Minimum Latency Joint Scheduling and Routing in Wireless Sensor Networks

Gang Lu and Bhaskar Krishnamachari

Department of Electrical Engineering

University of Southern California, Los Angeles, CA 90089

{ganglu, bkrishna}@usc.edu

Abstract

Wireless sensor networks are expected to be used in a wide range of applications from environment monitoring to event detection. The key challenge is to provide energy efficient communication; however, latency remains an important concern for many applications that require fast response. In this paper, we address the important problem of minimizing average communication latency for the active flows while providing energy-efficiency in wireless sensor networks. As the flows in some wireless sensor network can be long-lived and predictable, it is possible to design schedules for sensor nodes so that nodes can wake up only when it is necessary and asleep during other times. Clearly, the routing layer decision is closely coupled to the wakeup/sleep schedule of the sensor nodes. We formulate a joint scheduling and routing problem with the objective of finding the schedules and routes for current active flows with minimum average latency. By constructing a novel delay graph, the problem can be solved optimally by employing the M node-disjoint paths algorithm under FDMA channel model. We further extend the algorithm to handle dynamic traffic changes and topology changes in wireless sensor networks. We also propose a heuristic solution for the minimum latency joint scheduling and routing problem under single channel interference. Numerical results show the latency can reduced 15% under stationary scenario and 50% under dynamic traffic or topology changes.

I. Introduction

A wireless sensor network (WSN) is a distributed sensing network comprised of thousands, or even tens of thousands small devices that sense, collect and disseminate information about the environment [1], [18], [21]. Wireless sensor networks require a new set of protocol stacks because of new features of wireless sensor networks. First, most nodes in sensor networks are likely to be battery powered and it is not feasible to recharge or replace the batteries. Second, sensor networks are in large scale with hundreds or even thousands nodes randomly deployed in an ad hoc fashion with little human management. Third, the traffic pattern in sensor networks varies with different sensor network applications. Major traffic could be in-network local communication or from sensors to a common sink in a tree topology. An example sensor data gathering application is shown in figure 1.

Measurements [2], [3] have shown that the best way to save energy is to put a node to sleep. However, a node that is off is unable to deliver the sensor data to the base stations. This creates a fundamental trade-off. However, as the traffic flows in sensor network are likely to be predictable long-lived, we could schedule the on-off activity of the sensor nodes to wake up to help deliver the sensor data reports in time and to go back to sleep to save energy otherwise.

Putting nodes to sleep affects another important communication layer: the network layer. A node in sleep is no longer part of the network. Therefore, by the sleep scheduling, the topology of the network keeps changing at different times. A link between two neighboring nodes is available only if both of them are scheduled to be active at the same time slot. The paths selected by the routing algorithm also affects latency and power consumption. Different links could have different latency depending on the scheduled sending and receiving slots of the nodes. Clearly, a shortest hop path may not be the path with shortest latency.

Thus, given certain source nodes and base stations in a sensor network, there are two key design considerations: one is the scheduling, the other is routing. Those two are closely coupled together and will affect each other. For example, in figure 1, node J needs to allocate time slots for data reports from G and I. However if node I changes its next hop to other node or it does not generate any more reports, then node J only needs to schedule slots for reports from G.

There are three possible approaches to setup schedules and routes for the flows in a WSN:

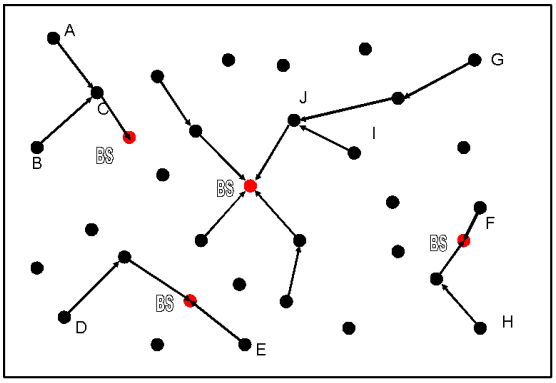


Fig. 1. A data gathering application in a wireless sensor network.

1) Scheduling first: Determine the schedules of the sensor nodes first. Based on the schedules, using a routing algorithm to find an energy efficient and low latency path.
2) Routing first: Using a routing algorithm to find a path first. Given the path, find the schedules of the nodes on the path.
3) Joint scheduling and routing: find the schedule and routing solution jointly.

Both scheduling first and routing first scheme have their disadvantages. Scheduling first schemes may cause routing scheme hard to find short paths while routing first schemes may cause low latency schedule impossible. There are many papers [6], [7], [8], [9], [10] working on energy-latency tradeoff wakeup schemes. Sichitiu [10] proposes a cross-layer scheduling for power efficiency in wireless sensor networks. In order to conserve energy, sensor nodes are turned off. However, since an in active sensor node is no longer part of the network, the network can become disconnected. The author proposes a deterministic, schedule-based energy conservation scheme, in which time-synchronized sensors form on-off schedules that enable the sensor to be awake only when necessary. The scheme can be decoupled into two distinct phases for each flow in the network: the setup and reconfiguration phase, and the steady state phase. In the setup and reconfiguration phase, first a route from the node originating the flow to the base station is selected, then the schedules are set up along the chosen route. If a schedule can not be set up along the chosen route, the routing protocol will find an alternative route. In this scheme, the scheduling and routing schemes work separately.

In our previous work of DMAC [6], we investigated an approach to delay-efficient sleep scheduling, designed specifically for wireless sensor networks where the communication pattern is restricted to an established unidirec-

tional data gathering tree. We showed that the sleep latency can be essentially eliminated by having a periodic receive-transmit-sleep cycle with level-by-level offset schedules. However, it is contention based approach and provides only best-effort service.

In DESS [7], we investigated the problem of minimizing the worst case communication latency for arbitrary all possible source-destination pairs while each node has a fixed duty cycle. In many sensor network application scenarios, typical communication patterns are from sensors sources to several limited number of base stations. All sinks are same, which means a packet can be routed to any one of the sink. Also at any specific time, not all sensor nodes have packets to sent to the base stations. So it is not necessary to minimize the worst case latency for arbitrary all-to-all communication pairs.

In this paper, we will discuss the joint scheduling and routing approach for current active flows in the network. Particularly, we are interested in finding paths and schedules that achieve the minimum average latency. It is different from DMAC in that it is schedule based and provide minimum latency. It is also different from DESS in that it only try to optimize average latency for active flows. We assume that energy efficiency will be provided automatically by the schedule, as nodes only wake up when needed and asleep during other times. This is clearly an issue of fundamental significance in the area of wireless sensor networks, and to our knowledge has never been investigated before.

