Energy-Efficient Multihop Polling in Clusters of Two-Layered Heterogeneous Sensor Networks

Zhenghao Zhang, *Member*, *IEEE*, Ming Ma, *Student Member*, *IEEE*, and Yuanyuan Yang, *Senior Member*, *IEEE*

Abstract—In this paper, we study two-layered heterogeneous sensor networks where two types of nodes are deployed: the basic sensor nodes and the cluster head nodes. The basic sensor nodes are simple and have limited power supplies, whereas the cluster head nodes are much more powerful and have many more power supplies, which organize sensors around them into clusters. Such two-layered heterogeneous sensor networks have better scalability and lower overall cost than homogeneous sensor networks. We propose using polling to collect data from sensors to the cluster head since polling can prolong network life by avoiding collisions and reducing the idle listening time of sensors. We focus on finding energy-efficient and collision-free polling schedules in a multihop cluster. To reduce energy consumption in idle listening, a schedule is optimal if it uses the minimum time. We show that the problem of finding an optimal schedule is NP-hard and then give a fast online algorithm to solve it approximately. We also consider dividing a cluster into sectors and using multiple nonoverlapping frequency channels to further reduce the idle listening time of sensors. We conducted simulations on the NS-2 simulator and the results show that our polling scheme can reduce the active time of sensors by a significant amount while sustaining 100 percent throughput.

Index Terms—Sensor networks, heterogeneous networks, clusters, polling, multihop polling, scheduling.

1 Introduction and Background

The introduction of wireless sensor networks will enable a wide variety of applications, including environmental monitoring, medical treatment, emergency response, outer space exploration, and so forth. In a sensor network, a large number of sensors are deployed in a large area, with each sensor capable of collecting data and communicating with each other wirelessly. The challenge in designing a sensor network is that sensors must be organized into a robust multihop wireless network that should be able to function properly for a long period of time. This, in general, is a difficult problem since sensors have limited computing capabilities and very limited power supply. Recently, many researchers have started to consider heterogeneity as a way to solve this problem, that is, to deploy different types of nodes in a network [12], [21], [22], [23], [34]. In a heterogeneous sensor network, the basic sensors are simple and perform the sensing task, while some other nodes, often called the cluster heads, are more powerful and focus on communications and computations. Basically, the cluster head organizes the basic sensors around it into a cluster, where sensors only send their data to the cluster head and the cluster head carries out the long-range intercluster communications. The concept of a heterogeneous sensor network is shown in Fig. 1. The advantages of a heterogeneous sensor network include the following: First, it has better scalability than a flat network without hierarchies; second, the majority of nodes in the network, which are the basic sensors, can be made very simple and inexpensive; thus, the overall cost of the network can be greatly reduced. Note that a unique feature of this type of network is that the transmission in a cluster is asymmetrical: The message sent by a cluster head can be received *directly* by all sensors in the cluster, whereas the message sent by a sensor may have to be *relayed* by other sensors, that is, to travel multiple hops, to reach the cluster head. This is because the transmission ranges of the basic sensors are short due to their limited power supply, whereas the transmission range of the cluster head can be much longer since it has a far richer or even replaceable power supply. Also, note that the cluster heads can communicate with each other by organizing themselves into a wireless network consisting of only the cluster head nodes, which will be referred to as the second layer of the sensor network. Many existing results on wireless ad hoc networks can be applied to the second-layer network and, in this paper, we will mainly focus on the first layer, that is, the operations within a cluster.

In addition to improving the scalability and reducing the cost, introducing heterogeneousness will also help reduce the power consumption of the sensors. In a sensor network, it is usually assumed that sensors use contention-based distributed MAC protocols for media access. Although such protocols have advantages such as being robust and requiring no central controller, from the energy point of view, they are not the most efficient way since much energy can be wasted in collision, overhearing, collision avoidance, and idle listening. First, energy can be wasted in collisions because several sensors may decide to send packets at the

- *Z. Zhang is with the Department of Computer Science, Florida State University, Tallahassee, FL 32306. E-mail: zzhang@cs.fsu.edu.*
- *M. Ma and Y. Yang are with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794. E-mail: {mingma, yang}@ece.sunysb.edu.*

Manuscript received 10 May 2006; revised 31 May 2007; accepted 25 June 2007; published online 18 July 2007.
Recommended for acceptance by A. Zomaya.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0180-0506.
Digital Object Identifier no. 10.1109/TC.2007.70774.

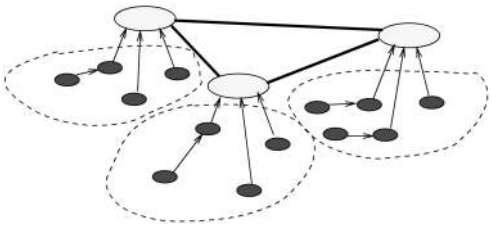

Fig. 1. A two-layered heterogeneous sensor network. The large nodes are the cluster heads. The small nodes are the basic sensor nodes.

same time and all of these packets have to be retransmitted. Second, energy can be wasted in overhearing because sensors may have to decode packets not destined for them. Third, energy can be wasted in collision avoidance because, to avoid collision, sensors may have to use control packets to coordinate with each other, which also consumes a significant amount of energy. Finally, energy can be wasted in idle listening because, when sensors send packets in a random manner, sensors have to be awake to constantly monitor the radio for possible data packets destined for them. Note that the power consumption ratio for sleeping, idle listening, receiving, and sending for typical sensors is 4.2 : 7.3 : 7.5 : 8.2 [10], which indicates that idle listening still consumes a comparable amount of energy as other active operations. Therefore, the contention-based MAC protocols may be best suited for applications where the environments can change rapidly due to their robustness. In other cases, for example, when the environment and the sensors are relatively static, for applications such as ground temperature monitoring, where the typical task of sensors is simply to gather the raw data and send it to the outside observer, other approaches are possible and have advantages in saving energy, especially in a heterogeneous sensor network with the presence of the powerful cluster head nodes. In this paper, we propose using *polling* in heterogeneous sensor networks for such applications to help reduce power consumption. Polling is different from the contention-based MAC protocols in that the packet sending is completely controlled by a central controller, which, in our case, is the cluster head: The cluster head sends out polling messages to poll sensors and only the polled sensors can send packets. Since the cluster head has full control of the cluster, energy wasting can be minimized as the cluster head can ensure that no collision and overhearing will occur and there will be no control packets needed from the sensors. The cluster head can also find a good polling schedule to collect data as fast as possible to reduce idle listening and thus prolong the lives of sensors. In this paper, we focus on finding fast collision-free polling schedules. Note that the unique challenge in finding polling schedules is that the cluster is a multihop network, whereas most existing wireless networks that adopt polling are single hop. We will show that, in a multihop cluster, the problem of completing polling in minimum time is NP-hard. We will then give a fast online algorithm to solve it which can also handle possible packet loss. It should be mentioned that, although we study polling in this paper, we are not overemphasizing the advantages of polling or considering it as being able to replace the contention-based MAC protocols completely; instead, we are proposing it for some most suited applications, as mentioned earlier.

The rest of the paper is organized as follows: Section 2 discusses some related work. Section 3 describes how a cluster operates. Section 4 shows that completing polling in minimum time in a multihop cluster is NP-hard and gives a fast online algorithm. Section 5 gives simulation results obtained by using the NS-2 simulator. Section 6 explores the possibility of dividing a cluster into sectors to further reduce the idle listening time of sensors. Section 7 considers using multiple nonoverlapping frequency channels to reduce the idle listening time of sensors. Finally, Section 8 concludes the paper.

2 Related Work

Adopting hierarchies in sensor networks has been considered in many works in the literature, see, for example, [4], [14], [24], [28], [35]. However, these works typically consider a homogeneous sensor network, where all nodes are identical, and focus on protocols for cluster forming and cluster head selection. In this paper, we consider a heterogeneous sensor network, where nodes are different, and mainly focus on the operations inside a cluster. We consider a heterogeneous sensor network because it has the advantages mentioned in the previous section, whereas a homogeneous network, although it can be more robust in case of node failure, may have a higher cost since every sensor can potentially be elected as the cluster head and, thus, more transmitting and storage capabilities are needed in every sensor. Also, note that cluster head selection is not needed in a heterogeneous network since there are nodes specifically designed as cluster head nodes.

Heterogeneous sensor networks have been considered in [12], [21], [22], [23]; however, these works typically assume that sensors use contention-based MAC protocols for media access, whereas we use polling to improve energy efficiency. Recognizing the need to reduce the idle listening time to prolong battery life, Ye et al. [9] introduced a new contention-based MAC layer protocol called SMAC in which sensors can enter the sleep mode periodically to save energy. However, as will be seen in the simulations section of this paper, in SMAC, the energy spent in idle listening is still quite significant as compared to the polling scheme we propose.

Polling in wireless networks have long been studied and used, for example, the 802.11 PCF and the Bluetooth network. These networks are one-hop networks, that is, the master node can reach the slave nodes with one hop and vice versa. Polling in one-hop networks is simple, where the master polls one slave at a time and the slave immediately replies in the next time slot. One-hop networks can also use Time Division Multiplexing (TDMA), where each node is given a unique time slot for data transmission. In this paper, we consider multihop networks in which sensors have to relay packets for other sensors due to the low transmission power of sensors. With lower transmission power, a multihop network has advantages in 1) prolonging battery life and 2) reducing intercluster interferences, thus improving the network-wide throughput. Polling and data collection in multihop sensor networks were studied, for example, in [16], [19]. However,

both assumed that the central controller has the same transmission range as ordinary sensors, in other words, as the sensor packets; the central controller's message also has to be relayed by sensors to reach sensors that are far from it. Note that we assume that the cluster head has a relatively high transmission power such that its polling message can be heard by all sensors in the cluster. Our assumption is quite reasonable and can greatly improve the efficiency of polling, as well as reducing the power consumption of sensors. Also, the problem of finding the optimal polling strategy is completely different under our assumption. In addition, only tree networks were considered in [19], whereas we consider general networks of an arbitrary topology.

The problem of sending packets from sensors to a single sink node with energy constraints has been studied in [31], [33]. However, the difference between our work and those in [31], [33] is profound. First, both [31] and [33] assume that data should be gathered by a data-forwarding tree, whereas it has been shown in [5] and [13] that a tree is not the best structure for data gathering applications. The best structure can be found by running a network flow algorithm, which is what we will adopt in our work. Second, in essence, [31] and [33] focus on traffic routing, whereas we consider both traffic routing and media access control.

