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

Zhenghao Zhang, Ming Ma and Yuanyuan Yang

Department of Electrical and Computer Engineering

State University of New York, Stony Brook, NY 11794, USA

Abstract—**In this paper we study two-layered heterogeneous sensor networks where two types of nodes are deployed in the network: basic sensor nodes and cluster head nodes. Basic sensor nodes are simple and inexpensive, while cluster head nodes are much powerful and much richer in energy. A cluster head node organizes the basic sensor nodes around it into a cluster. A basic sensor node does data collections and sends the data packets when polled by the cluster head. By introducing hierarchy, such a two-layered heterogeneous sensor network has better scalability than homogeneous sensor networks. It also has a smaller overall cost since networking functionalities are shifted from sensors to the cluster head. It also has a longer life time, as sensors send packets only when polled by the cluster head and less energy is consumed in collisions and idle listening. This type of network will be ideally suited for applications such as environmental monitoring. In this paper, we focus on finding energy efficient and collision-free polling schedules in the multi-hop cluster. To reduce energy consumption in idle listening, a schedule is optimal if it uses minimum time. We show that the problem of finding an optimal schedule is NP-hard, and then give a fast on-line algorithm. We also consider dividing a cluster into sectors 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% throughput.**

Index Terms: Sensor networks, Clusters, Polling, Multi-hop polling.

I. Introduction and Background

The use of wireless sensors networks will enable a wide variety of applications, including environmental monitoring, medical treatment, emergency response, outer-space exploration, etc. In a sensor network, a large number of sensors are deployed over a large area, with each sensor capable of collecting data and sending data via wireless channels. To ensure reliable data transfer, sensors need to be organized into a robust multi-hop wireless network, which poses several challenges in the network design. First, the network must be scalable, since there can be thousands of sensors in the network. Second, to reduce the cost, sensors should be made as simple as possible, and as a result, sensors should only have some very simple functionalities. Third, the power supplies of sensors are limited and cannot be replaced, therefore the network must be energy efficient.

In general, these challenges cannot be easily met. We can tackle the first challenge by introducing hierarchies: the network is partitioned into clusters, and sensors need only to communicate within clusters while the inter-cluster communication is handled by cluster heads. [3] [13] gave protocols for cluster forming and cluster head selection. However, in these methods, sensors become more complicated since every sensor could be potentially elected as a cluster head and thus more transmitting and storage capabilities are needed for each sensor.

The research work was supported in part by NSF grant numbers CCR-0207999 and ECS-0427345 and ARO grant number W911NF-04-1-0439.

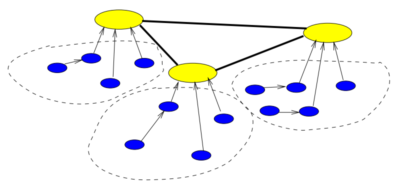


Fig. 1. A two-layered heterogeneous sensor network. The large nodes are the cluster heads. The small nodes are the basic sensor nodes.

We note that for some applications, for example, ground temperature monitoring, where the main job of sensors is to gather information and send it to outside observer, we can take the idea of clustering one step further by adding some powerful data gathering nodes to the network so that the sensors need not be burdened with the requirement of acting as the cluster head. We can deploy two different types of sensor nodes: basic sensor nodes and cluster head nodes, as shown in Fig.1. The basic sensor node is the majority of the sensor nodes, which is inexpensive and has relatively simple functionalities, such as data sampling and simple packet forwarding. It has limited power supply which cannot be replaced, thus its transmission power is small. The cluster head node, on the other hand, is much more powerful than basic nodes. It has much more computational capabilities, a far richer or replaceable power supply and a much larger transmission power. A cluster head node organizes basic sensor nodes around it into a cluster, and acts as the cluster head. The basic sensor node samples the data and sends it to its cluster head. The cluster head node then sends data to the outside observer or other nearby head nodes. Since the transmission power of a cluster head is large, the message sent by a cluster head can be received by all sensors in the cluster. On the other hand, since the transmission power of a basic sensor is small, the message sent by a basic sensor has to be relayed by other sensors to reach the cluster head. The advantage of such a heterogeneous network is that the majority of sensor nodes can be made very simple and inexpensive, thus the overall cost of the network can be greatly reduced. In the rest of the paper, we will refer to the basic sensor node as sensor, and the cluster head node as cluster head.