II. Scheduling and Routing in Wireless Sensor Network

A. Application Scenario

Broadly speaking, there are two kinds of applications in sensor networks: event driven and continuous monitoring. In event driver sensor network, most of the time the sensor nodes are off until certain interesting event happens. Then the nodes begin to send data to base station periodically until the event disappear. In a continuous monitoring sensor networks, sensor nodes sample and transmit data at regular intervals requested by the base station. In both kinds of applications, at any specific time, certain source nodes sample the environment and report to base station periodically. In the following discussions, we only assume source nodes generating report periodically and do not distinguish these two kinds of applications. Our goal is to find paths from sources to sinks, and schedules of the nodes on the paths, with the objective to minimize the average latency.

We first describe the basic application assumptions.

1) **Sources and sinks:** Certain source nodes sample the environment periodically. The period can be different for different source nodes. There are several sink nodes in the network. A sensor report can be delivered and only need to be delivered to any one sink.
2) **Radio:** Consider a static wireless sensor network, all nodes are equipped with a single radio with half-duplex omni-directional antennas. The transmission power and data rate are fixed.
3) **Synchronization:** None of the discussion about TDMA link scheduling would be relevant if there were not some mechanism to provide time synchronization in the sensor network. However, techniques capable of providing micro-second level synchronization have been developed for sensor networks [4]. Also only localized synchronization is needed. The timing offset error between two nodes far away from each other is not important since they can not communicate directly anyway.
4) **TDMA frame and flow:** Time is divided into equal sized slots that are long enough for one packet transmission. Thus a packet may travel at most one hop in a single slot. Slots are grouped into frames. The length of the frame is set to the period of the slowest source which generate one packet per frame. Sources with higher traffic rate could use multiple slots in a frame to report multiple packets per frame. Each packet is regarded as a separate **flow**. Some works on TDMA focus on minimizing the length of the frame subject to the constraint that every node or link is assigned at least one slot. In this work, however, the latency is not affected by the frame length but the sleep latency [15], [6], [7] caused by the scheduling.
5) **Graph Abstraction:** While several recent papers in sensor networks (e.g. [11], [13], [12]) have shown that wireless links can be quite unreliable and vary significantly in packet reception rates in each direction, we have used a binary-link-based graph-theoretic problem formulation in this work. This is justified because the communication graph we are referring to is not necessarily the full wireless network, but a logical topology which can be constructed, for instance, by filtering or blacklisting out all unreliable/unidirectional links. Others have suggested that such blacklisting is necessary for reliable packet delivery in any case [13].
6) **Energy Conservation:** Sensor node radios incur differing energy costs in idle listening, receiving and transmission modes. Transmission costs are generally higher than idle/reception costs. Technically, the minimum

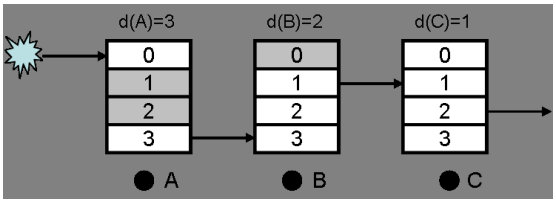


Fig. 2. Sleep latency caused by putting node to sleep.

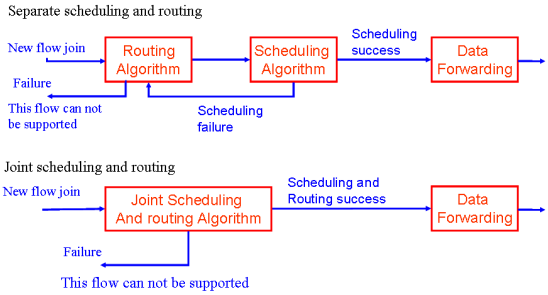


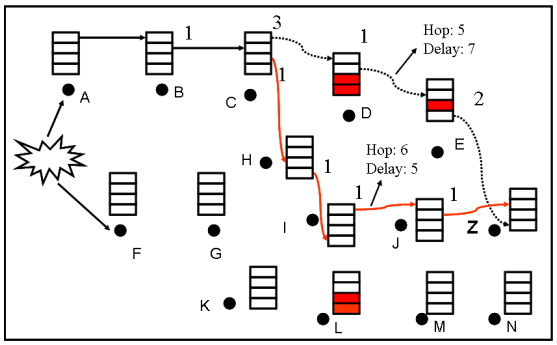
Fig. 3. Two scheduling and routing schemes in wireless sensor networks.

delay path obtained may involve longer hops (more transmissions) than the minimum hop-count path on the original graph. Thus delay minimization can result in a slight increase in the energy costs, however we believe this is a second order effect since the bulk of the energy savings in the network are provided by the sleep mode of the radio.

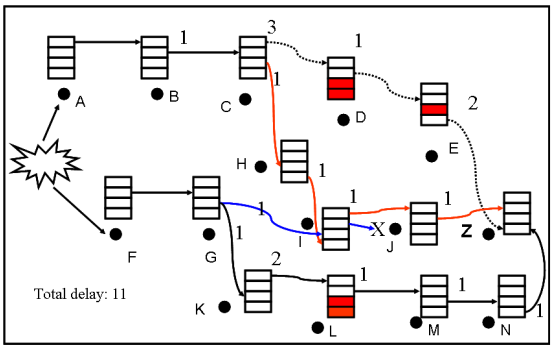
7) **Channel Model:** Due to space constraint, we will study the FDMA-based multiple channels model in this paper. We assume that the number of channels is large enough so that any two links within interference range are assigned two different FDMA channels. Interested reader could refer to [23] for single channel model with interference among nearby links.

B. MLSR Problem Definition

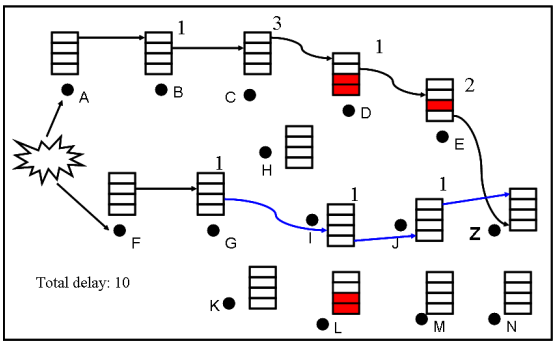
We use K as a parameter that indicates the number of slots in a TDMA frame. The K is big enough that all periodically report will have at least one report every K slots. Sources can have multiple reports in one frame. In each slot, the radio is in one of the three states: transmitting, receiving or free. Radio should be put to sleep in free state to save energy. If a node has to forward a packet to its neighbor, it need to find a slot that both the neighbor and itself are free. Then the slot in the sender is marked as transmitting and marked as receiving in the receiver. This conserves the energy of both the transmitting and the receiving node. We do not consider slots reserved for routing setup and neighbor discovering.