3 Cluster Operations

In this section, we describe how the sensor network is organized and operated. Throughout this paper, we will use S to denote a sensor and t to denote a cluster head. We will assume that no node can send and receive at the same time and one node can send or receive at most one packet at a time.

3.1 Deployment

In heterogeneous sensor networks, the basic sensors can be deployed randomly as in homogeneous sensor networks. The cluster heads, on the other hand, should be more carefully deployed to make sure all basic sensors are covered, that is, each sensor can hear from at least one cluster head. However, since the number of cluster heads is small, their best locations can be found within a reasonable amount of time and they can even increase their transmission power to cover remote sensors.

3.2 Cluster Partition

Initially, the sensor network should be partitioned into clusters. This is called *Cluster Partition* and has been extensively studied for homogeneous networks [6], [4], [14] in which cluster head selection algorithms have been given. However, in a heterogeneous network, the problem of cluster partition is quite different. The main issues here are letting the cluster head know which sensors are in its cluster and letting the sensors know to which cluster they belong. Since the main focus in the paper is the operation within a cluster, in the following, we briefly describe a simple method for cluster partition. Note that we assume that cluster heads and sensors know their locations.

First, based on the IDs given to them, the cluster heads will broadcast a message containing their location information *in turn*, the cluster head with the lowest ID first. Note that this can be done within a reasonable amount of time since the number of cluster head nodes is relatively small. (For example, in a network with 1,000 sensors and assuming that each cluster has 50 sensors, there will only be 20 cluster head nodes.) Each sensor will then make a list of cluster heads it has heard from, that is, whose messages have been correctly received by the sensor, according to the received signal strength, largest signal strength first. After this, each sensor will know to which cluster it may belong and will choose the cluster head at the top of the list as its preferred cluster head.

Then, in turn, the cluster head starts the "discovering process," that is, starts to find which sensors should be in its cluster, in a way much like a Breadth-First Search. Since this is the same for all clusters, we only explain it for cluster 1. Cluster head 1, denoted as t_1, will send a message saying "all sensors that have chosen me as the preferred cluster head and are within a distance of D to me should report to me," where D is chosen such that, with high probability, the sensors can directly communicate with t_1. Each qualified sensor will then send a packet to t_1 which includes its ID. Since there may be more than one qualified sensor and t_1 does not know them yet, sensors must use a contention-based MAC protocol to communicate to t_1 at this time. This means that, although polling is mainly used, a MAC protocol should also be installed in the sensors. However, note that the contention-based MAC protocol need not be a complex one and it is only needed when partitioning the clusters and is never needed afterward. After waiting long enough (that is, the channel idle time longer than the maximum length of the contention window), t_1 decides that all such sensors have reported. It will add them to a list L and broadcast an acknowledgment packet to them. t_1 then asks the sensor in L with the smallest ID, say, S_1, to broadcast a message that asks sensors to report to S_1 if they 1) choose t_1 as the preferred cluster head, 2) have heard this message from S_1, and 3) have not been acknowledged by t_1. All such sensors will regard S_1 as their *parent* and S_1 will tell t_1 about these sensors. t_1 will add these newly discovered sensors to L and then ask another sensor, say, S_2, to do the same as S_1 did, and so on, until no new sensors can be discovered. It can be verified that, after this, every sensor that chooses t_1 as the preferred cluster head and has a path to reach to t_1 by visiting only sensors that choose t_1 as the preferred cluster head will have been discovered by t_1. Note that, since each sensor will have only one parent, there is a unique path from each sensor to the cluster head, along which packets can be forwarded. Also, to ensure correctness, link-level retransmission may be needed to prevent information loss.

After t_1 has finished, t_2 can discover its sensors in the same way, then $t_3, t_4, \ldots$, until the last cluster head. After the last cluster head has finished, we say the first "round of discovery" is completed. Note that, after the first round, it is very likely that the majority of sensors have already been discovered by the preferred cluster heads. However, some sensor still may not have been discovered because it may not have a path to its preferred cluster head. Such sensors are called the "orphan sensors." To help the orphan sensors

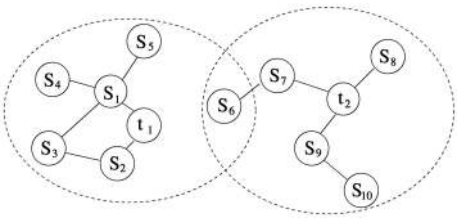

Fig. 2. A simple network.

find the cluster head, a second round of discovery is needed in which each orphan sensor may broadcast a message, according to the contention-based MAC protocol, saying that "nonorphan sensor who heard this message may add me to your cluster." The nonorphan sensor who responded first will be the parent of the orphan sensor and will report this new discovery to the cluster head. After a sufficiently long period of time, all sensors will be discovered.

As an example, a very simple network is shown in Fig. 2, where there are two cluster heads, t_1 and t_2, and 10 sensors. The communication range of the cluster heads is indicated by the ellipses, that is, sensors S_1 to S_5 can only hear from t_1 and sensors S_7 to S_{10} can only hear from t_2. S_6 can hear from both t_1 and t_2 and it is assumed that the signal strength from t_1 is stronger. If a sensor can send a packet to another node, there is an edge between them.

At first, t_1 and t_2 will broadcast their messages in turn, as shown in Fig. 3. After this step, S_1 to S_6 will choose t_1 as the preferred cluster head and S_7 to S_{10} will choose t_2 as the preferred cluster head.

Next, t_1 will try to discover sensors that can directly communicate with it. It will send a message and S_1 and S_2 will respond because they are within distance D to t_1, as shown in Fig. 4a. After this, S_1 will discover S_3, S_4, and S_5, as shown in Fig. 4b.

Next, t_2 will discover sensors S_7 to S_{10} in a similar way, as shown in Fig. 5a. Note that S_6 is an orphan because it chose t_1 as its preferred cluster head, but it cannot communicate with any sensor who has a path to t_1. Thus, S_6 will send a message and S_7 will respond to add S_2 to the cluster of t_2, as shown in Fig. 5b.

3.3 Connectivity and Compatibility

We can now focus on the operations inside a cluster. To control sensors, the cluster head needs to know the connectivity in the cluster, that is, a sensor can communicate with which other sensors. It should also know the compatibility, that is, whether a group of transmissions can occur simultaneously without interfering with each other. We now describe how the cluster head acquires this information.

Let $P(S_i, S_j)$ denote the signal strength received at sensor S_j when S_i is sending a packet. If the cluster head knows $P(S_i, S_j)$ for all $1 \leq i, j \leq n$, where n is the number of

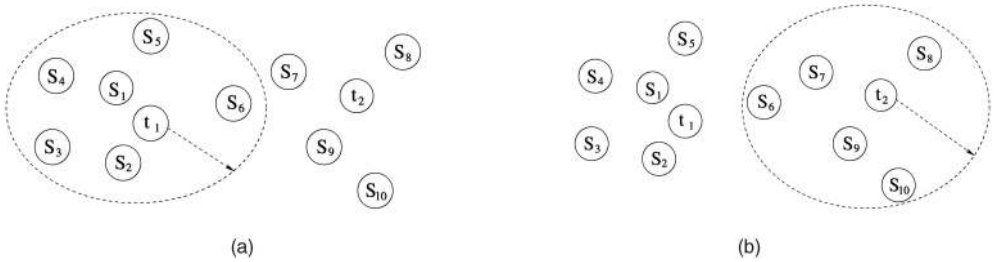

Fig. 3. t_1 and t_2 broadcast their messages in turn.

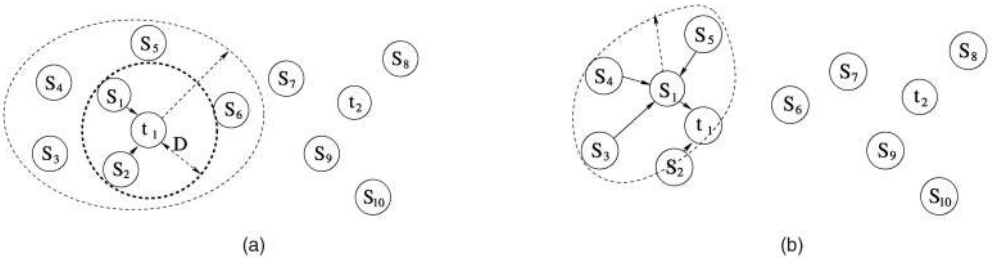

Fig. 4. (a) S_1 and S_2 respond to t_1's message. (b) S_1 discovers S_3, S_4, and S_5.

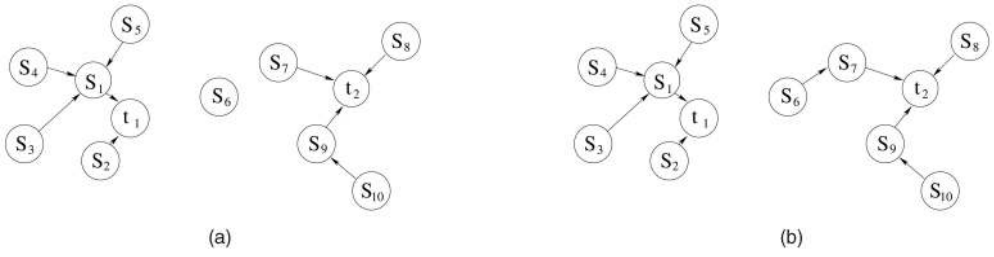

Fig. 5. (a) t_2 discovers S_7 to S_{10}. (b) S_6 joins the cluster of t_2.

sensors in the cluster, it can find out the connectivity and compatibility in the cluster as follows: To determine whether S_i can send a packet to S_j, the cluster head compares the *Signal-to-Noise Ratio (SNR)* $P(S_i, S_j)/N_0$ with a threshold γ, where N_0 is the typical noise power [2]. If the SNR is larger than γ, S_i can send a message to S_j; otherwise, S_i cannot send a message to S_j. To determine whether m transmissions, say, $S'_1 \rightarrow S_1, S'_2 \rightarrow S_2, \ldots, S'_m \rightarrow S_m$, are compatible, the cluster head compares the *Signal-to-Interference-and-Noise Ratio (SINR)*

$$\frac{P(S'_i, S_i)}{\sum_{j=1, j\neq i}^{m} P(S'_j, S_i) + N_0}$$

with γ for all $1 \leq i \leq m$ and the transmissions are compatible if the SINRs for all sensors are larger than γ [2].