We can regard this type of network as a two-layered network. The first layer is the individual clusters. Above the first layer, in the second layer, the cluster heads can organize themselves into another network, in which they can exchange information or send data to the outside observer. The second layer is essentially a static wireless ad hoc network, and much work in the literature can be applied. The main issue remains is how to design the first layer in an energy-efficient way to prolong the life of the network.

Most commonly, it is assumed that sensors decide to send data packets based on their local information on the network. From the point of view of energy efficiency, this has several disadvantages. First, energy could be wasted in collisions: several sensors may decide to send packets at the same time and all these packets have to be retransmitted. Second, energy can be wasted by overhearing: sensors may need to decode packets not destined for them. Third, in an effort to avoid collision, sensors may have to use control packets to coordinate with each other, which also consumes a significant amount of energy. Lastly and most importantly, when sensors send packets in a random manner, a lot of energy will be wasted in idle listening. This is because sensors have to constantly monitor the radio channel for possible data packets destined for them, while the power consumption ratio for sleeping, idle listening, receiving and sending for typical sensors is $4.2 : 7.3 : 7.5 : 8.2$ [9], which indicates that idle listening still consumes more than 50% of the energy of other active operations.

To reduce energy consumption, we should reduce collision, overhearing and control messages, and set sensors to sleep mode as much as possible. For this purpose, [8] introduced MAC layer control called the S-MAC. However, S-MAC has no central controller and the energy spent in idle listening is still quite high. We note that the energy is wasted in S-MAC because of the lack of coordination among sensors: sensors do not know what other sensors will do and have to "be prepared" for most of the time. This is inevitable in a completely distributed system. However, in a heterogeneous network, with the presence of the cluster head, sensors within a cluster can be fully controlled by the cluster head and be fully coordinated, thus all possible sources of energy wasting mentioned above can be eliminated. The cluster head can send out messages to inform sensors to send packets that do not cause collision. The cluster head can also tell sensors when to receive a packet, thus no overhearing will occur. There will be no control packets from sensors. After enough data has been collected, the cluster head can tell all sensors to enter the sleep mode until next wake up time. Other advantages of this framework is that it needs less functionalities of the sensors. Such a locally-centralized framework is unconventional, but is not impossible since with some overhead, the cluster head can gather enough information about all sensors in its cluster. This type of network is ideally suited for data gathering applications where timing requirement is not strict, such as ground temperature monitoring.

We will give methods for the cluster head to organize the sensors in a robust and energy efficient way. The goal is to maximize the battery lives of sensors in the cluster. We use *polling* to collect data from the sensors, i.e., the cluster head sends a message to the sensor it wants to hear from, and only the sensor received this message will start sending. We will show that in a multi-hop cluster, the problem of completing polling in minimum time is NP-hard. We will then give a fast on-line approximation algorithm, which can also handle possible packet loss. Note that the difference of our algorithm from other polling protocols, such as 802.11 PCF and Bluetooth, is that the latter are for single hop networks while the former is for multi-hop networks.

In the past, the problem of partitioning sensors into clusters has been studied extensively, see, for example, [3] [13] [5]. These works considered homogeneous sensor networks and focused on how to partition networks into clusters, while we consider heterogeneous sensor networks and mainly focus on the operations inside a cluster. Sensor networks with two different types of nodes were considered in [11]. However, [11] focused on data compression techniques using the correlations of sensor data, not on the polling. Polling and data collection in sensor networks were studied in [15] [18], and both assumed that the central controller's polling message cannot be heard by all sensors and has to be relayed, while we assume the size of the cluster is modest and the cluster head has a relatively large transmission power, so that its polling message can be heard by all sensors in the cluster.

The rest of the paper is organized as follows. Section II describes how a cluster operates. Section III shows that completing polling in minimum time in a multi-hop cluster is NP-hard, and gives a fast on-line algorithm. Section IV explores the possibility of dividing a cluster into sectors to further reduce the idle listening time of sensors. Section V discusses some implementation issues. Section VI gives simulation results obtained by using the ns-2 simulator. Finally, Section VII concludes the paper.

II. Cluster Operations

Before moving on, we first describe how a cluster operates. In this and next two sections, we will assume that the network has been partitioned into clusters and focus only on the operations in an individual cluster. We target at the applications where the data generation rate is low. For such applications, sensors can sleep for most part of the time, and wake up only for a short period to send data. The time between two consecutive wake ups is referred to as a *cycle*. The time when sensors are active is referred to as a *duty cycle*.