(a) Scheduling and routing for 1st flow



(b) One possible scheduling and routing for two flows



(c) A better scheduling and routing for two flows

Fig. 4. Example of scheduling and routing for two active flows.

Let $G = (V, E)$ be an arbitrary graph. Let P_m be the path for flow m from a source to a sink. Let f denotes the slot assignment. For a given node i, let R_m^i and S_m^i denote the receiving and transmitting slots of node i for flow m. Clearly, R_m^i and S_m^i determine the delay incurred in transmitting data from one node to the other. Let $d_m(i, j)$ be the delay in transmitting data from i to j where $(i, j) \in E$ [1]:

$$d_m(i,j) = \begin{cases} S_m^i - R_m^j \quad (\text{if } S_m^i - R_m^j > 0) \\ (S_m^i - R_m^j) + K (\text{otherwise}) \end{cases} \tag{1}$$

Delay on a path P_m under the slot assignment is defined as

$$d(P_m) = \sum_{(i,j) \in P_m} d_m(i,j) \tag{2}$$

As seen from the above discussion, the end-to-end delay for flow m depends on both the path and the slot assignment.

Figure 2 shows an example of the sleep latency. Suppose the TDMA frame size is $K = 4$. Thus each node have

[1] $S_m^i = R_m^j$

4 slots for Tx/Rx. Gray slot means the slot is not available for use (used by other flows already). Source node A generates one packet per frame at slot 0. A can only forward the packet to node B at slot 3. Thus there is a sleep latency of 3 at node A. Similarly, the packet is delayed 2 slots at node B and 1 slot and node C.

Active Flows Communication: For a sensor network application, at any specific time, not all nodes have sampling data to report to the base station: there are only a limited number of active flows in the network. Here, it would be of interest to minimize the average delay of the active flows in the network, which is defined as follows:

***Definition** 1:* **Average Delay** ($D_{P,f}^{avg}$): For a given graph $G = (V, E)$, number of slots k, a slot assignment function f and paths P for M flows, the *average delay* is defined as $\sum_{m \in M} d(P_m)$, where $d(P_m))$ is the delay along the path P_m under the given slot assignment function f.

In *active flows communication*, our design goal is the following:

***Definition** 2:* **Minimum Latency joint Scheduling and Routing (MLSR)** Given a graph $G = (V, E)$, the number of slots K and M flows, find paths P and slots assignment f to support maximum number of flows with minimum *average delay* (D_f^{avg}) i.e.

$$(P, f) \quad = \quad \arg\min_{P', f'} \{D_{P', f'}^{avg}\} \tag{3}$$

Figure 3 shows an example of separate routing and scheduling solution. When there is a new flow request, the routing algorithm will find a route first, then on the given route, if a schedule is achievable, the following data forwarding will use the route and scheduling to forward data report. If the scheduling process fails, then the routing algorithm is asked to find a new route.

Figure 4 shows an example of scheduling and routing for two active flows. In figure 4(a), first there is only one active flow. If we run a shortest hop routing algorithm, it will find the path of $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow Z$. A schedule can be assigned on the nodes on the path. The number of hops is 5 and the delay is 7. If using the link delay as the weight of the link, however, shortest path routing algorithm now select $A \rightarrow B \rightarrow C \rightarrow H \rightarrow I \rightarrow J \rightarrow Z$ which has a latency of only 5 even though the number of hops is increased to 6. Now assume there is another flow request from node F to node Z. Suppose the flow want to use route $F \rightarrow G \rightarrow I \rightarrow J \rightarrow Z$. However at link (i, j), there is no slot that both i and j are free, so the flow is forced to choose route $F \rightarrow G \rightarrow K \rightarrow L \rightarrow M \rightarrow N \rightarrow Z$ which has a delay of 6. So the total delay for the two active flows is 11 in figure 4(b). However, as shown in figure

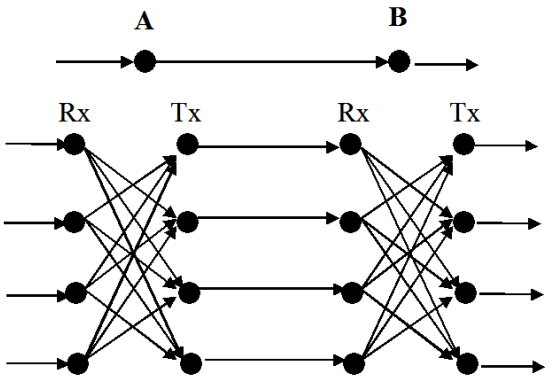


Fig. 5. Delay Graph: Link

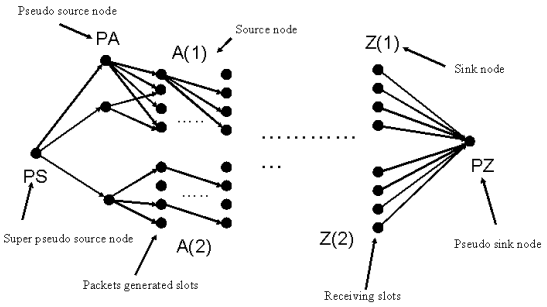


Fig. 6. Delay Graph: Network

4(c), if source A use route $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow Z$ and source F use route $F \rightarrow G \rightarrow I \rightarrow J \rightarrow Z$. A schedule can be achieved with total delay of only 10. This example shows that latency reduction by joint scheduling and routing.

Intuitively, in MLSR, the objective is to color a graph with the given K colors such that the desired global objective, minimizing the *average delay diameter*, is achieved. The reader may perceive a connection to the well-known NP-complete graph coloring problem, which deals with minimizing the number of colors needed to ensure that no two adjacent vertices are colored the same. However, a key difference between the graph coloring problem and MLSR is that the former is essentially about a local constraint (adjacent vertices requiring distinct colors), while the latter is inherently more global in nature: adjacent vertices may share the same slot assignment but the average delay paths for the active flows must be minimized. While TDMA is NP-hard, we present a polynomial optimal solution for MLSR.