To obtain $P(S_i, S_j)$, the cluster head can ask each sensor, one at a time, to send a packet and ask all other sensors to record the received signal strength. After all sensors have finished sending, the cluster head can *poll* the sensors to ask them to report the recorded information. Note that the cluster head can only allow one transmission at a time since it does not yet know the compatibility. The packet can be sent along the path established earlier for cluster partition. The size of the information that sensors need to report is actually quite small. For example, suppose there are 50 sensors in a cluster, each sensor has a 16 bit ID, and the received signal strength is measured in 8 bits. Then, each sensor needs to report around 150 bytes of information to the cluster head, which can be easily encapsulated in one or two packets.

Note that, since the wireless channel can be time varying and the noise power can fluctuate, the connectivity and compatibility obtained above are at best an *estimation* of the actual situation. However, it can be very accurate in a relatively static environment. Although the process of obtaining the connectivity and compatibility information is the overhead of using polling, which is not required in other approaches, it is needed only at the beginning phase or when the environment significantly changes, which can be expected to be quite rare in the applications we are considering. Thus, this overhead of polling is small in the long run. Also, note that, to get connectivity and compatibility, we have used the *measured signal strength* and have not used any models, which is because such models are best suited for mathematical performance analysis but not for protocol design. Neither have we used the location information because the measured signal strength is much more reliable in determining connectivity and compatibility than location information. Aguayo et al. [1] have shown that the connectivity cannot be directly determined by the locations of the nodes in many cases and the signal power received at a relatively long distance can be arbitrary due to multipath fading. To make the network robust, in this paper, we do not make any assumptions about the connectivity and compatibility and treat them as arbitrary.

3.4 Data Relaying Path

After finding the connectivity and compatibility, the cluster head will have enough information for controlling sensors in the cluster. It will first find the *data relaying path* for each sensor, which is the path by which the packets can be forwarded to the cluster head. Note that the data relaying paths are different from the paths established earlier when partitioning the clusters because the latter is only used temporarily to ensure that each sensor has a path to the cluster head, whereas the former is more load balanced, that is, there will be no sensors carrying too much traffic while others are carrying too little. For example, in cluster 1 in Fig. 5b, since S_3, S_4, and S_5 all chose S_1 as their parent, S_1 will carry the traffic of three other sensors, whereas S_2 does not carry any additional traffic. However, we can change the data relaying path of S_3 to $S_3 \rightarrow S_2 \rightarrow t_1$ so that the load of S_1 and S_2 is more balanced, as shown in Fig. 6.

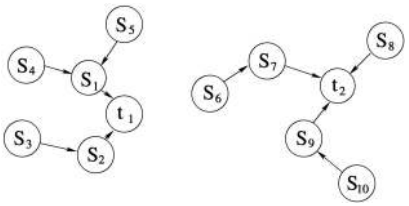

Fig. 6. Data relaying paths in cluster 1 with a more balanced load.

Since we target applications where the data generation rate is low, sensors can sleep for most of the time and wake up only for a short period to send the sensed data. The time between two consecutive wake-ups is referred to as a *cycle* and the time when sensors are active is referred to as a *duty cycle*. To find the relaying paths, define the *load* of a sensor as the average number of packets it needs to send out during a duty cycle, including its own packets and the packets from other sensors it relays. The problem of finding a set of balanced relaying paths can be formalized as a min-max problem: Find a routing strategy such that the maximum load of sensors is minimized. Chang and Tassiulas [5] and Bogdanov et al. [13] showed that it can be solved in polynomial time by formalizing the problem into a network flow problem, where either links or nodes have limited capacities. To be self-containing, we briefly explain the idea of the network flow formalization in the simplest case when each sensor has exactly one packet to send. For more information related to this issue, readers are referred to [5], [13]. Consider a cluster with n sensors. We draw a directed graph G according to the connectivities in the cluster. Let the cluster head be node t, that is, the *sink* of G. Each sensor, say, S_i, corresponds to two nodes s_i and s'_i in G, which are the "input" node and "output" node of S_i, respectively. There is an arc from s_i to s'_i with capacity δ, where δ is a positive integer. If sensor S_j can hear the message from S_i, there is an arc from s'_i to s_j with infinite capacity. Similarly, if the cluster head can hear S_i, there is an arc from s'_i to t with infinite capacity. Also, add a source node s to G, where there is an arc from s to s_i with unit capacity for all $1 \leq i \leq n$. We can see that, in this construction, δ limits the maximum load of sensors. After running a maximum flow algorithm on G, for example, the Ford-Fulkerson algorithm, if a flow of value n is found, the relaying paths for each sensor have been found, and no sensor's load will exceed δ. Therefore, we can start with a small δ, for example, 1, and then run the Ford-Fulkerson algorithm. If the value of the resulting maximum flow

found by the algorithm is less than n, increment δ by one and run the algorithm again until the value of the maximum flow is n. The runtime of this method is $O(n^3)$.

Chang and Tassiula [5] and Bogdanov et al. [13] have mainly focused on the theoretical part of the problem and, in the following, we describe some implementation-related issues in a heterogeneous cluster and our solutions. First, in general, to find the relaying paths, the network flow algorithm only needs to run occasionally by using the *average traffic intensity* of sensors, which is the average number of packets generated by a sensor in a cycle. Second, sensors can use *source routing* to make sure that the packets are sent along the right paths. However, to save energy, the path need not be added to the data packets explicitly because, equivalently, we can let each sensor remember a one-hop routing table for all of its dependents, where a dependent of sensor S is a sensor whose relaying path visits S. Third, note that a sensor may have several relaying paths. For example, if, on average, three packets are generated by sensor S_1 in a cycle, the maximum flow algorithm may find two paths, with path 1 and path 2 carrying two units of flow and one unit of flow, respectively. To simplify the control, in a duty cycle, we would like sensors to send packets only on one path. Thus, to balance the load, sensors can send packets on these paths alternatively in proportion to the units of flows the paths carry, whereas, in this example, S_1 can send packets along path 1 for two duty cycles and along path 2 for one duty cycle.

In the following sections, we will assume that the relaying paths have been found and, in one duty cycle, a sensor will send its packets along one fixed relaying path.

3.5 Polling

We can now describe the basic polling operations inside a cluster. More details on polling, in particular, how the *polling schedule* is found, are provided in later sections. First, near the beginning of a duty cycle, sensors wake up and enter the listening mode at the time specified by the cluster head before they went to sleep in the last duty cycle. This can be achieved by setting a timer in each sensor and letting the sensor wake up when the timer expires. Due to possible clock drifts, sensors may not wake up at exactly the same instant; however, the differences should be small. Sensors will then wait for a message broadcast by the cluster head indicating the beginning of a duty cycle. In this way, the initial synchronization between sensors and the cluster head can be established. The cluster head will then poll the sensors according to a polling schedule in a time-slotted manner, where the length of a time slot is the time for sending a packet. The beginning of a time slot is indicated by a polling message broadcast by the cluster head that contains the information about which sensors are polled and which sensors should receive packets at this time slot. After receiving this message, sensors that are polled will send their packets; also, sensors that received packets from other sensors in the previous time slot will relay such packets to the next hop. Note that, since packets are relayed immediately without delay, no queuing is needed in the intermediate sensors, which greatly reduces the memory size of the sensors. The cluster can thus be imagined as similar to a pipelined system, where the polling message acts as the clock. The polling continues until all packets have been received by the cluster head. After this, the cluster head will broadcast a message to set all sensors to the sleep mode and also inform them of the next wake-up time. Note that, to reduce the length of a time slot, no link-level retransmission is used: If a packet is lost, the cluster head will poll the sensor again.

3.6 Fault Detection

In the case when sensors or cluster heads fail, the network must have a mechanism to detect the faulty nodes and recover from the failure. We focus on the failure of sensors since the cluster head is much less likely to fail. Even if there is such a failure, it is much easier to detect and repair as the cluster heads themselves constitute a more intelligent network and are closer to the outside human observer and thus can be repaired or replaced even manually.

The cluster head detects a faulty sensor in its cluster when it fails to receive the packet after polling some sensors. For example, in cluster 1 in Fig. 5b, a packet relaying path is $S_4 \rightarrow S_1 \rightarrow t_1$. Suppose t_1 polls S_4. If S_4 fails, S_1 will not be able to hear from it. In this case, we can let S_1 send its own packet in the next time slot, which will be received by t_1. When t_1 receives the packet and finds out that it is from S_1, it knows that something is wrong with the link between S_4 and S_1. It can poll S_4 several more times and, if still nothing is received from S_4, it will determine that S_4 has died. When a sensor dies, the cluster head will run the algorithm for finding data relaying paths again and tell all sensors the new paths.

If a sensor, say, S_i, cannot find a path to the cluster head due to the failure of some sensors, the cluster head can tell S_i to join its next preferred cluster, say, cluster 2. t_1 can tell t_2 about the joining of S_i and t_2 can ask S_i to send a packet and let all other sensors in cluster 2 measure the signal strength to update the connectivity and compatibility in cluster 2.

3.7 Removing Intercluster Interference

In practice, there may be many clusters in a wireless sensor network. With the proposed polling scheme, sensors in the same cluster do not interfere with each other. However, at the boundaries of the clusters, sensors belonging to different clusters may still do so if their cluster heads decide to poll them at the same time. This is called the intercluster interference.

The simplest way to remove intercluster interference is to allow data transmission in only one cluster at a time. It works for the case when the number of clusters is not large and when the data transmission within a cluster can be completed in a relatively short time period as compared to a cycle. Another more efficient way is to let sensors in nearby clusters operate in different radio channels. Regarding a radio channel as a color, this problem is equivalent to giving interfering clusters different colors. Denoting each cluster as a vertex and using an edge to connect two vertices if the two clusters will cause interference with each other, the problem is reduced to the standard vertex-coloring problem. Many algorithms can be applied to find practically good coloring schemes, in particular, if the graph is planar, it is known that four colors or four radio channels will be sufficient [15].

