to the $\mathbf{p}(w^*)[0]$ and $\mathbf{p}(\tilde{h}^*)[0]$ fields of the DPA (line 60). This is done at each node for each DPA packet that it receives. Once the source receives all the DPAs, it must determine the winning path w^* for iteration 0. To do this, as shown on lines 15-22 of Fig. 6, it finds the minimum SPFD value from all the DPAs, and records the corresponding path

```
1  // Internal Variables
2  int K_MAX=_K_MAX; // max iterations
3  int J_MAX=_J_MAX; // max nodes
4  minSPFS[K_MAX] = [LARGE,...,LARGE] // min sum price from source
5  minW_FS[K_MAX] = [NULL,...,NULL] // min price path from source
6  minSPFD[K_MAX] = [LARGE,...,LARGE] // min sum price from dest
7  minW_FD[K_MAX] = [NULL,...,NULL] // min price path from dest
8  $w^*$[K_MAX]; // winning path for iteration $k$
9  $\tilde{h}^*$ = $w^*$[0]; // lowest cost path
10 priceLocked = FALSE;
11 nodesConstrained = $\emptyset$;
12 //Main Function
13 $i$ = this→nodeID;
14 links = this→allOutgoingLinks;
15 while (TRUE) {
16   while (Receive(SPA[$k$])) {
17     [spfs,w_SPA,$w^*[k]$, $p[w^*][k]$, $p[\tilde{h}^*][k]$] = ExtractFields(SPA[$k$]);
18     if ($i \notin$ w_SPA) { // Prevent loops
19       if (spfs < minSPFS[$k$])
20         minSPFS[$k$] = spfs;
21       foreach (link $\in$ links)
22         if (link→nodeID == nextNode($\tilde{h}^*$))
23           send_SPA_Link(link,$k$,spfs + $p[i][k]$, w_SPA+→ $i$);
24         else
25           send_SPA_Link(link,$k$,spfs + $c_{i,next}$(link),w_SPA+→ $i$);
26     }
27   }
28   while (Receive(DPA[$k$])) {
29     [spfd, w_DPA, $w^*$] = ExtractFields(DPA[$k$]]);
30     if ($i \notin$ w_DPA && spfd < minSPFD[$k$])
31       minSPFD[$k$] = spfd;
32     if (($i \notin \tilde{h}^*$) || (nextNode(w_DPA) != nextNode($\tilde{h}^*$)))
33       $p[i][k] = c_{i,next}$(w_DPA); // Return link cost
34     else if ($i \in w^*[k]$ && nextNode(w_DPA) $\in w^*[k]$) { // On $w^*[k]$
35       k_last = $k$; // Store last time on best path
36       if (priceLocked)
37         $p[i][k+1] = p[i][k]$;
38       else if ((nodesConstrained$\backslash w^*[k]$) ! = $\emptyset$) {
39         $p[i][k+1] = p[i][k]$;
40         willLock = FALSE;
41       }
42       else if (nodesConstrained $\subseteq w^*[k]$ && willLock) {
43         priceLocked = TRUE;
44         $p[i][k+1] = p[i][k]$;
45         nodesConstrained = $\emptyset$;
46       }
47       else if ($\tilde{h}^* \cap w^*[k] \neq \emptyset$)
48         $p[i][k+1] = p[i][k]$;
49       else { // Node is not locked, so unfreeze price updates
50         SumOtherPricesOnPath[$k$] = minSPFS[$k$] + minSPFD[$k$];
51         $p[i][k+1] = (\alpha - 1)$SumOtherPricesOnPath[k]+$\alpha c_{i,next}(\tilde{h}^*)$;
52       }
53     }
54     else { // Not on $w^*[k]$
55       [$\epsilon$, nodesConstrained] =
           Surplus($w^*$[k_last], $w^*[k]$, $p$[][k_last], $p[][k]$, $i$);
56       $p[i][k+1] = p[i][k] + \epsilon \sum_{j \in \text{nodesConstrained}} (p[j][k] - p[j][\text{k\_last}])$
57       willLock=TRUE;
58     } // Finished computing price, now send DPA
59     if ($i \in w^*$ && $i \in \tilde{h}^*$)
60       send_DPA(links,$k$,spfd+$p[i][k]$,(w_DPA)+→ $i$, $p[i][k]$, $p[i][k]$);
61     else if ($i \notin w^*$ && $i \in \tilde{h}^*$)
62       send_DPA(links,$k$,spfd+$p[i][k+1]$,(w_DPA)+→ $i$,NULL, $p[i][k+1]$);
63     else if ($i \in w^*$ && $i \notin \tilde{h}^*$)
64       send_DPA(links,$k$,spfd+$p[i][k]$,(w_DPA)+→ $i$, $p[i][k]$,NULL);
65     else
66       send_DPA(links,$k$,spfd+$p[i][k]$,(w_DPA)+→ $i$,NULL,NULL);
67   }
68   if (Receive($w^{**}$)) {
69     if ($i \in w^{**}$) lockedPrice = $p[i][k]$;
70     break;
71   }
72 }
```

Fig. 7. Distributed protocol executed by each relay node.

w as w^*.