III. Minimum Latency Joint Scheduling and Routing

In this section, we present the optimal solution of minimum latency joint scheduling and routing given static network topology and traffic flows. First we will introduce the novel concept of *Delay Graph*. Then we will prove that the minimum weight M node-disjoint paths on the *Delay Graph* can be mapped to the minimum latency joint scheduling and routing solution.

A. Delay Graph

Suppose there are M source nodes and N sinks. The number of slots per TDMA frame is K. We create a *Delay Graph* as following:

1) For each sensor node u, create $2K$ graph nodes. The first K nodes represent the receiving slots of sensor node, R_u^i, which means node u receiving a packet at slot i from the previous hop. The second K nodes represent the transmitting slots of the sensor node, S_u^i, which means node u transmitting the packet at slot i to the next hop.
2) From the each receiving node of u, R_u^i, add a link to all transmitting nodes of u, S_u^j except $j = i$. The delay of the link is decided by equation 1.
3) If two sensor nodes u and v can communicate with each other, create a graph link from each transmitting node of u, S_u^i to the corresponding receiving node of v, R_u^i, which means node u transmits a packet to v at slot i. Assign the weight of the link 1.
4) Add a super sink graph node PZ. Add a link from each receiving nodes $R_{Z(n)}^i$ of sink $Z(n)$ to PZ. Assign the weight of the link 1.
5) Add a graph node $PA(m)$ for each source sensor node $A(m)$, which is called pseudo source node. Add a graph link from $PA(m)$ to each receiving node $R_{A(m)}^i$ of $A(m)$, with link delay weight of 1. If a source have more than one packet to send per TDMA frame, treat each packet as a different source.
6) Add a super pseudo source graph node PS. Add a directed link from PS to each $PA(m)$ with link delay weight of 1.

Figure 5 shows how to split a node to $2K$ nodes in the delay graph and the connection between two nodes. Figure 6 shows an example of a complete delay graph with two sources while source $A(1)$ have two packets per frame and the second packet can be generated only at slot 2 and 4.

B. Minimum weight M node-disjoint paths

Before we present our solution to MLSR problem, we first give some background on the minimum weight M node-disjoint paths problem which can be defined as follows: Given a weight graph $G = (V, E)$ with weight on each link W_{uv} and a source-destination pair (s, z), find M node-disjoint paths $(P_1, P_2, ..., P_M)$ from s to z, such that the total weight of the paths $\sum_i W(P_i)$ is minimized. Node-disjoint paths means that a node can be appeared no more than once on the paths.

References [16], [17] proposed algorithms to find Minimum weight (here means delay) M node-disjoint paths. We will employ this algorithm to find M node-disjoint paths on the delay graph. Here we briefly explain the steps of the algorithm on the delay graph. Any shortest path algorithm can be used to find the first minimum weight node disjoint path from s to z, where s is the super pseudo source node, and z is the pseudo sink node. We assume P_M is a given optimal set of M node-disjoint paths in delay graph DG. Now we need to find optimal $(M+1)th$ node-disjoint paths P_{M+1}, as follows:

1) Reverse the direction of each edge on P_M, and make its length negative. These edges are called negative arcs. Other edges are called positive arcs.
2) Split each vertex v on P_M into two nodes v_1 and v_2, joined by an arc of length zero, directed towards a. Assign output links on v_2 and input links on v_1.
3) Find a shortest path from source node s to sink node z on this transformed delay graph. We call this path an interlacing S, which may contain both positive arcs and negative arcs.
4) let $P_M + S$ represent the graph obtained by adding to P_M the positive arcs of S, and removing from P_M the negative arcs of S. This is called Augmentation, which results in the optimal P_{M+1} paths, the minimum weight $M + 1$ node-disjoint paths.

Figure 7 shows a simple example. Interested reader could refer [16], [17] for details.

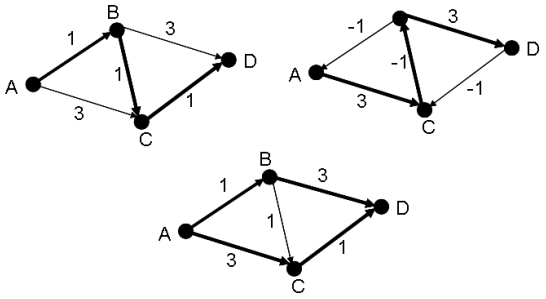


Fig. 7. An example of finding minimum length 2 node disjoint paths.

C. MLSR Algorithm

In this section, we present the minimum latency joint scheduling and routing algorithm by employing the minimum weight M node-disjoint paths algorithm. First we discuss the constraints of the radios in FDMA-based channel model:

1) With a single omni-directional antenna, no node can transmit or receive data at the same time.
2) A node cannot transmit different packets to different receivers at the same time.
3) A node can only receive a packet from a single sender at one time.

We also assume no node will broadcast a single packet to multiple receivers. Therefore each sender can only have one receiver.

Given the constraints, we prove that the M node-disjoint paths on the delay graph can be mapped to the minimum latency joint scheduling and routing solution for M sources satisfying these constraints.

Proposition 1: The minimum weight M node-disjoint paths in the delay graph can be mapped to a minimum latency joint scheduling and routing solution.

Proof: First we explain how to get a route and schedule from the node-disjoint disjoint path. Suppose one of the M node-disjoint path is P_m. Assume the nodes the path goes through are: $PS, PA(m), R^{i1}_{A(m)}, S^{j1}_{A(m)}, ..., R^{i_u}_u, S^{j_u}_u, R^{i_v}_v, S^{j_v}_v, ...,$
It is easy to know that $j_u == i_v$. The route for the flow m then is $A(m), ..., u, v, ...Z(n)$. The schedule of node u on the route is to wake up to receive packet at i_u and send it at j_u. At j_u, node v is wake up to receive packet and v will forward it to the next hop at time j_v.

Then we need to prove that the schedule we get satisfy the three constraints of the radio. It is clear that no node can transmit and receive at the same time since there is no link in the delay graph from a node's receiving node to

its transmitting node that has the same slot. A node will not be scheduled to transmit different packets to different receivers at the same time, otherwise it will violate the node-disjoint rule. A node can also receive a packet from a single sender at the one time because of the node-disjoint rule.

Finally, the sum of the weights of the M node-disjoint paths can be interpreted as the total delay of M paths directly. Thus the joint scheduling and routing solution is optimal in terms of latency. ■

Given M flows, if we can iteratively find M node-disjoint paths, then we get the minimum latency joint scheduling and routing solution for M flows. If we can only find K node-disjoint paths ($K < M$), then only K flows can be supported and these K flows will have the minimum average latency.