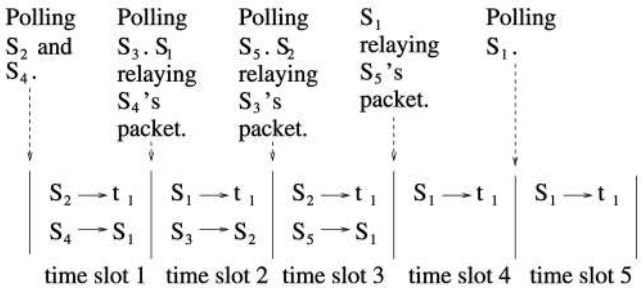

Fig. 7. A polling schedule for cluster 1, shown in Fig. 5b, when each sensor has exactly one packet.

4 Polling in Multihop Clusters

In this section, we address the main issue of this paper, which is to find a polling schedule in a cluster. To reduce energy consumption, we study the problem of finding the optimal polling schedule, which is a schedule using the minimum time. We show that this problem is NP-hard and then give a fast online algorithm to solve it approximately.

4.1 An Example of Polling in a Multihop Cluster

The major difference between single-hop polling and multihop polling (MHP) is that, in multihop clusters, the cluster head can poll more than one sensor simultaneously if their packet transmissions do not cause collision. As a simple example, consider cluster 1 in Fig. 6. In this cluster, the relaying paths are $S_1 \rightarrow t_1$, $S_2 \rightarrow t_1$, $S_3 \rightarrow S_2 \rightarrow t_1$, $S_4 \rightarrow S_1 \rightarrow t_1$, and $S_5 \rightarrow S_1 \rightarrow t_1$. Suppose, at the beginning of a duty cycle, t_1 knows that, in this duty cycle, every sensor has one packet to send. Since the packets from S_3, S_4, and S_5 have to use two hops to reach t_1, if only one sensor is polled at a time, that is, a sensor is polled only after the packet from the sensor polled previously has been received by t_1, eight time slots are needed in total. However, suppose, based on the reported measured signal strength, t_1 knows that transmissions $S_1 \rightarrow t_1$ and $S_2 \rightarrow t_1$ are compatible with other transmissions. t_1 can then poll the sensors in the schedule shown in Fig. 7, which needs only five time slots, as explained in the following:

- **Time slot 1.** t_1 polls S_2 and S_4, which results in two transmissions, $S_2 \rightarrow t_1$ and $S_4 \rightarrow S_1$. Since the two transmissions are compatible, after this time slot, both packets will be correctly received, that is, t_1 will have S_2's packet, and S_1 will have S_4's packet.
- **Time slot 2.** S_1 must forward S_4's packets to t_1, which means that there will be transmission $S_1 \rightarrow t_1$ since we do not allow packets to be queued at intermediate sensors. However, since $S_1 \rightarrow t_1$ is compatible with $S_3 \rightarrow S_2$, t_1 can also poll S_3 at this time slot. After this time slot, t_1 will have collected the packets from S_2 and S_4, whereas S_2 will have S_3's packet.
- **Time slot 3.** S_2 must forward S_3's packets to t_1. t_1 can also poll S_5 at this time slot. After this time slot, t_1 will have collected the packets from S_2, S_3, and S_4, whereas S_1 will have S_5's packet.
- **Time slot 4.** S_1 forwards S_5's packets to t_1. After this time slot, t_1 will have collected the packets from S_2, S_3, S_4, and S_5.
- **Time slot 5.** t_1 polls S_1. After this time slot, t_1 will have collected all of the packets.

Note that, in this example, the cluster head knows how many packets each sensor has, that is, has the *packet number information*. This could be true in many applications, for example, in ground temperature monitoring in which each sensor samples data periodically, generates data at a fixed rate, and sends exactly one packet in each cycle. However, in some other applications, the cluster head may not have such information and some sensor may have no packet to send, while others may have more than one packet to send. In the following, we will start with the case when the cluster head has the packet number information and move on to the more complicated case later.

4.2 Polling with Packet Number Information

We first consider polling when the cluster head knows how many packets each sensor has before a duty cycle. We study the problem of finding the optimal polling schedule, that is, the schedule that finishes the polling in minimum time. We say a sensor is in level i if its hop count is i, that is, is i hops away from the cluster head. If all sensors are in level 1, the scheduling reduces to the single-hop polling and is trivial. We call the problem of finding the optimal polling schedule when some sensors are in levels higher than 1 the *MHP* problem. We will show that the MHP problem is NP-hard under virtually all scenarios, which will justify our use of a simple greedy algorithm for finding the polling schedule.

4.2.1 MHP Problem

We first study the MHP problem when the number of packets can be arbitrary. To see that the MHP problem is NP-hard in this case, we first consider a special structure called "two-level star with relaying only sensors in the first level," which is referred to as *TSRF*. TSRF is a tree whose root is the cluster head. There are branches connected to the root, where each branch consists of two sensors. Fig. 8a shows a TSRF with five branches. More specifically, let t be the root. Sensors denoted by $S_1, S_2, S_3, \ldots$ are connected to t, that is, are in the first level, and sensors denoted by $S'_1, S'_2, S'_3, \ldots$ are in the second level, where S'_1 is only connected to S_1, S'_2 is only connected to S_2, and so forth. Each sensor in the second level has exactly one packet to send and sensors in the first level have no packet to send. The relaying path for S'_i is $S'_i \rightarrow S_i \rightarrow t$. Some transmissions, say, $S'_1 \rightarrow S_1$ and $S_3 \rightarrow t$, can occur at the same time if they do not cause collision.

The problem is given as follows: Given a TSRF and the compatibility, does there exist a schedule by which all packets can be received by the cluster head by time slot k, where k is a given positive integer? We call it the *TSRF Polling* problem and sometimes simply refer to it as the *TSRFP* problem.

Lemma 1. *The TSRFP problem is NP-complete.*

Proof. It is clear that this problem belongs to NP. To see that it is NP-complete, we reduce the well-known NP-complete *Hamiltonian Path* (HP) problem to it.

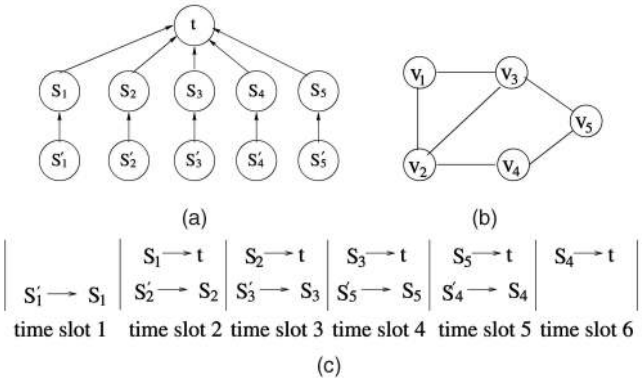

Fig. 8. (a) A TSRF with five branches. (b) A graph with five vertices. (c) A polling schedule for Fig. 8a with compatibility defined by Fig. 8b. This schedule corresponds to the Hamiltonian Path $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_5 \rightarrow v_4$.

Given any instance of the undirected HP problem, an instance of the TSRFP problem can be constructed as follows: Denote the graph of the HP problem as G and denote the vertices in G as $v_1, v_2, v_3, \ldots$. Each node in G corresponds to a branch of the TSRF. For example, the TSRFP instance for the graph shown in Fig. 8b is the one in Fig. 8a, where v_1 corresponds to branch $S_1' \rightarrow S_1 \rightarrow t$, v_2 corresponds to branch $S_2' \rightarrow S_2 \rightarrow t$, and so forth. If two vertices are adjacent, for example, v_1 and v_2, transmissions $S_1 \rightarrow t$ and $S_2' \rightarrow S_2$ are compatible, and so are transmissions $S_2 \rightarrow t$ and $S_1' \rightarrow S_1$. Otherwise, these two pairs of transmissions are not compatible. k is set to be $n+1$, where n is the number of vertices in G.

Note that, in a schedule that needs only $n+1$ time slots, the transmissions must be back to back: All sensors in the second level must start to send in consecutive time slots with no "pause" in between. Sensors S_i' and S_j' can start in consecutive time slots if and only if $S_i \rightarrow t$ and $S_j' \rightarrow S_j$ are compatible. In this case, by our construction, there is an edge connecting v_i to v_j in G. Therefore, this schedule determines an HP in G. For example, as Fig. 8c shows, a polling schedule using six time slots for the TSRF shown in Fig. 8a determines an HP $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_5 \rightarrow v_4$ in Fig. 8b. Similarly, it can be verified that an HP in G can always be used to determine a polling schedule that needs only $n+1$ time slots. Thus, the TSRFP problem is NP-complete. □

Since any algorithm that solves the MHP problem can solve the TSRFP problem, we have the following:

Theorem 1. *The MHP problem is NP-hard.*

Note that, in the proof, we constructed a TSRF whose compatibility is determined by an arbitrary graph. A natural question is whether the compatibility in a TSRF can indeed be arbitrary. We verify this with the following interference model, which is the physical model in [18] without the power-law signal decay assumption. We assume that, in the TSRF, two transmissions, say, $S_i' \rightarrow S_i$ and $S_j \rightarrow t$, do not interfere with each other if and only if the following two inequalities hold:

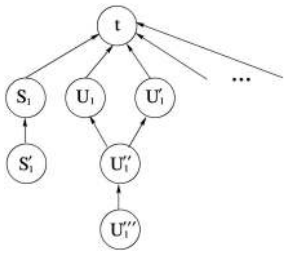

Fig. 9. The auxiliary branch for branch $S_1' \rightarrow S_1 \rightarrow t$.

$$\begin{cases} P(S_i', t) < \gamma P(S_j, t) \\ P(S_j, S_i) < \gamma P(S_i', S_i), \end{cases}$$

where γ is a positive constant. In wireless environments, the received signal power can be arbitrary [1]. Thus, we can let $P(S_j, t)\gamma$ be larger than any $P(S_i', t)$ for all i and j. Also, let $P(S_i', S_i) = \eta$ for all i. If there is an edge between v_i and v_j, let $P(S_j, S_i) < \eta\gamma$; otherwise, let $P(S_j, S_i) > \eta\gamma$. This will produce the desired compatibility. It can be shown that the needed compatibility can be constructed for other proofs used in this paper in a similar manner.

So far, we have shown that the MHP problem is NP-hard when the number of packets can be arbitrary. However, as mentioned earlier, in some applications, sensors sample data periodically and generate exactly one packet per cycle. Thus, a special case of the MHP problem needs to be studied in which each sensor has exactly one packet to send. We call it *Exact One Packet MHP* problem and abbreviate it as the *X1MHP* problem. We now show that this problem is still NP-hard.