In iteration 1, the source initiates the SPA using the w^* and the node prices from iteration 0. Once a relay node i receives an SPA, it appends its price $p_i[0]$ to the SPFS field

```
1   [$\epsilon$,nodesConstrained]=Surplus($w^*(k-1), w^*(k), p[][l], p[][k], i$) {
2     [oldPath,newPath] = FindRemovedSegment($w^*(l)$, $w^*(k)$, $i$);
3     cap = $\sum_{j \in \text{newPath}} p[j][k]$;
4     $\epsilon = \frac{\text{cap} - \sum_{j \in \text{oldPath}} p[j][l]}{\sum_{j \in \text{oldPath}} (p[j][k] - p[j][l])}$;
5   }
```

Fig. 8. Surplus subroutine for the relay nodes.

```
1    // Internal Variables:
2    K_MAX = _K_MAX;
3    minSPFS[K_MAX] = [LARGE,...,LARGE];
4    $k = 0$;
5    // Main function
6    links=this→allOutgoingLinks;
7    while (TRUE) {
8      while (Receive(SPA[$k$])) {
9        minSPFS = LARGE;
10       [spfs, $w$, $p(\tilde{h}^*)$] = Extract(SPA[$k$]); // Uses Sequence No. of SPA
11       if (spfs < minSPFS[$k$]) {
12         minSPFS[$k$] = spfs;
13         $w^* = w$;
14       } // Got all SPAs for iteration $k$
15         send_DPA(links,$k$,0,this→nodeID,$w^*$,NULL,NULL);
16       $k$++;
17       }
18     }
```

Fig. 9. Distributed protocol executed by the destination node.

and its node ID to the w field, and forwards the SPA to its next nodes (lines 16-26, Fig. 7). The destination node now computes w^* for iteration 1, by finding the minimum SPFS value, and sends out DPAs on all of its outgoing links (lines 8-17, Fig. 9). When a relay node receives a DPA packet for iteration 1, it checks if it is on w^*, and if so, it appends its price for the current iteration to $\mathbf{p}(w^*)[k]$ and $\mathbf{p}(\tilde{h}^*)[k]$. Then, it computes its price for the next iteration, and forwards the DPA to its next nodes (lines 59-66, Fig. 7).

Returning to our example in Fig. 5, suppose that in iteration k, Node 2 receives a DPA in which Node 2 is *not* on w^*. In this case, because Node 2 is on $\tilde{h}^*$, it will append its price for iteration k to the SPFD and $\mathbf{p}(\tilde{h}^*)[k]$ fields, and forward it to all of its next nodes. Node 2 would like to lower its price to return to w^*, but it does not know the constraint cap imposed on it; that is, it does not know $c_{14} + c_{43}$. Hence, it must wait for the SPA in iteration $k+1$ to travel on the path $1 \to 4 \to 3 \to 6$, and then for a DPA in iteration $k+1$ to inform Node 2 of the prices of the nodes in the constraint, which is only Node 4 in this case. Once Node 2 receives the DPA for iteration $k+1$, it computes the cap from $\mathbf{p}(w^*)[k]$, and interpolates its price back using the Surplus subroutine of Fig. 8. Then, it immediately adds this new price to the SPFD and $\mathbf{p}(\tilde{h}^*)[k]$ fields of the DPA, and forwards it to all of its next nodes. This is done to ensure that Node 2 can return to w^* as quickly as possible. Since this is the only constraint imposed on it in Fig. 5, Node 2 will return to w^* in iteration $k+2$. Because the constraint cap is satisfied with equality, Node 2 can now lock its price for the remainder of the iterations (lines 54-58, Fig. 7).