D. Distributed implementation

The algorithm can be easily implemented in a distributed manner. The only operation needed in the algorithm is to find the shortest path between source and sink, with possible negative weights (no negative cycles). Distributed versions of Bellman-Ford algorithm can be employed directly [19].

Bellman-Ford algorithm has been used widely in Internet, known as the Distance Vector routing algorithm. In the decentralized implementation in Internet, each node periodically broadcasts its routing tables, which contains the cost from itself to all other nodes, to all its neighbors. A router updates its own routing table according to the routing tables of its neighbors. The algorithm will converge to the optimal solution after certain number of iterations. Each node then knows the next hop to forward a packet by its routing table. AODV [20], Ad hoc On demand Distance Vector, is an on-demand routing protocol for wireless multi-hop networks, that is able to find the shortest path between two nodes. One thing need to be mentioned is that the shortest path is in the delay graph, not the original network. Each real node will present $2K$ nodes in the delay graph and the $(k-1)^2$ links among them. In order to support broadcast, one awake slot is reserved every M slots for all nodes for local updates or route message forwarding. The larger M, the longer route setup latency but more energy efficient. The smaller M, the shorter route setup latency but more energy wastage. Each real node broadcasts to its neighbor nodes if its slot usage has any change. We will not discuss the detail of the distribution algorithm here which is one of our future work. Generally, the distributed algorithm has a communication complexity of $O(V)$ since each node forward the packets at most once. It has a time complexity of $M * d$, where d is the number of hops between

source and destination. Assume that traffic flows are long lived and the frequency of route setup process is rare, the overhead, e.g. communication message exchanges and latency of route setup, of the distributed routing algorithm can be ignored. In the performance evaluation section, we will use the centralized version of the algorithm.

IV. Dynamic Minimum Latency Joint Scheduling and Routing

In previous section, we present an algorithm to find M joint scheduling and routing solutions for M flows assuming static traffic flows and topology. However, in a real sensor network, flows can appear or disappear dynamically and randomly. There are also frequent link disconnections due to node failures or strong interference from the harsh environment. Thus we will study MLSR under dynamical traffic and topology changes in this section.

A. MLSR under Traffic change

The algorithm in reference [16] is aimed to find M node-disjoint path from the beginning. In a real sensor network application, however, flows can appear or disappear dynamically and randomly. The overhead to re-run the whole algorithm whenever there is a change of existing flows is high and also hard to implement. We propose an extension of the algorithm that need only incremental updates on the newly added flow or removed flow.

1) Adding a Flow: Given a Delay Graph and M existing flows, suppose we already have minimum latency M node-disjoint paths. For each source added, we add a pseudo source node in the Delay Graph and connect the super pseudo source node to it. Then we try to construct an interlacing on the new Delay Graph and interpreted it to construct $M + 1$ node-disjoint paths. The paths found maybe different from the paths found by rerun the whole algorithm on the Delay Graph from beginning, but the latencies are same.

In general, the new source could be a new source node or an additional report slot request from an existing source node. Without loss of generality, suppose there is a new source node A that needs to report to sinks.

AddFlow Algorithm:

1) Add a pseudo source node PA for source node A. Create links from PA to each receiving node or packet generated node of A, R_A^i with link delay 1. Also create a link from super pseudo source node PS to PA.
2) Assign the link (PS, PA) a very large weight β which assure that any path use this link will have longer delay than the path without using it.
3) Find a minimum length interlacing S, from PS to PZ.
4) By $P_M + S$, we get P_{M+1} minimum latency paths.

We will prove that the new schedules and routes calculated by AddFlow algorithm for the $M+1$ flows are still optimal in terms of latency. First we present a LEMMA that will be used in the proof.

***Lemma** 1:* The output of the augmentation of $P_M + S$ is a set of minimum weight $M+1$ nod-disjoint paths: $P_{M+1} = P_M + S$.

We omit proofs here. Interested readers could see [16], [17] for details.

Proposition 2: AddFlow algorithm reserves optimality.

Proof: We will prove it by induction. Suppose we have M optimal paths P_M. Now a new source node S_i start to generate packets. We add a pseudo source node PS_i and the necessary links as described in the algorithm. We call this delay graph G. Then we assign a very large weight β to the link of (PS, PS_i). We call the new graph G'. As β is large enough that the path using this link always has length larger than any other path. Thus P_M remains optimal in G'. By Lemma 1, P_{M+1} is optimal in G'. Now we want to prove that P_{M+1} is also optimal in G.

First the links from PS to all other pseudo source nodes PS_i must be in P_{M+1} as those nodes are the only $M+1$ out neighbors of PS. Second, let us consider only the paths from all PS_i to pseudo sink node PZ. These paths are optimal in G' and remain optimal in G. Therefore the $M+1$ optimal paths in G' are still optimal in G.

This completes the proof. ■

2) Removing a Flow: Removing a flow from the M flows is different. We will construct a shortest length interlacing that originated from the pseudo sink node to the super pseudo source node. We remove all links from super pseudo source node to the pseudo source nodes, except the one that we want to remove. After interpret the interlacing with the existing k node-disjoint paths, only $K-1$ paths remains, which is a minimum latency $k-1$

node-disjoint paths. Suppose we need to remove the flow originated from source node i; its pseudo source node in delay graph is s_i.

Before we present the algorithm to remove a flow, we extend the result on minimum weight M node-disjoint paths. The authors [16], [17] only proposed algorithm to compute P_{M+1} from P_M. This can be extended to compute $M-1$ node-disjoint paths from M node-disjoint path as follows:

1) Reverse the direction of each edge on P_M, and make its length negative. These edges are called negative arcs. Other edges are called positive arcs.
2) Split each vertex v on P_M into two nodes v_1 and v_2, joined by an arc of length zero, directed towards a. Assign output links on v_2 and input links on v_1.
3) Find a shortest path from sink node z to source node s on this transformed delay graph. We call this path an interlacing rS, which may contain both positive arcs and negative arcs.
4) let $P_M + rS$ represent the graph obtained by adding to P_M the positive arcs of S, and removing from P_M the negative arcs of S. This is called Augmentation, which results in the optimal P_{M-1} paths, the minimum weight $M-1$ node-disjoint paths.

***Lemma** 2:* The output of the augmentation of $P_M + rS$ is a set of minimum weight $M-1$ node-disjoint paths: $P_{M-1} = P_M + rS$.