Theorem 2. *The Exact One Packet Multihop Polling problem is NP-hard.*

Proof. We prove this by reducing TSFRP to it. Given any instance of TSFRP, we construct an instance of X1MHP as follows:

For each branch in the TSFRP instance, say, $S_1' \rightarrow S_1 \rightarrow t$, we add another auxiliary branch, as shown in Fig. 9. The relaying path for U_1''' is $U_1''' \rightarrow U_1'' \rightarrow U_1' \rightarrow t$. The relaying path for U_1'' is $U_1'' \rightarrow U_1 \rightarrow t$. U_1' and U_1 send their packets directly to t. Transmission $U_1'' \rightarrow U_1'$ is compatible with $S_1 \rightarrow t$. Other transmissions in this auxiliary branch are not compatible with any other transmissions.

In any optimal solution to the X1MHP problem, suppose sensor U_1''' starts to send its packet at time slot α. At time slot $\alpha+1$, U_1'' will relay this packet to U_1'. If sensor S_1 is not sending its packet to t at time slot $\alpha+1$, we can "move" it to this time slot since $U_1'' \rightarrow U_1'$ is compatible with $S_1 \rightarrow t$ and the resulting schedule should still be optimal. After pairing them up, we move them to the beginning of the schedule, that is, let U_1''' start to send at time slot 1 and let S_1 start to send at time slot 2. The resulting schedule should still be optimal since $U_1''' \rightarrow U_1''$ and $U_1' \rightarrow t$ are not compatible with any other transmissions and thus are not occurring at the same time with any other transmissions in the optimal schedule. For the same reason, U_1'', U_1', and U_1 can also be moved to the front, that is, start to send at time slots 4, 6, and 7, respectively. S_1 and sensors in its auxiliary

TABLE 1
The Polling Algorithm

```
Input : The list of polling requests.
Output: Polling schedule.
while the request list is not empty
   if a packet has been received
      Delete the request for this packet
   end if
   if a packet expected to arrive has not been received
      Set the request for this packet to active.
   end if
   while TRUE
      Grant an active request if it does not cause collision.
      Mark this request as idle.
      break from the loop if no request can be granted.
   end while
   wait until next time slot.
end while
```

branch will have finished sending by time slot 7. Then, similarly, we can pair up S_2 with its auxiliary branch and move them to the beginning of the schedule, starting at time slot 8. The same can be done to $S_3, S_4, \ldots, S_n$, where n is the number of branches in the TSRF. S_i and sensors in its auxiliary branch will start sending at time slot $7(i-1)+1$ and will finish sending at time slot $7i$.

As a result, the optimal schedule will consist of two parts. The first part is from time slot 1 to time slot $7n$, which is for sensors $S_1, S_2, \ldots, S_n$ and sensors in their auxiliary branches. Following the first part, the second part starts at time slot $7n+1$, which must be an optimal schedule for the TSFRP instance. □

4.2.2 An Online Polling Algorithm

In this section, we focus on finding an algorithm that gives suboptimal contention-free polling schedules. Note that, since successful packet delivery is not guaranteed in wireless communications, the algorithm should also be able to handle possible packet loss. The cluster head knows that a packet is lost somewhere along the path if it did not receive the packet after polling the sensor. To solve this problem, the cluster head can simply poll the sensor again. Therefore, the scheduling algorithm should be an *online* algorithm since new polling requests may come (actually the old ones come again) when the polling is going on.

The NP-hardness of the problem and the online requirement left us no other choices but to adopt a very simple algorithm, described in Table 1. We call a packet a sensor intends to send a *request*. The input to the algorithm is all of the requests. Initially, all requests are set to be *active* and are stored in a list. Before each time slot, the algorithm scans the list for active requests according to an arbitrary order and grants a request, that is, allows the packet to be sent at this time slot, if its transmissions in this and the following time slots are compatible with all other already granted transmissions. When the maximal number of requests has been granted, the algorithm halts and waits for the next time slot. After a request is granted, it becomes *idle* and will be deleted from the list after the packet has been received. If the packet has not been received, the request will become active again.

At a time slot, say, α, to see whether a request can be granted, suppose the source sensor's hop count is c. The algorithm will check whether the c transmissions of this request are compatible with the transmissions in the existing schedule from time slot α to $\alpha+c-1$. Let m be the number of requests that have been granted earlier whose packets are still on their way to the cluster head. There will be at most m granted transmissions in each time slot from α to $\alpha+c-1$. To see the compatibility of the new request at any time slot, the cluster head can use the method described in Section 3.3 based on the received signal strength that takes $O(m)$ time. Thus, $O(mc)$ time is needed to check whether a request is compatible or not with already granted requests.

4.3 Polling with No A Priori Packet Number Information

We can now discuss the problem of polling when the cluster head does not know the number of packets each sensor has before a duty cycle. Note that, with or without the packet number information, the cluster head has to poll some sensor first. A sensor who has to forward another sensor's packet may append its ID and its packet number information to the packet. Therefore, for example, if the relaying path for S_i is $S_i \rightarrow S_j \rightarrow t$, after polling S_i, the cluster head will have the packet number information of both S_i and S_j. In light of this fact, we propose polling the sensors in two phases. In the first phase, the cluster head polls a subset of sensors, denoted by π, under the constraint that the union of the relaying paths of sensors in π must cover all sensors in the cluster. Therefore, after the first phase, in addition to getting some data packets, the cluster head will also have all of the packet number information. Then, in the second phase, the cluster head will be able to poll the sensors as discussed in Section 4.2, according to the algorithm in Table 1.

Note that, in the first phase, the cluster head can poll the sensors by assuming that each sensor in π has exactly one packet. Of course, it may happen that some sensor in π, say, S_i, does not have a packet. In this case, S_i can send an "empty" packet and S_j can take this chance to send its packet to the cluster head, if it has one, again assuming that the relaying path for S_i is $S_i \rightarrow S_j \rightarrow t$. Clearly, the cluster head will fail to get a data packet after polling a sensor in π only if all sensors in the relaying path have no packet to send. Since the number of packets in the sensors can be arbitrary, the cluster head may not be able to guarantee that it can get a packet after polling a sensor. Therefore, to optimize the operations in the first phase, it is reasonable to consider the problem of finding π and a polling schedule such that the first phase can be finished in minimum time. However, it is not hard to see that this problem is still NP-hard because, clearly, in the TSRF shown in Fig. 8a, π must be the sensors in the second level and the problem reduces to finding a minimum time polling schedule in a TSRF, which has been proven to be NP-hard. We therefore propose solving the problem in two steps, that is, breaking the problem of *jointly* finding π and the polling schedule into two subproblems and solving them one by one, where the first subproblem is to find π and the second subproblem is to find a polling schedule after π has been given. Since the second subproblem can be solved by the algorithm in Table 1, we need only focus on finding π.

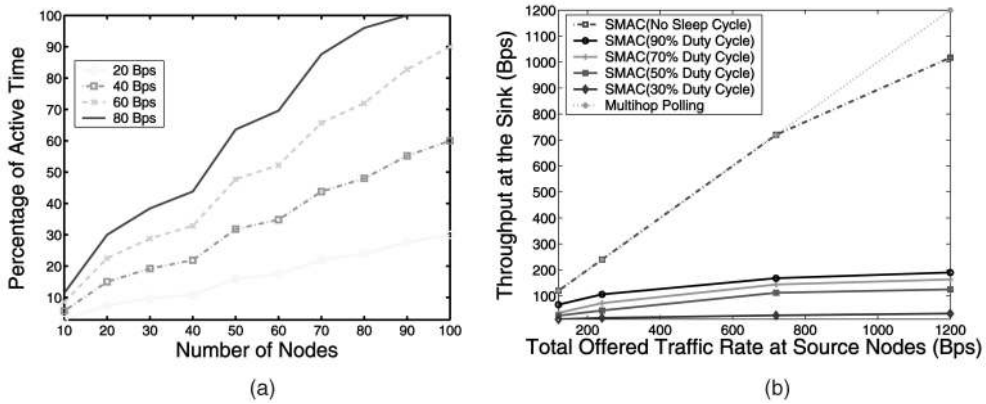

Fig. 10. (a) Percentage of active time of sensors as a function of cluster size and data generating rate when polling is used. (b) Throughput of the cluster with 30 sensors as a function of total offered load.

A "good" π should likely result in a short polling schedule. We therefore propose using the following criterion: Choose π such that the relaying paths cover all sensors in the cluster and have minimum total hop counts. To solve it, each sensor can be regarded as an element and each relaying path can be regarded as a subset, where an element is in a subset if the relaying path covers the sensor. Also, each subset has a cost equal to the hop count of the relaying path. The problem can then be formalized as finding a group of subsets that cover the whole set with minimum total cost. This problem is a case of the *Weighted Set Cover Problem* and is also NP-hard, but suboptimal solutions can be found by a fast greedy algorithm. This greedy algorithm iteratively chooses a subset that has the minimum *covering cost*, where the covering cost of a subset is defined as the ratio of the cost of the subset divided by the number of *currently* uncovered elements it contains, until all elements have been covered.

4.4 Discussions

Note that we have treated the problems related to heterogeneous sensor networks separately, that is, we first partition the network into clusters, then find the relaying paths, and then find the polling schedules. It may be tempting to consider them jointly since this allows more flexibility and possibly will result in better solutions. However, this approach is impractical since it is clearly NP-hard because the polling problem in a single TSFR, which is NP-hard, is a special case of it, which is why we have chosen to break it into several subproblems and solve them one by one in a greedy way. Also note that we have mainly considered the case when the packets are not queued at the intermediate sensors, that is, after receiving a packet, a sensor will forward the packet to the next hop in the next time slot immediately and will not temporarily store it in its buffer. As mentioned earlier, the advantage of it is that it requires less memory in the sensors and thus may reduce the cost. In addition, as the Appendix shows, allowing the queuing of packets will not make the polling problem easier, which is another reason why we have adopted the current approach.

5 Performance Evaluations

In this section, we evaluate the performance of the proposed polling scheme based on the results obtained by the NS-2 simulator. We assume that all sensor nodes are uniformly deployed within a $200 \times 200\ m^2$ 2D square and the cluster head is located at the center of the square. A two-ray propagation model is used to describe the feature of the physical layer. With the maximum transmission power $0.858\ mW$, each node can communicate with other nodes as far as $40\ m$ away. The radio bandwidth is 200 Kbps. CBR traffic on top of UDP is generated to measure the throughput. Each packet has a fixed size of 80 bytes, including header and payload. The simulation runs for 1,000 sec, which contains a 100 sec warm-up period. All simulation data are collected from 100 to 1,000 sec.