In general, we may have multiple relay nodes that together exceed some constraint cap. In this case, we would like all the nodes that are capped by the same constraint to act together to reduce their prices, such that the sum of their prices is exactly equal to the cap. Any contiguous segment of nodes that is removed from $\tilde{h}^*$ has encountered some constraint imposed on it. We denote this segment of nodes on $\tilde{h}^*$ by $\mathcal{R}_1$, and the constraining set of nodes by $\mathcal{C}_1$. In the Surplus subroutine of Fig. 8, *FindRemovedSegment* on line 2 returns the contiguous set of removed nodes $\mathcal{R}_1$, of which relay node i is a member. On line 3, node i sums the prices of the nodes in $\mathcal{C}_1$ to determine the constraint cap that is imposed on it, and on line 4, it computes the surplus.

When multiple nodes exceed this constraint cap, increasing any of the nodes' prices would violate the cap; hence, we freeze the price updates for all the nodes on $\mathcal{R}_1$ (lines 42-46, Fig. 7). However, after this, if only a subset of the nodes in $\mathcal{R}_1$ has returned to w^*, this implies that there is another constraint acting on them (lines 38-41, Fig. 7). We denote this smaller subset by $\mathcal{R}_2$, and the new set of constraining nodes by $\mathcal{C}_2$. At this point, we temporarily freeze the prices of all the nodes that have returned to w^*, namely $\mathcal{R}_1 \setminus \mathcal{R}_2$. Then, the nodes in $\mathcal{R}_2$ interpolate their prices again (lines 54-58, Fig. 7), using the new segment of contiguous nodes and the new cap, imposed by $\mathcal{C}_2$. Let m be the number of nodes on segment $\mathcal{R}_1$. Then, after $\mathcal{O}(m^2)$ applications of this procedure, all of the relevant constraints for $\mathcal{R}_1$ will have been discovered, and hence, all the nodes on $\tilde{h}^*$ will have returned to w^*.

Although all the nodes on $\mathcal{R}_1$ have returned to w^*, their prices are not in Nash equilibrium: Even though we initially set the sum of the prices for the nodes on $\mathcal{R}_1$ equal to the cap $\mathcal{C}_1$ with equality, at the end of the above procedure, this is no longer the case. In fact, only the last set of nodes in the procedure to return to w^* are set exactly equal to the final cap. In Fig. 7, the variable *willLock* is TRUE only for this set of nodes. Hence, we permanently lock the prices of the last nodes to return to w^*, and allow all the other nodes in $\mathcal{R}_1$ to resume monopoly pricing (lines 42-46, Fig. 7). This process continues for the rest of the nodes until the prices of the nodes have converged, or until K_{MAX} has been reached (lines 25-28, Fig. 6).

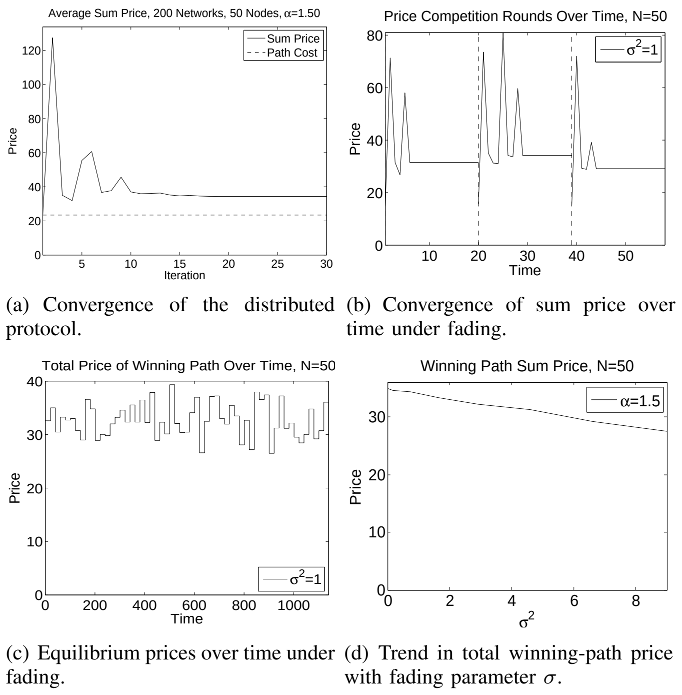
IV. Simulation Results

We simulate the distributed protocol of Section III, using the same random node placement and link costs as of the centralized protocol simulations, as in [16]. The dominant factor in the convergence to Nash equilibrium is the number and structure of the competing paths; however, it is difficult to characterize analytically the dependence of the convergence rate upon such a complex factor. As a remedy, we examine the convergence rate of the protocol via the simulation studies in this section. It should be noted that the simulation studies in this section should be read as a sequel to the simulation studies in [16]. We also compute the control overhead of our protocol and show that it is reasonable for use in sensor networks. Figs. 10(a) and 10(d) show ensemble average results taken over 200 simulations with $N = 50$ nodes, and $\alpha = 1.5$, while Figs. 10(b) and 10(c) display particular realizations. For Figs. 10(a) and 10(d), an average over 200 simulations is sufficient since both the oscillations of the price in Fig. 10(a), and the monotonic decrease of the price of the winning path in Fig. 10(d) display their clear trends.