In a duty cycle, the sensors work as follows. First, sensors wake up, and enter the active listening mode, at the time specified by the cluster head before they went to sleep in the last duty cycle. They will wait for a message broadcast by the cluster head, indicating the beginning of a duty cycle. After the cluster head broadcasts this message, each sensor sends a short packet back, acknowledging the cluster head, also informing the cluster head of the number of packets it needs to send in this duty cycle.

The cluster head will poll the sensors according to the information carried in the ack packets. It controls the sensors in a time slotted manner, where a time slot is the length of time for one data packet transmission. At the beginning of a time slot, the cluster head broadcasts a polling message, indicating which sensors can start to send their packets, also which sensors should receive packets. All other sensors will enter the idle listening mode for one time slot. Before the next time slot starts, all sensors will listen to the radio channel, waiting for the next polling message. After hearing the polling message, sensors that are polled send their packets, and sensors that had received a packet in the previous time slot will relay the packet to the next hop neighbor, and so on, until all packets have been received. After this, the cluster head broadcasts a message, sets all sensors to the sleep mode, and also informs them the next wake-up time. We can see that sensors work in a way similar to

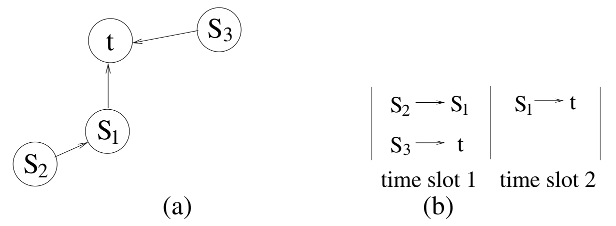


Fig. 2. (a). A simple cluster. S_1, S_2 and S_3 have 0, 1 and 1 packet to send, respectively. $S_2 \to S_1$ and $S_3 \to t$ do not cause collision, where t is the cluster head. (b). The optimal polling schedule.

a pipelined system, and the polling message acts as the clock. To reduce the length of a time slot, we do not use link level acknowledgment. If a packet is lost, the cluster head will poll the sensor again.

III. Polling in Multi-Hop Clusters

As mentioned earlier, the cluster head uses polling to get data from sensors. The major difference between single hop polling and multi-hop polling is that in multi-hop clusters, the cluster head can poll more than one sensors at the same time if their packet transmissions do not cause collision. As a simple example, consider the cluster shown in Fig.2(a). Suppose sensors S_1, S_2 and S_3 have 0, 1 and 1 packet to send, respectively. Because the packet from S_2 has to use two hops to reach the cluster head t, if only one sensor is polled at a time, i.e., a sensor is polled only after the packet from the sensor polled previously has been received by t, 3 time slots are needed. However, if transmissions $S_2 \to S_1$ and $S_3 \to t$ do not cause collision, the cluster head can poll both S_2 and S_3 at the first time slot. S_3's packet will be received by t at the first time slot. At the second time slot, S_1 relays the packet from S_2 to t. Thus the polling is completed in only 2 time slots, as shown in Fig.2(b).

To reduce energy consumption, we want to find a polling schedule that uses minimum time. Such a schedule is called an optimal polling schedule. We will show in this section that the problem of finding an optimal polling schedule is NP-hard, and then give a fast on-line approximation algorithm for it.

A. Relaying Paths

The first problem needs to be solved is to find paths for each sensor, by which its packet is relayed to the cluster head. 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 packets from other sensors it relays. The relaying paths for sensors are chosen such that the loads of sensors are balanced, i.e., there is no sensor relaying too many packets while other sensors relaying too few.

The problem can be formalized as a min-max problem: find a routing strategy such that the maximum load of sensors is minimized. [4] [12] showed that it can be solved in polynomial time by formalizing into a network flow problem, where either links or nodes have limited capacities.

The idea of the network flow formalization can be explained as follows. Suppose there are n sensors in the cluster and in a duty cycle each sensor has exactly one packet to send. We draw a directed graph G according to the connection patterns in the cluster. Let the cluster head be node t, or the *sink* of G. Each sensor, say, S_1, corresponds to two nodes s_1 and s'_1 in G, which are the "input" node and "output" node of S_1, respectively. There is an arc from s_1 to s'_1 with capacity δ, where δ is a positive integer. For any sensor that can hear the message from S_1, say, S_2, there is an arc from s'_1 to the input node of S_2, or s_2, with infinite capacity. Similarly, if cluster head can hear S_1, there is an arc from s'_1 to t with infinite capacity. We also add a source node s to G. There is an arc from s to s_i with unit capacity for all $1 \le i \le n$.