5.1 Percentage of Active Time

The major goal of our polling scheme is to reduce the active time of sensors. Fig. 10a shows the percentage of active time needed to ensure that all packets are received by the cluster head, where the number of sensors in a cluster ranges from 10 to 100 and the data generating rate ranges from 10 to 80 Bps. We can see that, in a modest-sized cluster with 30 sensors, when the data generating rate is 60 Bps, sensors only need to be active for about 30 percent of the time.

It can also be seen that, when the number of sensors increases or when the data generating rate increases, the active time of sensors will increase to ensure packet delivery. Note that, for a given data generating rate, for example, 80 Bps, when the number of nodes increases to 90, all sensors in the cluster must be active for *all* time. This implies that, under a certain data generating rate, there is a maximum size of a cluster above which packets will be lost. Thus, the size of a cluster should be carefully chosen such that no packets are lost and sensors can also enjoy long sleeping time.

5.2 Comparison with SMAC

To show the effectiveness of the proposed polling scheme, we compare its performance with an energy-efficient MAC protocol, named SMAC [9]. Similar to our polling scheme, SMAC also allows sensors to sleep periodically to save energy. We will mainly compare the throughput of the two

schemes, which is defined as the average number of packets received by the cluster head in a given time period. Since SMAC is only for the MAC layer, to find the relaying path for each sensor, we use the Ad Hoc On-Demand Distance Vector (AODV), which is an efficient topology-based routing protocol. The simulations for SMAC were carried out with the code contributed by the designer of SMAC, available on the Web.

Fig. 10b shows the throughput of a cluster with 30 sensors, when the total offered load is 210, 750, and 1,200 Bps (correspondingly, the data generating rates at individual sensors are 7, 25, and 40 Bps, respectively). It can be seen that, under all offered loads, our polling scheme achieves 100 percent throughput, that is, all packets generated by sensors are correctly received by the cluster head. We also measured the throughput of SMAC+AODV, when the active time of sensors is configured as 30 percent, 50 percent, 70 percent, 90 percent, and 100 percent. Quite surprisingly, as can be observed from the figure, the throughput of SMAC+AODV is far less than the total offered load when the active time is not 100 percent. Note that the percentage of the active time of sensors under the polling scheme is less than 20 percent, as can be derived from Fig. 10a, which is much less than the active time of SMAC. Thus, we can conclude that our polling scheme has much better throughput performance than SMAC, even with much shorter sensor active time.

One of the major reasons for the poor performance of SMAC+AODV is that many control packets were generated for routing since sensors must frequently use AODV to find relaying paths as the path used previously may no longer be valid after some sensors along the path have entered the sleeping mode. This large number of control packets will reduce throughput and increase collision. Another reason is that, unlike the centralized polling, SMAC allows sensors to compete for channel access randomly. As the data rate increases, more and more collisions will occur due to this random channel access competing, which will also reduce the total throughput. This is why, even when sensors are fully active, the throughput of SMAC+AODV is still not 100 percent when the total offered traffic rate becomes as high as 1,200 Bps. Overall, the inefficiency of SMAC+AODV is due to the fact that they are designed for random traffic, whereas the traffic in a cluster is not completely random, for example, all traffic is destined for the cluster head. For such applications, polling has a clear advantage.

6 Partitioning a Cluster into Sectors

Fig. 10a shows that the active time is longer in a larger cluster than in a smaller cluster. This leads us to further consider partitioning a cluster into smaller *sectors* and letting each sector wake up and do data transmission in turn to reduce the active time of sensors. Note that, since sectors can be considered as small clusters, their routing and polling mechanisms are the same as for the clusters described in previous sections.

6.1 Optimal Sector Partition

Define the *polling time* of a sector as the time needed for the cluster head to finish polling. The lifetime of a sensor is determined by its transmission load and the polling time of the sector. We can assume that the lifetime of sensor S_i is reversely proportional to the *power consumption rate* $\alpha R_i + \beta T$, where R_i is the transmission load of S_i, T is the polling time, and α and β are constants. Since our primary goal is to prolong sensor lifetime, we may want to find a partition by which the maximum power consumption rate of sensors can be minimized. However, the problem is NP-hard by this criterion since finding the optimal polling schedule is NP-hard. Another criterion that makes sense is given as follows: A partition is optimal if the maximum *pseudopower consumption rate* is the minimum. The pseudopower consumption rate of S_i is defined as $\alpha R_i + \beta' L$, where R_i is the load of S_i and L is the number of sensors in the sector S_i belongs to since the polling time is roughly proportional to the number of sensors involved. Unfortunately, even when reduced to this, the optimal partitioning problem is still NP-hard.

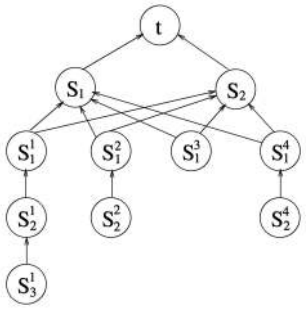

Fig. 11. The construction of the CPAR problem for set {3, 2, 1, 2}.

The problem is given as follows: Given a cluster and its connectivity, does there exist a partition such that $\max\{\alpha R_i + \beta' L\} \leq \Theta$ for a given Θ? We call it the *Cluster Partition* problem and refer to it as *CPAR*.

Theorem 3. *The CPAR problem is NP-complete.*

Proof. The CPAR problem clearly belongs to NP. To see that it is NP-complete, we use the following *Partition* problem: Given a set of positive integers $\{B_1, B_2, \ldots\}$, can the integers be partitioned into two subsets where the sums of the numbers in the two subsets are equal?

Given any instance of the Partition problem, we construct an instance of the CPAR problem as follows: Draw two sensor nodes, S_1 and S_2, which are connected to the cluster head t. For the ith integer in the set denoted as B_i, draw B_i sensors denoted as S_1^i to S_δ^i, where $\delta = B_i$. These δ sensors are in the same branch in which S_j^i is connected only to S_{j-1}^i for $1 < j \leq \delta$ and S_1^i is connected to both S_1 and S_2. Θ is set to be $(n/2+1)(\alpha+\beta')$, where $n = \sum B_i$. For example, the construction for set {3, 2, 1, 2} is shown in Fig. 11.

Note that this cluster can be divided into at most two sectors since there are only two sensors, S_1 and S_2, that can directly communicate with the cluster head. Also note that the cluster cannot be a single sector since, if it is, the pseudopower consumption rate of either S_1 or S_2 will exceed Θ. Thus, if there is a partition satisfying Θ, the cluster must be divided into two sectors, with S_1 and S_2 in different sectors. In each sector, S_1 and S_2 must have exactly $n/2$ dependents. Therefore, the partition of the sector gives a solution to the Partition problem. It can be

easily verified that the reverse is also true. For example, for the cluster shown in Fig. 11, we can let the first and third branches be in the same sector as S_1 and let the second and fourth branches be in the same sector as S_2, which is to partition set {3, 2, 1, 2} into {3, 1} and {2, 2}.□

6.2 Heuristic for Sector Partition

We now describe a heuristic for partitioning a cluster into sectors. Since the length of a duty cycle is short as compared to the length of a cycle, it is not necessary to set a limit on the number of sectors in a cluster. First, in the simplest case, if the union of the optimal relaying paths found by the maximum flow algorithm is a tree, we can let each first-level branch be a sector, where a first-level branch is defined as a sensor that is one hop away from the cluster head and all of its dependents. After this partition, the load of sensors remains the same and the numbers of sensors in all of the sectors are likely to be evenly distributed. However, more commonly, the union of the optimal relaying paths is not a tree. In this case, we modify the relaying paths into a tree and then let each first-level branch be a sector. Note that, when the union of the relaying paths is not a tree, there must be *flow-splitting sensors*, that is, sensors sending packets not only to one sensor but to multiple sensors. We can modify the relaying paths of such sensors to let them send packets to only one sensor or force them to "choose a parent." This is referred to as "flow merging." A flow-splitting sensor chooses a particular sensor as its parent if the maximum load of sensors along the path from the chosen parent to the cluster head is the minimum. To make sure that there are no flow-splitting sensors along this path, flow merging can start at flow-splitting sensors closest to the cluster head. After all flow-splitting sensors have chosen parents, the union of the relaying paths will become a tree and we can let each first-level branch be a sector. This method is conceptually simple and practically effective, as will be shown by our simulations.

6.3 Effects of Dividing a Cluster into Sectors

We studied the lifetime of a cluster both when divided into sectors and when not divided into sectors, while sustaining 100 percent throughput, and the lifetime ratio of the former over the latter is shown in Fig. 12. The settings of the simulations are the same as in Section 5. It can be seen that the ratio is always larger than 1, which means that, by dividing a cluster into sectors, the sensor lifetime will always increase. Also, because, usually, larger clusters can be divided into more sectors, the increase in lifetime is greater for larger clusters.

7 Multichannel Polling

Another way to reduce the polling time is to use multiple interference-free radio channels within a cluster or a sector. If a transmission causes large interferences in one frequency channel, it can be scheduled at another channel. Thus, with multiple frequency channels, more simultaneous transmissions can be supported and the polling time can be reduced. Note that the same mechanism for multichannel polling can be applied to either a cluster or a sector and, in the following, we will describe it as applied to a cluster without sectors.

Our algorithm for determining the polling schedule for multichannel polling is very much the same as that for single-channel polling, except that, to see whether a transmission can be scheduled, the scheduler will try more than one frequency channel. If there are a total of f channels, $F_1, F_2, \ldots, F_f$, the scheduler will check them in turn until the new transmission can be fitted in one of the channels *without* changing the current schedule.

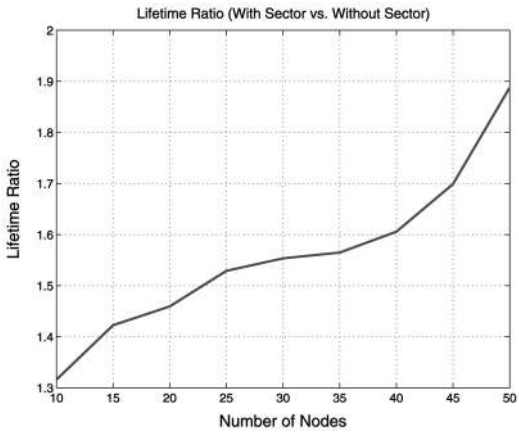

Fig. 12. Lifetime ratio of a cluster when divided into sectors versus when not divided into sectors.