Fig. 10(a) shows the price of the currently winning path w^* as a function of the number of iterations, when the distributed protocol of Section III is used. We see three successive peaks, which correspond to price-adjusting cycles. The first peak corresponds to the price increase in the first iteration. Because the price of all the nodes are initially free to change, this peak corresponds to every node's incrementing its price, and consequently is the largest. Each successive peak is lower, because as more nodes lock their prices, there are fewer nodes raising their prices. In addition, some of the networks we average converge within 1 or 2 price-adjusting cycles, thereby reducing the average height of the latter peaks. It takes 18 iterations for the average price to converge. The difference between the final sum price and the initial path cost is the ensemble average of the sum of the surplus of all relay nodes.

We calculate the control overhead of our protocol as follows: Let n_p denote the number of price iterations until convergence. Per iteration, each relay transfers 1 SPA and 1 DPA packet. Let N denote the number of nodes in the network. Then, $\lceil \log_2(N) \rceil$ bits are needed to encode the node IDs. The maximum possible number of hops in the network is $N-1$. Let b_p denote the number of bits of quantization for the price, in fixed-point representation. Then, in the fields of an SPA (or DPA) packet, it takes $\lceil \log_2(n_p) \rceil$ bits to encode the number of iterations so far, $\lceil \log_2(N) \rceil$ bits times $\lceil N-1 \rceil$ hops to encode the path of node IDs, the same number of bits to encode the winning path, b_p bits times $N-1$ hops to encode the prices on the winning path, and b_p bits times $N-1$ to encode the prices on the lowest cost path. This worst-case control overhead analysis thus shows that a total of $\lceil \log_2(n_p) \rceil + 2(N-1)\lceil \log_2(N) \rceil + 2b_pN$ bits per SPA (or DPA) packet. Hence, per iteration, each relay node transfers a control overhead of twice this amount (1 SPA and 1 DPA), that is, a total of $2(\lceil \log_2(n_p) \rceil + 2(N-1)\lceil \log_2(N) \rceil + 2b_pN)$ bits of control overhead until convergence to the equilibrium prices. Hence, the control overhead grows as $\mathcal{O}(\mathcal{N} \log_{\in}(\mathcal{N}))$ for large-scale sensor network deployments. However, assuming midsize deployments, the exact numbers might be more important than asymptotic growth. For example, for a network of 100 sensor nodes, and with $b_p = 8$ bits, and $n_p = 10$ iterations, the control overhead is 5980 bits. Let f be the fraction of the tolerable control overhead compared with sensor network data. Then, if $f = 0.01$, then 598 kbits of sensor data need to be accumulated for the control overhead to be justified.

Fig. 10(b) shows the effects of large-scale fading on successive price competition rounds when the distributed protocol is used. We have used a log-normal distribution to model fading on the links between the relay nodes. (Since the sensor nodes are stationary in many settings, the source of the lognormal shadowing variations that we model are due to the variations of the obstacles in between, e.g., in an urban setting. When sensor nodes are placed over such a terrain, the channel between the nodes is rarely static, but rather show significant variations due to the variations in the urban clutter



(a) Convergence of the distributed protocol.
(b) Convergence of sum price over time under fading.
(c) Equilibrium prices over time under fading.
(d) Trend in total winning-path price with fading parameter σ.

Fig. 10. Simulation results for the distributed protocol.

in between.) Specifically, the link costs are recomputed each round as $c_{ij} = d_{ij}^4 \times 10^{(L_{ij}/10)}$, where $L_{ij} \sim \mathcal{N}(0, \sigma^2)$, and are independent for each link ij, and at each round. It is assumed that the coherence time is longer than the convergence time; that is, the link costs are fixed within each competition round. We see that for the same network, varying link costs can have an impact on both the equilibrium price, and the rate of convergence. In the second competition round, three price-adjusting cycles are needed to reach a Nash equilibrium, whereas only two are needed in the first and third. (Note that in our control overhead calculation, we allowed for up to 10 iterations, and the simulations show that this maximum still holds under fading. Hence, the control overhead calculation is still valid.) Fig. 10(c) shows the variation of equilibrium prices under fading.