Note that if we run the Ford-Fulkerson algorithm on G and find a flow of value n, we have found for each sensor a relaying path to the cluster head, and, if sensors send packets along these paths, no sensor's load will exceed δ. Therefore, we can start with a small δ, for example, 1, then run the Ford-Fulkerson algorithm. If the value of the resulting maximum flow found by the algorithm is less than n, we can increment δ by one and run the algorithm again, until the value of the maximum flow is n. At this time δ is the maximum load of sensors. The running time of this algorithm is $O(n^3)$.

So far we have used the simplest case to explain the basic idea of the network flow formalization. By varying the link capacities from the source node to input nodes, this method can also be extended to the case when the number of packets generated by sensors are different. When the energy levels of sensors are different, the problem can be formalized as a network flow problem where the nodes, not the links, have capacity limitations. Also, note that to balance sensor load we need only to run the network flow algorithm once every long time period, by using the *average traffic intensity* of sensors which is the average number of packets generated by a sensor in a cycle. For details of the routing issue, the readers are referred to [4] [12].

From now on, we assume that the relaying paths have been found, and in one duty cycle, a sensor will send its packets along one fixed relaying path. For example, the relaying path for S_2 in Fig.2(a) is $S_2 \to S_1 \to t$. The *hop count* of a sensor is defined as the number of hops its packet has to travel before reaching the cluster head.

Note that to find the relaying paths, the cluster head needs to know which sensors a sensor can reliably communicate with, or the *connectivity patterns* of the cluster. We will describe methods to find the connectivity patterns in later sections. For now, we assume the cluster head has this knowledge.

B. Interference Patterns

To find a collision-free schedule, it is also crucial for the cluster head to know the *interference patterns* in the cluster. That is, the cluster head has to know whether a group of transmissions are contention-free, or *compatible*. [17] introduced two models for determining interferences in sensor networks: the protocol model and the physical model. Of the two, the protocol model is widely used. It assumes that sensor S_1 can receive packets from sensor S'_1 if S_1 lies in a disc centered at S'_1 with radius r, and two transmissions, say, $S'_1 \to S_1$ and $S'_2 \to S_2$, are compatible if and only if the distance between S'_1 and S_2 is more than $(1+\Delta)$ r, as well as between S'_2 and S_1, where Δ is a positive constant. This model is convenient for performance analysis, but we cannot use it in determining polling schedules for the following reasons.

First, in reality, the covering area of a sensor, which is the area in which other sensors can correctly receive messages from it, may very likely not be a disc. A number of factors, for example, obstacles and multi-path fading, can make the covering area

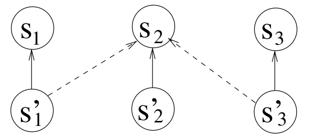


Fig. 3. Pairwise compatibility does not guarantee compatibility.

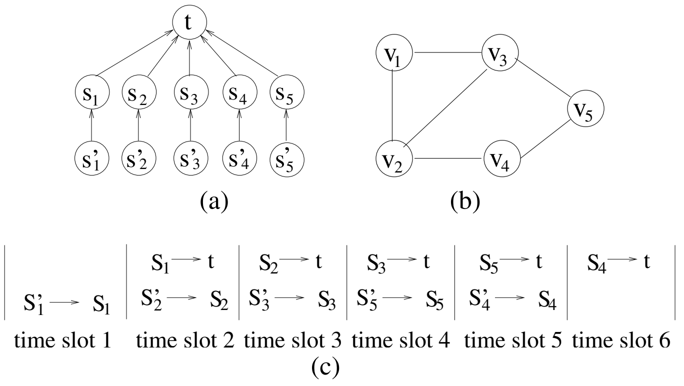

t
s_1 s_2 s_3 s_4 s_5
s'_1 s'_2 s'_3 s'_4 s'_5
(a)
v_1 v_3 v_5 v_2 v_4
(b)
$S'_1 \to S_1$
$S_1 \to t$ $S'_2 \to S_2$
$S_2 \to t$ $S'_3 \to S_3$
$S_3 \to t$ $S'_5 \to S_5$
$S_5 \to t$ $S'_4 \to S_4$
$S_4 \to t$
time slot 1
time slot 2
time slot 3
time slot 4
time slot 5
time slot 6
(c)