7.1 Channel Assignment Problem

Note that, to see whether a request can be granted, we require no changes to the current schedule, which means that all transmissions that have been scheduled will stay in the frequency channel assigned to them. Fixing channels for scheduled transmissions will turn down some requests that can otherwise be granted. For example, suppose there are two channels, F_1 and F_2. Suppose in a time slot there are three transmissions in the schedule, $S_1' \rightarrow S_1$ and $S_2' \rightarrow S_2$ in F_1 and $S_3' \rightarrow S_3$ in F_2. Suppose the new transmission $S_4' \rightarrow S_4$ causes large interferences to $S_2' \rightarrow S_2$ in F_1 and to $S_3' \rightarrow S_3$ in F_2. If the current schedule cannot be changed, $S_4' \rightarrow S_4$ cannot be scheduled. However, if the current schedule can be changed and if $S_1' \rightarrow S_1$ and $S_4' \rightarrow S_4$ do not interfere with each other in F_1 and $S_2' \rightarrow S_2$ and $S_3' \rightarrow S_3$ do not interfere with each other in F_2, $S_4' \rightarrow S_4$ can be scheduled to F_1.

The reason for us to fix the channel assignment is its simplicity and the intractability of a decision problem we have to face otherwise.

Theorem 4. *Given a group of m transmissions and f channels, it is NP-complete to determine whether there is an assignment such that all transmissions can be scheduled in the same time slot.*

Proof. We will use the NP-complete *Vertex Coloring (VC) Problem*: Given a graph G, do there exist f colors such that every vertex in G can be given a color and no adjacent vertices are given the same color?

Given a graph G which is an instance of the VC problem, for each vertex v_i in G construct a transmission $S_i' \rightarrow S_i$. A group of transmissions is compatible in a frequency channel if and only if, among the corresponding vertices, no two vertices are adjacent in G. It is then clear that there exists an f-coloring in G if and only if the transmissions can be fitted into f channels. □

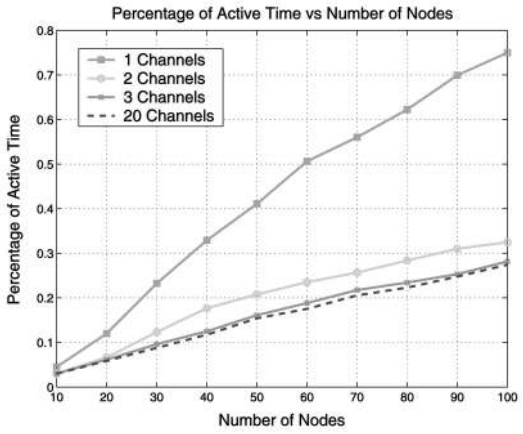

Fig. 13. Percentage of active time versus the number of nodes.

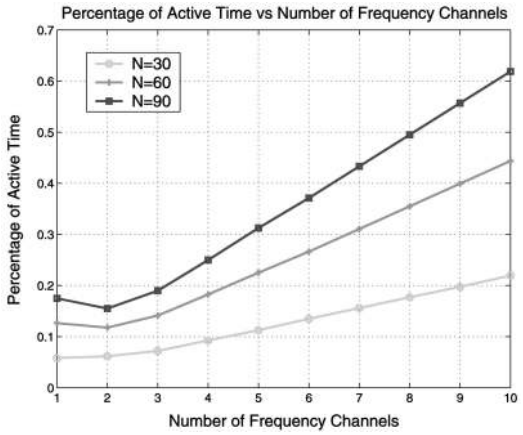

Fig. 14. Percentage of active time versus the number of frequency channels.

7.2 Simulation Results for Multichannel Polling

Fig. 13 shows the percentage of active time needed to ensure that all packets are received by the cluster head, where the number of sensors in a cluster ranges from 10 to 100 and the data generating rate is 50 Bps. The percentages of active time when there are 1, 2, 3, and 20 frequency channels are shown in the figure, where the bandwidth of each channel is 200 Kbps. Note that the percentage of active time is reduced by half when the number of channels increases from 1 to 2. However, when the number of channels keeps increasing to 3 and, finally, to 20, the percentage of active time has relatively little improvement. This is because, when the number of channel increases to 2, most cochannel interferences have been resolved. However, there is still *node conflict*, which means that transmissions cannot be scheduled in the same time slot if they are sending to or receiving from the same sensor. When there is more than one channel, node conflict becomes dominant that it cannot be resolved by adding more channels.

7.3 Optimal Number of Frequency Channels with Limited Frequency Resource

From the above discussions, we know that if the number of frequency channels keeps increasing, the percentage of active time can always be reduced. However, the total frequency resource that can be used by sensors may be limited in some situations. In other words, the product of the number of channels and the bandwidth of each channel should be fixed. In this scenario, we assume that the entire available frequency resource is equally divided into C channels and, for simplicity, we ignore the bandwidth of guard bands between frequency channels. Apparently, as the number of channels increases, the active time of sensors will likely decrease due to the reduction of interference. However, if the number of channels keeps increasing, the benefit of having more channels will decrease and, moreover, the time needed to send a packet will increase since the bandwidth of a channel is now small, which will cause the active time of sensors to increase. Therefore, it is worth studying the optimal trade-off between the number of channels and the bandwidth of a channel to achieve the minimum active time.

We assume that the total available frequency bandwidth for a cluster is 1 $Mbps$, which will be divided equally into a number of channels. To see the effect of the density of the nodes, 30, 60, and 90 sensors are deployed in the field. In Fig. 14, we can observe that the percentage of active time for 30 nodes increases monotonously as the number of frequency channels increases, whereas the curves for 60 and 90 nodes decrease a little at the beginning and achieve the minimum value when the number of frequency channels is 2. The reason for this is that, when sensors are deployed sparsely in the field, interference is not the dominant factor and, if the limited frequency resource is divided into C channels, at any time, only $\frac{1}{C}$ of the total bandwidth can be used by a one-hop transmission. Thus, the transmission is slowed down and the percentage of active time goes up. However, when sensors are densely deployed, interference becomes dominant; thus, there is some gain by dividing the frequency bandwidth into two channels. After that, node conflict becomes the major conflict in the cluster and increasing the number of channels cannot solve the conflict and will only slow down the transmission.

7.4 Multichannel Polling in Clusters with Sectors

Both using multiple channels and partitioning clusters into sectors can improve the lifetime of sensors. In fact, these two methods can be combined to achieve even better results. Fig. 15 shows the lifetime ratio of a cluster partitioned into sectors with two channels versus a cluster not partitioned into sectors with only one channel. It can be seen that the lifetime is greatly improved by using multiple channels and partitioning. For example, in a cluster with 30 sensors, the lifetime ratio is about 2.45. Comparing with Fig. 12, which shows the lifetime ratio of a cluster partitioned into sectors versus a cluster not partitioned into sectors when both use only one channel, the advantage of using multiple channels can also clearly be seen. For example, in a cluster with 30 sensors, the lifetime ratio in Fig. 12 is only about 1.55. Therefore, we expect both of these two methods be incorporated in sensor network design.

8 Conclusions

In this paper, we have studied two-layered heterogeneous sensor networks, where the network is partitioned into clusters and a powerful cluster head controls all sensors in a cluster. We mainly focused on the energy-efficient design

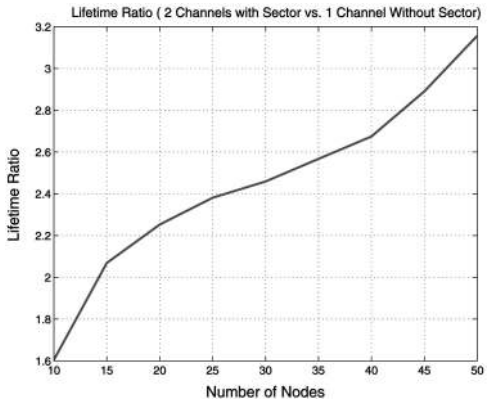

Fig. 15. Lifetime ratio of a cluster partitioned into sectors with two channels versus a cluster not partitioned into sectors with only one channel.

within a cluster to prolong network lifetime. We used polling to collect data from sensors instead of letting sensors send data randomly so that less energy is consumed. We showed that the problem of finding a contention-free polling schedule that uses the minimum time is NP-hard and then gave a fast online algorithm to solve it approximately. We also conducted simulations on the NS-2 simulator and the results show that our polling scheme achieves 100 percent throughput even when the sensor active time is not long. We have also introduced two other methods, namely, partitioning clusters into sectors and using multiple frequency channels, to further improve the network lifetime.

APPENDIX

In this appendix, we show that the MHP problem is still NP-hard when packets can be queued at intermediate sensors. We first consider the case when the number of packets in sensors is arbitrary.

Theorem 5. *The MHP problem is still NP-hard even when packets can be queued at intermediate sensors.*

Proof. It suffices to show that the following decision problem is NP-complete: Given a multihop cluster, does there exist a polling schedule by which all packets can be received before a given time slot when packets can be queued? We will use the NP-complete VC problem: Given a graph G with n vertices, do there exist k colors such that every vertex in G can be given a color and no adjacent vertices are given the same color?

Given any VC instance, we construct an instance of the MHP problem as follows: For each vertex v_i in $G, 1 \leq i \leq n$, construct a branch with two sensors $S_i' \rightarrow S_i \rightarrow t$. All such branches are called the *primary* branch. Transmissions $S_i \rightarrow t$ and $S_i' \rightarrow S_i$ will be referred to as a "first-level transmission" and a "second-level transmission," respectively. Transmission $S_i \rightarrow t$ is not compatible with any other transmissions. Transmission $S_i' \rightarrow S_i$ is compatible with $S_j' \rightarrow S_j$ if v_i and v_j are not adjacent in G. There are also k *auxiliary* branches with only one sensor each, denoted by $U_i \rightarrow t$ for $1 \leq i \leq k$. Transmission $U_i \rightarrow t$ is compatible with all transmissions. Each S_i' for $1 \leq i \leq n$ and each U_i for $1 \leq i \leq k$ has one packet and each S_i for $1 \leq i \leq n$ has zero packets. For example, the constructed instance for Fig. 16a is shown in Fig. 16b.