Proof: Suppose we compute P_M from P_{M-1} by augmentation of an interlacing S. Suppose the original graph is G. According to [16], in an equivalent graph G' of G, P_{M-1} contains all negative links of G' and all links on S have 0 length. In the link reversed graph of G', G'_M, the exact reverse of S, rS have the smallest weight of 0. By augmentation of P_M and rS, we will get P_{M-1}. It is possible there are other reverse interlacing with length 0, but the result P_{M-1} has the same total length. Thus the augmentation of $P_M + rS$ is a set of minimum weight $M-1$ node-disjoint paths. ■

Now we present the RemoveFlow Algorithm.

RemoveFlow Algorithm:

1) Assign the link from PS to PS_i a very large weight β such that any path using (PS, PS_i) is larger than any other paths through other pseudo source nodes.

2) Reverse the direction of each edge on P_M, and make its length negative. These edges are called negative arcs. Other edges are called positive arcs.
3) Split each vertex v on P_M into two nodes v_1 and v_2, joined by an arc of length zero, directed towards PS. Assign output links on v_2 and input links on v_1.
4) Find a shortest path from PZ to PS on this transformed delay graph, which is a reverse interlacing rS.
5) let $P_M + rS$ represent the graph obtained by adding to P_M the positive arcs of rS, and removing from P_M the negative arcs of rS. This is called Augmentation, which results in the optimal P_{M-1} paths, the minimum weight $M-1$ node-disjoint paths.

Proposition 3: The output of the augmentation of $P_M + rS$ is a set of minimum weight $M-1$ nod-disjoint paths without source node i.

Proof: Similarly we can prove the algorithm by induction. As in the algorithm, after we assign a very large weight β on link $(PS, PS_i$, we get a new delay graph G'. Then we remove the link of (PS, PS_i), we get a new delay graph G. Our objective is to get minimum latency $M-1$ node-disjoint paths on G.

It is clear that the output of RemoveFlow is a set of $M-1$ node-disjoint path in G'. Since rS is a minimum reverse path from PZ to PS, by Lemma 2, $P_M + rS$ is a set of minimum weight $M-1$ node-disjoint paths in G'. As β is large enough, the minimum reverse path rS from PZ to PS must include the link (PS_i, PS) which has a weight of $-\beta$ (note there are no negative cycles). Thus the link (PS_i, PS) will be removed which means flow i be removed from the paths. Now let us consider only the paths from all other pseudo source node to pseudo sink node PZ. These paths are optimal in G' and also optimal in G. Therefore the $M-1$ optimal paths in G' are also optimal in G. ■

B. MLSR under Topology change

The basic algorithm in the previous section assumes that the network topology is static while the k disjoint paths are discovered. In this section, we generalize this approach to a network that is undergoing constant topological changes. For instance, a mobile node can simply walk away from the communication network or a link is under strong interference in a harsh environment. We model such changes by link failures. Note that node failures can be modeled as a special case of a certain set of link failures. That is, if node v were to fail, this event can be

modeled as the failure of all links adjacent to v. Besides link failures, new sensor nodes may be put into the field when there are not enough old nodes available for the application. Or an environmental noise disappeared so a link between two nodes is now available. We model such changes by link joins.

When link failures or link joins happen, the topology of the network changed. If a failed link is currently used by the one of the k minimum latency disjoint paths, the flow using this path is unable to transmit the report. If a new link joins, the existing k disjoint paths may no longer have the minimum latency. Under both situations, a new set of k minimum latency disjoint paths need to be computed. One way is to recompute the k disjoint paths from beginning on the new topology. However, the overhead is high if topology changes happen frequently. Same as the flow changes, we propose algorithms which can adjust the existing k disjoint paths incrementally with only two computation of the shortest path algorithm.

When a link $e = (u, v)$ currently used by one of the P_M path failed, a new set of P_M is needed. Assume e is used by source s_i and the path is P_i. The idea is to first remove the flow 1 to get minimum latency $M - 1$ node disjoint path P_{M-1} without using the failed link, then construct a new set of P_M paths. The steps of the algorithm are following:

LinkFailure Algorithm:

1) Reverse the direction of each edge on P_M, and make its length negative. These edges are called negative arcs. Other edges are called positive arcs. Failed link $e = (u, v)$ now becomes $e' = (v, u)$.
2) Split each vertex v on P_M into two nodes v_1 and v_2, joined by an arc of length zero, directed towards PS. Assign output links on v_2 and input links on v_1.
3) Find a shortest path from PZ to v on this transformed delay graph. This is a reverse interlacing rS_1.
4) Find a shortest path from u to PS on this transformed delay graph. This is a reverse interlacing rS_2.
5) let $P_M + rS_1 + rS_2 + (v, u)$ represent the graph obtained by adding to P_M the positive arcs of rS, and removing from P_M the negative arcs of rS. This is called Augmentation, which results in the optimal P_{M-1} paths, the minimum weight $M - 1$ node-disjoint paths with link $e = (u, v)$ removed.
6) Using algorithm described in section "adding a flow" to construct a new set of P_M minimum latency node-disjoint path.

For LinkJoin Algorithm, suppose the newly joint link is $e = (u, v)$.

LinkJoin Algorithm:

1) Reverse the direction of each edge on P_M, and make its length negative. These edges are called negative arcs. Other edges are called positive arcs. The newly added link is (u, v).
2) Split each vertex v on P_M into two nodes v_1 and v_2, joined by an arc of length zero, directed towards PS. Assign output links on v_2 and input links on v_1.
3) Find a shortest path from PZ to u on this transformed delay graph. This is a reverse interlacing rS_1.
4) Find a shortest path from v to PS on this transformed delay graph. This is a reverse interlacing rS_2.
5) Let $P_M + S_1 + S_2 + (u, v)$ represent the graph obtained by adding to P_M the positive arcs of S, and removing from P_M the negative arcs of S. This is called Augmentation, which results in the optimal $P^{'}_{M-1}$ paths with (u, v), the minimum weight $M + 1$ node-disjoint paths with link $e = (u, v)$ added.
6) Compute $P^{''}_{M-1}$ by RemovFlow algorithm without link (u, v).
7) Use the smaller of $P^{''}_{M-1}$ or $P^{'}_{M-1}$ as P_{M-1} to compute P_M.

V. Numerical Results

In this section, we evaluate the performance of the MLSR algorithm and heuristic algorithms through high level simulations. Since the current study focuses on comparing the total latency only across these heuristics, even the distributed algorithms are simulated in a centralized manner (without analyzing their overhead).