In Fig. 10(d), we see that large-scale fading reduces the total price of the winning path. As the variance of the L_{ij} increases, the winning-path sum price decreases. Because large-scale fading reduces the costs of some of the links in the network, the protocol is able to take advantage of this and find new lower-cost paths as σ increases.

V. Conclusion and Future Work

We have presented a distributed protocol for wireless sensor networks, based on multiple-leader Stackelberg games. These games allow the relay nodes, each of which acts as a Stackelberg leader, to incorporate the utility function of the source into their utilities. We have designed a distributed protocol to arrive at a Nash equilibrium of the game. We have also shown the convergence of the protocol to the Nash equilibrium as a function of time, and quantified its control overhead. In our future work, we aim to generalize this model to a network with multiple source and multiple destinations.

REFERENCES

[1] V. Rodoplu and T. Meng, “Minimum energy mobile wireless networks,” *IEEE J. Select. Areas Commun.*, vol. 17, no. 8, pp. 1333 – 1344, Aug. 1999.

[2] ——, “Bits-per-joule capacity of energy-limited wireless networks,” *IEEE Transactions on Wireless Communications*, vol. 6, no. 3, pp. 857–865, 2007.

[3] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, “Directed diffusion for wireless sensor networking,” *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 2–16, 2003.

[4] N. Sadagopan, M. Singh, and B. Krishnamachari, “Decentralized utility-based sensor network design,” *Mobile Networks and Applications*, vol. 11, pp. 341–350, 2006.

[5] P. Nurmi, “Modeling energy constrained routing in selfish ad hoc networks,” in *Proceeding from the 2006 workshop on Game theory for communications and networks*, ser. GameNets ’06. ACM, 2006.

[6] H. Liu and B. Krishnamachari, “A price-based reliable routing game in wireless networks,” in *GameNets ’06: Proceeding from the 2006 workshop on Game theory for communications and networks*. ACM, 2006, p. 7.

[7] S. Sengupta, M. Chatterjee, and K. Kwiat, “A Game theoretic framework for power control in wireless sensor networks,” *IEEE Transactions on Computers*, vol. 59, no. 2, pp. 231–242, 2010.

[8] E. Campos-Nanez, A. Garcia, and C. Li, “A Game-theoretic approach to efficient power management in sensor networks,” *Operations Research-Baltimore*, vol. 56, no. 3, pp. 552–561, 2008.

[9] R. Kannan and S. Iyengar, “Game-theoretic models for reliable path-Length and energy-constrained routing with data aggregation in wireless sensor networks,” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 6, pp. 1141–1150, 2004.

[10] M. Felegyhazi, J. Hubaux, and L. Buttyan, “Cooperative packet forwarding in multi-domain sensor networks,” in *Third IEEE International Conference on Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops*, 2005, pp. 345–349.

[11] Y. Korilis, A. Lazar, and A. Orda, “Achieving network optima using Stackelberg routing strategies,” *IEEE/ACM Transactions on Networking (TON)*, vol. 5, no. 1, pp. 161–173, 1997.

[12] T. Başar and R. Srikant, “A Stackelberg network game with a large number of followers,” *Journal of Optimization Theory and Applications*, vol. 115, no. 3, pp. 479–490, 2002.

[13] H. Shen and T. Başar, “Differentiated Internet pricing using a hierarchical network game model,” *Proceedings of the 2004 American Control Conference*, vol. 3, pp. 2322–2327 vol.3, July 2004.

[14] S. Shakkottai and R. Srikant, “Economics of network pricing with multiple ISPs,” *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1233–1245, 2006.

[15] M. Bloem, T. Alpcan, and T. Başar, “A Stackelberg game for power control and channel allocation in cognitive radio networks,” in *Proc. IEEE ICST*, 2007, pp. 1–9.

[16] V. Rodoplu and G. Raj, “Computation of a nash equilibrium of multiple-leader stackelberg network games,” in *2010 Fifth International Conference on Systems and Networks Communications*, ICSNC 2010, pp. 232–237.

[17] “MICAz OEM edition data sheet,” Crossbow, San Jose, CA, USA.

[18] V. Rodoplu and T. Meng, “Core capacity region of energy-limited, delay-tolerant wireless networks,” *IEEE Transactions on Wireless Communications*, vol. 6, no. 5, pp. 1844–1853, 2007.