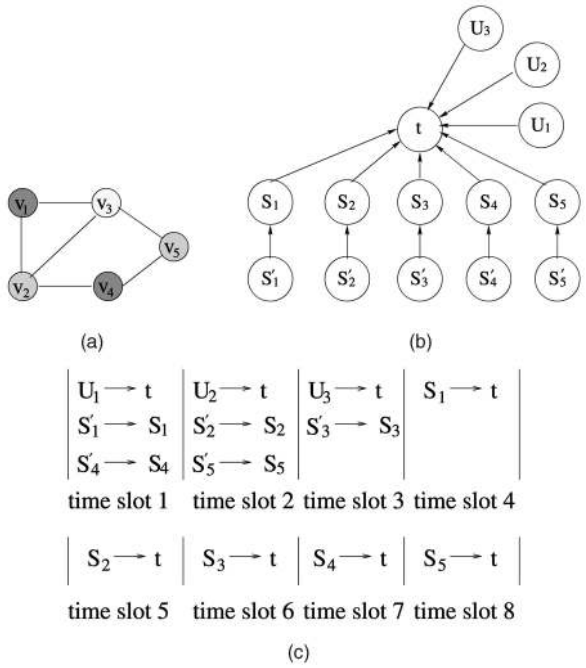

Fig. 16. (a) A graph with five vertices. It has a three-coloring. (b) The constructed MHP instance. (c) A polling schedule.

We claim that, if there is a schedule that finishes polling by time slot $n + k$ in the constructed instance, there is a k-coloring in G. To see this, note that, if such a schedule exists, in each time slot there must be a sensor sending a packet to the cluster head since there are a total of $n + k$ packets and the cluster head can receive at most one packet per time slot. Since first-level transmissions in the primary branches are not compatible with any other transmissions, second-level transmissions in the primary branches must have been scheduled to occur at the same time slot with some of the transmissions in the auxiliary branches, that is, be "fitted" into k time slots. Since second-level transmissions in the primary branches can be scheduled at the same time slot if and only if there is no edge between any of the corresponding vertices in the VC instance G, the schedule determines a k-VC G. For example, Fig. 16c shows a schedule with eight time slots in which the first three time slots determine the three-VC in Fig. 16a. On the contrary, it can also be easily verified that, if there is a k-coloring in G, there is a schedule that finishes polling by time slot $n + k$. This completes the proof. □

Similarly, it can be shown that:

Theorem 6. *The Exact One Packet Multihop Polling problem is NP-hard even when packets can be queued at intermediate sensors.*

ACKNOWLEDGMENTS

This research work was supported in part by US National Science Foundation Grants CCR-0207999 and ECS-0427345 and US Army Research Office Grant W911NF-04-1-0439.

REFERENCES

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-Level Measurements from an 802.11b Mesh Network," *Proc. ACM SIGCOMM,* 2004.

[2] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication.* Cambridge Univ. Press, 2005.

[3] A. Woo and D.E. Culler, "A Transmission Control Scheme for Media Access in Sensor Networks," *Proc. ACM MobiCom,* 2001.

[4] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocols for Wireless Microsensor Networks," *Proc. 33rd Ann. Hawaii Int'l Conf. System Sciences,* 2000.

[5] J.H. Chang and L. Tassiulas, "Energy Conserving Routing in Wireless Ad-Hoc Networks," *Proc. IEEE INFOCOM,* 2000.

[6] A. Amis, R. Prakash, D. Huynh, and T. Vuong, "Max-Min D-Cluster Formation in Wireless Ad Hoc Networks," *Proc. IEEE INFOCOM,* 2000.

[7] Y.-D. Lin and Y.-C. Hsu, "Multihop Cellular: A New Architecture for Wireless Comm.," *Proc. IEEE INFOCOM,* 2000.

[8] S. Banerjee and S. KhullerA, "Clustering Scheme for Hierarchical Control in Multi-Hop Wireless Networks," *Proc. IEEE INFOCOM,* 2001.

[9] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2002.

[10] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava, "Energy-Aware Wireless Microsensor Networks," *IEEE Signal Processing Magazine,* vol. 19, no. 2, pp. 40-50, 2002.

[11] V. Kawadia and P. Kumar, "Power Control and Clustering in Ad Hoc Networks," *Proc. IEEE INFOCOM,* 2003.

[12] J. Chou, D. Petrovic, and K. Ramchandran, "A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks," *Proc. IEEE INFOCOM,* 2003.

[13] A. Bogdanov, E. Maneva, and S. Riesenfeld, "Power-Aware Base Station Positioning for Sensor Networks," *Proc. IEEE INFOCOM,* 2004.

[14] O. Younis and S. Fahmy, "Distributed Clustering in Ad-Hoc Sensor Networks: A Hybrid, Energy-Efficient Approach," *Proc. IEEE INFOCOM,* 2004.

[15] D.B. West, *Introduction to Graph Theory.* Prentice Hall, 1996.

[16] X. Hong, M. Gerla, H. Wang, and L. Clare, "Load Balanced, Energy-Aware Communications for Mars Sensor Networks," *Proc. IEEE Aerospace Conf.,* 2002.

[17] W. Hu, N. Bulusu, and S. Jha, "A Communication Paradigm for Hybrid Sensor/Actuator Networks," *Proc. 15th Ann. IEEE Conf. Personal, Indoor and Mobile Radio Comm.,* 2004.

[18] P. Gupta and P.R. Kumar, "The Capacity of Wireless Networks," *IEEE Trans. Information Theory,* vol. 46, no. 2, pp. 388-404, 2000.

[19] C. Florens, M. Franceschetti, and R.J. McEliece, "Lower Bounds on Data Collection Time in Sensory Networks," *IEEE J. Selected Areas in Comm.,* vol. 22, no. 6, pp. 1110-1120, 2004.

[20] "The Network Simulator—NS-2," http://www.isi.edu/nsnam/ns/, 2007.

[21] M. Yavis, N. Kushalnagar, H. Singh, A. Rangarajan, Y. Liu, and S. Singh, "Exploiting Heterogeneity in Sensor Networks," *Proc. IEEE INFOCOM,* 2005.

[22] V.P. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, and N. Shroff, "A Minimum Cost Heterogeneous Sensor Network with a Lifetime Constraint," *IEEE Trans. Mobile Computing,* vol. 4, no. 1, pp. 4-15, Jan./Feb. 2005.

[23] S. Rhee et al., "Techniques for Minimizing Power Consumption in Low Data-Rate Wireless Sensor Networks," *Proc. IEEE Wireless Comm. and Networking Conf.,* 2004.

[24] S. Bandyopadhyay and E.J. Coyle, "An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2003.

[25] W.-P. Chen, J.C. Hou, and L. Sha, "Dynamic Clustering for Acoustic Target Tracking in Wireless Sensor Networks," *IEEE Trans. Mobile Computing,* vol. 3, no. 3, pp. 258-271, July-Sept. 2004.

[26] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," *Proc. ACM MobiCom,* 2000.

[27] Y. Xu, J. Heidemann, and D. Estrin, "Geography-Informed Energy Conservation for Ad Hoc Routing," *Proc. ACM MobiCom,* 2001.

[28] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks," *Proc. ACM MobiCom,* Sept. 2002.

[29] M. Bhardwaj and A. Chandrakasan, "Bounding the Lifetime of Sensor Networks via Optimal Role Assignments," *Proc. IEEE INFOCOM,* 2002.

[30] D.M. Blough and P. Santi, "Investigating Upper Bounds on Network Lifetime Extension for Cell-Based Energy Conservation Techniques in Stationary Ad Hoc Networks," *Proc. ACM MobiCom,* 2002.

[31] Y. Yu, B. Krishnamachari, and V.K. Prasanna, "Energy-Latency Tradeoffs for Data Gathering in Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2004.

[32] H. Li, P. Shenoy, and K. Ramamritham, "Scheduling Messages with Deadlines in Multi-Hop Real-Time Sensor Networks," *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symp.,* 2005.

[33] W. Liang and Y. Liu, "Online Data Gathering for Maximizing Network Lifetime in Sensor Networks," *IEEE Trans. Mobile Computing,* vol. 6, no. 1, pp. 2-11, Jan. 2007.

[34] M. Ma and Y. Yang, "Adaptive Triangular Deployment Algorithm for Unattended Mobile Sensor Networks," *IEEE Trans. Computers,* vol. 56, no. 7, pp. 946-958, July 2007.

[35] M. Ma and Y. Yang, "SenCar: An Energy Efficient Data Gathering Mechanism for Large Scale Multihop Sensor Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 10, Oct. 2007.

Zhenghao Zhang received the BEng and MS degrees in electrical engineering from Zhejiang University, China, in 1996 and 1999, respectively, and the PhD degree in electrical engineering from the State University of New York at Stony Brook, in 2006. From 1999 to 2001, he worked in the industry as an embedded system engineer. From 2006 to 2007, he worked as a postdoctoral research fellow in the Computer Science Department at Carnegie Mellon University. He is now an assistant professor in the Computer Science Department at Florida State University. His research interests include network security, computer systems, scheduling algorithm design, performance analysis, optical networking, and wireless networking. He is a member of the IEEE.

Ming Ma received the BEng degree in electrical engineering from the University of Science and Technology of China, Hefei, China, in 2002. Since then, he has been working toward the PhD degree in the Department of Electrical and Computer Engineering at the State University of New York at Stony Brook. His research interests include wireless sensor networks, wireless mesh networks, and wireless local area networks. He is a student member of the IEEE, the IEEE Computer Society, and the IEEE Communications Society.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a professor of computer engineering and computer science at the State University of New York at Stony Brook. Her research interests include wireless networks, optical networks, high-speed networks, and parallel and distributed computing systems. Her research has been supported by the US National Science Foundation and the US Army Research Office. She has published more than 180 papers in major journals and refereed conference proceedings and holds six US patents in these areas. She is currently an editor of the *IEEE Transactions on Computers* and the *Journal of Parallel and Distributed Computing* and was as an editor of the *IEEE Transactions on Parallel and Distributed Systems*. She has also served on program/organizing committees of numerous international conferences in her areas of research. She is a senior member of the IEEE and a member of the IEEE Computer Society.