A sensor network is generated by randomly scattering 200 nodes on a 200×100 rectangle. There are four sinks in the corners of the area. The radio range is set to be 10 meters. We use the topology generation tool provided by [14] to get the packet reception ratio (PRR) of links between two nodes. Links with PRR larger than 0.9 are added into the abstract communication graph.

A. Heuristic solutions

First we propose a heuristic solution based on Dijkstra algorithm for static network.

Dijkstra-based-heuristic: This algorithm is a very basic algorithm that finds the link disjoint paths. It entails running Dijkstra's shortest path algorithm M times on the Delay Graph G, where after each run, links belongs to

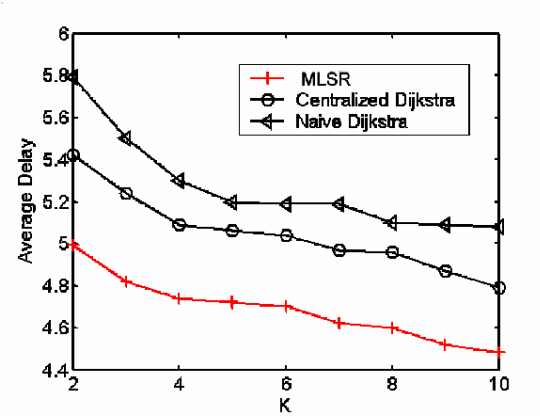


Fig. 8. Average delay under different frame length

the last path found are removed from G, ensuring link-disjointness among the M paths.

Then we propose the following two simple heuristic solutions under traffic and topology changes:

1) **Naive Dijkstra algorithm:** When a flow joins, a new Dijkstra's shortest path is found for the new flow with the existing paths and schedules unchanged. When a flow ends, links belonging to the path of the flow are added back to the Delay Graph G, paths and schedules of other flows remain unchanged. When a link is broken, if it is used by one of the path, other links on the path are added back to the Delay Graph and then a new path is computed for that source using the old path. Nothing is done if a new link is added. We refer this algorithm as $NaiveD$. Actually, Dijkstra-based-heuristic can be seen as a special case of $NaiveD$ when network is initialized and M flows join one by one.
2) **Centralized Dijkstra algorithm:** Whenever there is a flow or link change, a completely new Delay Graph G is constructed and paths and schedules for all flows are recomputed from scratch by Dijkstra-based-heuristic. We refer this algorithm as $CentraD$.

B. Results

Figure 8 shows the average delay under different frame length for MLSR, $NaiveD$ and $CentraD$ algorithms. For each frame length k, the number of flows are $4K$ which is the largest possible number of flows that can be supported by the 4 sinks. The results are averaged over 10 different seeds. Clearly MLSR has the smallest average latency while $NaiveD$ has the largest latency. Compared to $NaiveD$ algorithm, the delay of MLSR is reduced to around 15%. As K increases, the number of flows increases but the rate of each flow decreases, so the total traffic

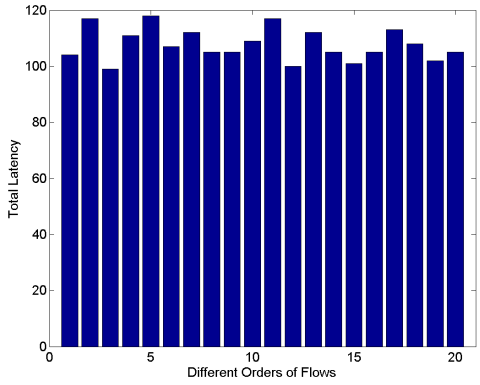


Fig. 9. Total Delay of $NaiveD$ heuristic under different flow join order

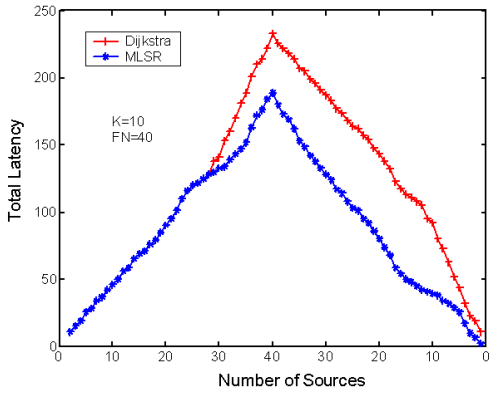


Fig. 10. Total delay of $NaiveD$ heuristic and MLSR for flow joins and leaves

load is unchanged. But because of distribution reason, more flows are closed to the sink, thus the latency decreases.

Figure 9 shows the $NaiveD$ algorithm under different flow join order. In this scenario, $K = 5$, the number of flows $FN = 20$. The flows are fixed but the order of being processed by the $NaiveD$ is different. Clearly the order will affect the performance of the algorithm. However, $CentraD$ algorithm is likely to achieve lower latency while MLSR achieves the minimum latency regardless of the order.

Figure 10 shows the performance of MLSR and $NaiveD$ under traffic changes: flows comes in order and then later leaves. The $CentraD$ algorithm is not simulated here as it can not handle traffic change adaptively. For the first 27 flows, the $NaiveD$ algorithm achieves the same latency as MLSR. This shows that when the number of flows is small, there is no need to do the complicated MLSR algorithm. However when the number of flows is larger than 27, the MLSR has smaller latency. Later when flows finished their data transmission in the same order as they joins (the first joined flow first left), the MLSR always achieves the minimum latency for current active

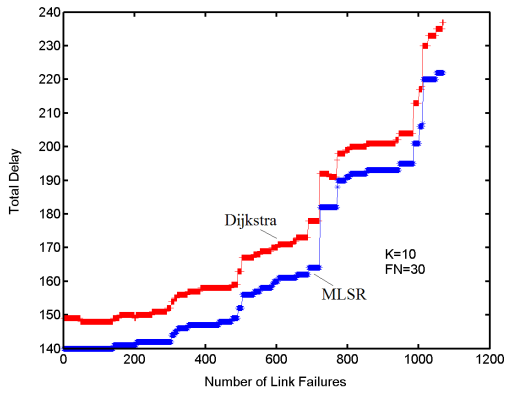


Fig. 11. Total delay of $NaiveD$ heuristic and MLSR under topology change

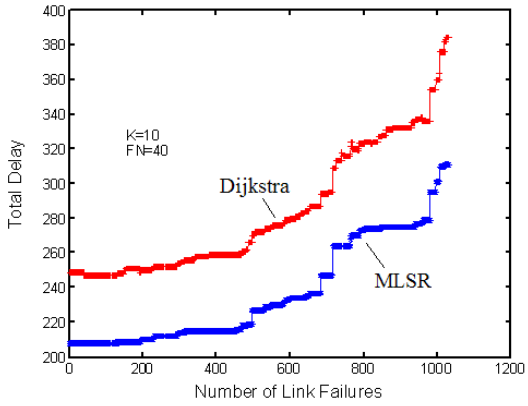


Fig. 12. Total delay of $NaiveD$ heuristic and MLSR under topology change

flows. The naive algorithm, however, performs poorly even when there are less then 27 active flows, it has higher latency than MLSR.