Fig. 4. (a) A TSRF with 5 branches. (b). An graph with 5 vertices. (c). A polling schedule for Fig.4(a) with interference patterns defined in Fig.4(b). This schedule corresponds to the Hamiltonian Path $v_1 \to v_2 \to v_3 \to v_5 \to v_4$.

very oddly shaped and might not even be convex, especially in urban areas [1]. The second reason is even more profound. The protocol model suggests that a group of transmissions are compatible if and only if they are pairwise compatible. However, in reality, a necessary condition for a group of transmissions being compatible is that the accumulated interferences at any receiver are not too large. For example, as in Fig.3, suppose three transmissions, $S'_1 \to S_1$, $S'_2 \to S_2$ and $S'_3 \to S_3$ are pairwise compatible. Let $P(S'_i, S_j)$ be the signal power from S'_i received by S_j when S'_i is sending packets. The three transmissions being pairwise compatible implies that any single $P(S'_i, S_j)$ is not large for $i \neq j$. However, when the three transmissions occur at the same time, if $P(S'_1, S_2) + P(S'_3, S_2)$ is too large, transmission $S'_2 \to S_2$ will still fail.

Neither can we use the other model, the physical model, in [17], because it assumes that the signal power decreases with distance strictly according to the power law, while the experiments in [1] showed that the signal power received at relatively long distance can be arbitrary. To make the network robust in any environment, in this paper we do not make any assumption about the covering areas and interference patterns and treat them as arbitrary. The cluster head obtains the connectivity patterns and interference patterns by testing the involved transmissions and sensors, and we will describe the method in detail later. Note that testing the interference patterns for all possible groups of transmissions needs exponential time and is impractical. Thus, in our algorithm we will assume that the cluster head only knows the compatibility of M transmissions, where M is a small positive integer, such as 2 or 3.

C. Optimal Polling Schedule

We now consider the problem of finding the optimal polling schedule, which is the schedule that finishes the polling in minimum time. We assume that with the acknowledgments collected at the beginning of a duty cycle, the cluster head knows the number of packets each sensor needs to send. The details of the acknowledgments collecting will be deferred to Section V.

We say a sensor is in level i if its hop count is i. If all sensors are in level 1, the scheduling reduces to single hop polling and is trivial. We call the problem when some sensors have levels more than 1 the *Multi-Hop Polling* problem and refer to it as *MHP* problem.

We will show that the MHP problem is NP-hard under virtually all scenarios, which would justify our use of a simple greedy algorithm for finding the polling schedule. To be specific, we will first show that it is NP-hard when packets are not delayed, i.e., a received packet is forwarded to the next hop immediately in the next time slot. We will then show that problem is still NP-hard when packets can be delayed. Note that in these two cases, the number of packets sensors need to send can be any non-negative integer. Finally we will show that even a special case of the MHP problem, in which each sensor has exactly one packet to send is also NP-hard.

C.1 Multi-Hop Polling Problem without Packet Delay

To see MHP problem is NP-hard, first we consider a special structure called "two level star with relaying only in the first level", which is referred to as *TSRF*. A TSRF is a tree, with the root of the tree being the cluster head. There are several branches connected to the root, where each branch consists of 2 sensors. Fig.4(a) shows a TSRF with 5 branches. More specifically, let t be the root. Sensors denoted by S_1, S_2, S_3, $\ldots$ are connected to t, or are in the first level, 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, etc. 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 the packet generated in sensor S'_i is $S'_i \to S_i \to t$. Some transmissions, say, $S'_1 \to S_1$ and $S_3 \to t$ can occur at the same time if they do not cause collision.

The problem is: Given a TSRF and its interference pattern, does there exist a schedule by which all sensors can send their packets to the cluster head by time k, where k is a given positive integer? We call it *TSRF Polling* problem and sometimes refer to it as *TSRFP*.

Lemma 1: The TSRF Polling problem is NP-complete when packet delay is not allowed.

Proof. It is clear that this problem belongs to NP. To see its NP-completeness, we use the well-known NP-complete *Hamiltonian Path* problem.

Given any instance of the undirected Hamiltonian Path problem, we construct an instance of TSRFP as follows. Denote the graph of the Hamiltonian Path 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.4(b) is Fig.4(a), where v_1 corresponds to branch $S'_1 \to S_1 \to t$, v_2 corresponds to branch $S'_2 - S \to 2 \to t$, etc. If two vertices are adjacent, for example, v_1 and v_2, transmissions $S_1 \to t$ and $S'_2 \to S_2$ are compatible, and transmissions $S_2 \to t$ and $S'_1 \to S_1$ are also compatible. 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. Sensor S'_i and S'_j can start in consecutive time slots if and only if $S_i \to t$ and $S'_j \to S_j$ do not cause collision. In this case, by our construction, there is an edge connecting v_i to v_j in G. Therefore, this schedule determines a Hamiltonian path in G. For example, as Fig.4(c) shows, a polling schedule using 6 time slots for the TSRF shown in Fig.4(a) determines a Hamiltonian path $v_1 \to v_2 \to v_3 \to v_5 \to v_4$ in Fig.4(b). Hence any algorithm that answers the TSRFP will also answer the Hamiltonian Path problem, i.e., the former is at least as hard as the latter. ■

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

Theorem 1: The Multi-Hop Polling problem is NP-hard when packet delay is not allowed.

Note that in the proof we constructed a TSRF with interference patterns resembling the connection patterns of an arbitrary graph. A natural question is whether the interference pattern in a TSRF can indeed be arbitrary.

We verify this with the following interference model, which is the physical model in [17] without the power law signal decrease assumption, as follows. We assume that in the TSRF, two transmissions, say, $S'_i \to S_i$ and $S_j \to t$, do not interfere with each other if and only the following two inequalities hold:

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

where γ is a positive constant. Note that as illustrated in [1], whether a packet can be successfully received or not is not completely determined by signal power. Thus, this model is still an approximation to reality. However, we still use it because it makes sense and also because of the lack of other good models.

In wireless environments, $P(S_j, S_i)$ can be arbitrary [1]. Thus, we can assume that $P(S_j,t)$ is larger than any $P(S'_i,t)\gamma$ for all i and j. We can let $P(S'_i,S_i) = \eta$ for all i. Then if there is an edge between v_i and v_j, we can let $P(S_j,S_i)\gamma < \eta$. Otherwise, we can let $P(S_j,S_i)\gamma\eta >$. This will produce the desired interferences.

C.2 Multi-Hop Polling Problem with Packet Delay

Not allowing packet delay makes the controlling simple, and also reduces the size of memory in sensors. Moreover, as we will soon show, the MHP problem is still NP-hard even if packet delay is allowed. Therefore there is no benefit for allowing packet delay.

Theorem 2: The Milti-Hop Polling problem is NP-hard even when packets can be delayed.

Proof. Again consider the TSRF. Suppose there is an algorithm that answers the question: Given a TSRF and interference pattern, does there exist a schedule by which all sensors can send their packets to the cluster head by time k, when allowing packet delay? Again we can let $k = n+1$, where n is the number of branches in the TSRF.

We claim that if there is a schedule by which all sensors' packets will have been received by the cluster head by time $n+1$, the same schedule must also be a legitimate schedule when packets are not allowed to be delayed. To see this, note that in this schedule, in every time slot after the first one, there must be a sensor in the first level relying a packet to the cluster head. Consider the first packet received by the cluster head. It must be received at the second time slot, and suppose it was from S_i. Then in the first time slot, S'_i must be sending this packet to S_i and therefore this packet did not experience any delay. Now consider the second packet received by the cluster head. This packet must be received at the third time slot, and suppose it was from S_j. This packet must be sent to S_j by S'_j at the second time slot, and thus is also not delayed. The same argument can be carried on, and as a result, no packet is delayed in this schedule. Hence, the same algorithm can be used to answer the TSRFP problem when packet delay is not allowed. ■

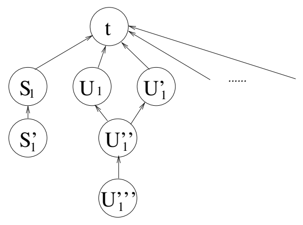


Fig. 5. The auxiliary branch for branch $S'_1 \to S_1 \to t$.

C.3 MHP Problem when Each Sensor has Exactly One Packet to Send

In some applications, sensors sample data periodically and generate exactly one packet per cycle. Now we consider a special case of the Multi-Hop Polling problem, in which each sensor has exactly one packet to send. We will call it *Exact One Packet Multi-Hop Polling* problem, and abbreviate it as *X1MHP* problem. This problem, unfortunately, is still NP-hard.