Figure 11 and 12 show the performance of MLSR and $NaiveD$ under topology change with $k = 10$. We keep removing links and computes the latency after each link failure until we can not find 30 or 40 node-disjoint paths. As link failure happens, the latency increases. The MLSR achieves smaller average latency under various topology changes than $NaiveD$.

Another important objective of routing in sensor network is load balancing which is to distributed the traffic load evenly to the sensor nodes in the network. This would help maximize the lifetime of the network. Figure 13 shows the average busy slots (for either receiving or transmitting) for nodes in the routes to forward the traffic packets. Clearly, MLSR achieves the lowest average traffic load which means better performance in load balancing. The reason is simple since in order to reduce latency, MLSR will likely choose nodes with more free slots.

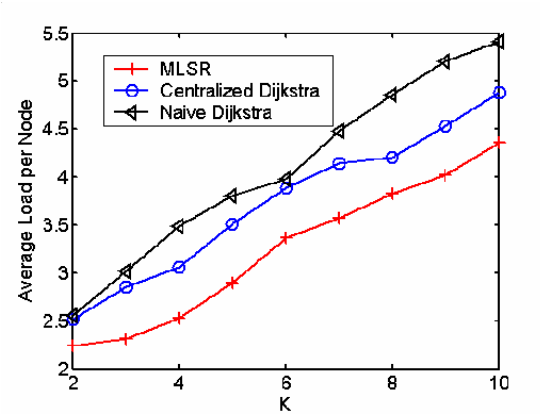


Fig. 13. Average slots occupied per node for nodes in the routes.

VI. Conclusion

In this paper we addressed the important problem of minimizing communication latency while providing energy-efficiency for nodes in wireless sensor networks. Different from DESS whose objective is to minimize the worst case latency given the duty cycling requirement that each sensor has to be awake for $\frac{1}{k}$ fraction of time slots on an average, MLSR is interested in the average latency for the current active flows. A node is allowed to wake up multiple slots to receive/transmit and asleep otherwise. We formulated a joint scheduling and routing problem with objective to find the schedule and route for current active flows with minimum average latency. By constructing a delay graph, the problem can be solved optimally by M node-disjoint paths algorithm under FDMA channel model. We further extended the algorithm to handle dynamic traffic changes and topology changes in wireless sensor networks. We also proposed a heuristic solution for the minimum latency joint scheduling and routing problem under single channel interference. Numerical results show the latency can be reduced 15% under stationary scenario and 50% under dynamic traffic or topology changes.

References

[1] D. Estrin , J. Heidemann , R. Govindan and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks" , *ACM MobiCom*, August 1999

[2] L. M. Feeney, M. Nilsson, "Investigating the Energy Consumption of Wireless Network Interface in an Ad Hoc Networking Environment", *IEEE INFOCOM* 2001

[3] M. Stemm and R. H. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices,", *IEICE Transactions on Communications*, vol. E80-B, no.8, pp.1125-1131, Aug. 1997

[4] J. Elson, L. Girod and D. Estrin, "Find-Grained Network Time Synchronization using Reference Broadcasts", *ACM SIGOPS* 2002

[5] W. Liang, "Constructing Minimum Energy Broadcast Trees in Wireless Ad Hoc Networks", *MOBIHOC* 2002.

[6] G. Lu, B. Krishnamachari, C. S. Raghavendra, "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Sensor Networks", *4th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks WMAN*, April 2004.

[7] G. Lu, N. Sadagopan and B. Krishnamachari, Ashish Goel, "Delay Efficient Sleep Scheduling in Wireless Sensor Networks", *IEEE INFOCOM*, Miami, FL, March 2005

[8] X. Yang and N. Vaidya, "A Wakeup Scheme for Sensor Networks: Achieving Balance between Energy Saving and End-to-end Delay", *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)*, 2004

[9] M. J. Miller and N. H. Vaidya,"Minimizing Energy Consumption in Sensor Networks Using A Wakeup Radio", *IEEE WCNC*, 2005

[10] M. L. Sichitiu, "Cross-Layer Scheduling for Power Efficiency in Wireless Sensor Networks", *IEEE Infocom* 2004

[11] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin and S. Wicker, "Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks", UCLA CS Technical Report UCLA/CSD-TR 02-0013, 2002.

[12] J. Zhao and R. Govindan, "Understanding Packet Delivery Performance in Dense Wireless Sensor Networks", *ACM Sensys*, November 2003.

[13] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Issues for Reliable Multhop Routing in Sensor Networks", *ACM SenSys*, November 2003.

[14] M. Zuniga, B. Krishnamachari, "Analyzing the Transitional Region in Low Power Wireless Links", *IEEE SECON*, October 2004

[15] W. Ye, J. Heidemann, and D. Estrin, "Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks", *IEEE/ACM Transaction on Networking*, To Appear.

[16] J. W. Suurballe, "A Quick Method for Finding Shortest Paris of Disjoint Paths", *Networks*, 14(1974) pp. 125-145

[17] R. Bhandari, "Optimal physical diversity algorithms and survivable networks", *Proc. Second IEEE Symposium on Computers and Communications* 1997

[18] R. Min, M. Bhardwaj, S. Cho, A. Sinha, E. Shih, A. Wang, and A. P. Chandrakasan, "Low-power wireless sensor networks", *14th International Conference on VLSI Design* 2001

[19] D. Bertsekas, R. Gallager, Data Networks, Prentice-Hall

[20] C. E. Perkins and E. M. Royer, "Ad-hoc On Demand Distance Vector Routing", *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, February 1999

[21] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors", *Communications of the ACM*, 43(5):551-558, May 2000

[22] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "Habitat Monitoring: Application Driver for Wireless Communications Technology", *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001

[23] G. Lu and B. Krishnamachari, "Minimum Latency Joint Scheduling and Routing in Wireless Sensor Networks", *Technical Report*, USC, Department of Electrical Engineering, 2006

[24] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, “Wireless Sensor Networks for Habitat Monitoring”, *IEEE WSNA* 2002