Theorem 3: The Exact One Packet Multi-Hop Polling problem is NP-hard when packets are not delayed.

Proof. We prove this by using the TSFRP. Given any instance of the TSFRP, we construct an instance of the X1MHP as follows.

For each branch in the TSFRP instance, say, $S'_1 \to S_1 \to t$, we add another auxiliary branch, as shown in Fig.5. The relaying path for U'''_1 is $U'''_1 \to U''_1 \to U'_1 \to t$. The relaying path for U''_1 is $U''_1 \to U_1 \to t$. U'_1 and U_1 send their packets directly to t. Transmission $U''_1 \to U'_1$ is compatible with $S_1 \to t$. Other transmissions in this auxiliary branch are not compatible with any other transmissions.

For any optimal solution to the X1MHP problem, suppose sensor U'''_1 start 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 $U''_1 \to U'_1$ is compatible with $S_1 \to t$, and the resulting schedule should still be optimal. After pairing them up, we move them to the beginning of the schedule, that is, we 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 $U'''_1 \to U''_1$ and $U'_1 \to t$ 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, we can also move U''_1, U'_1 and U_1 to the front, or to let them start to send at time slots 4, 6 and 7, respectively. Then we can pair up S_2 with its auxiliary branch and move them to the beginning of the schedule, then S_3, S_4 and so on.

As a result, the optimal schedule will consist of two parts. The first part is from time slot 1 to time slot $7n$, where n is the number of branches in the TSRF, which is for sensor S_1, S_2, ..., 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. ■

Using similar arguments as in Theorem 2, it can be shown that

Theorem 4: The Exact One Packet Multi-Hop Polling problem is NP-hard even when packets can be delayed.

To generate interference pattern in the cluster used in the proof of the above theorems, we can first let the received power for transmissions in the auxiliary branches be very weak. Consider the first auxiliary branch. To make transmissions $U_1 \to t$ and $U_1' \to t$ incompatible with all other transmissions, we need only to raise $P(\$,\ )$ for all sensors S not in the first level. To make $U_1''' \to U_1''$ and $U_1'' \to U_1$ not compatible with any other transmissions, we can raise $P(\$U,\ _1'')$ and $P(\$U,\ _1)$ for all sensors S. To make $U_1'' \to U_1'$ compatible only with $S_1 \to t$, we can let $P(\$U,\ _1')$ be relatively large for all sensors other than S_1.

D. *On-Line Polling Algorithm*

In this subsection we focus on finding an algorithm that gives sub-optimal contention-free polling schedules. Note that another requirement of this algorithm is that it should be able to be run *on-line*, since it needs to deal with packet loss, as success packet delivery is not guaranteed in wireless communications. Since the cluster head knows which time slot a sensor starts to send its packet and the hop count from the sensor to it, it knows exactly which time slot it should expect to receive that packet. Therefore, the cluster head can find out when a packet is lost and once this occurs, it can simply poll the sensor again. This is why the scheduling algorithm should be on-line, 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 on-line requirement left us no other choices but to adopt a very simple algorithm described in Table 1. We refer each packet as a *polling request*, or simply a *request*. Initially, each request is active. When a request has been added to the schedule, it becomes idle. At the time slot when the packet should have been received by the cluster head, if it is not received, the request will become active again. Otherwise, it will be deleted. The algorithm will only consider active polling requests. Since the cluster head only knows all possible combinations of no more than M transmissions, at any time, the algorithm will allow at most M concurrent transmissions.

Before a time slot, the algorithm finds the schedule only for this time slot. Before the first time slot, the schedule is empty, and the algorithm tries to find a group of requests that can start at the first time slot. It will scan through the requests according to an arbitrarily predetermined order, and add a request to the schedule if it does not cause contention with the existing ones, if started at the first time slot. After a maximal number of requests have been added to the schedule, or after M requests have been added, the algorithm halts. The cluster head will tell sensors in the schedule to send their packets at the first time slot. Before the second time slot, the algorithm will find requests that can start at the second time slot without causing collision with the requests already in the schedule, and so on, until all packets have been received.

To see whether a packet can be sent at time slot $\alpha + 1$, suppose the hop count is c. Then we need to check whether the c transmissions involved will collide with the transmissions in the existing schedule from time slots $\alpha + 1$ to $\alpha + c$. Suppose there are m requests in the schedule. For each time slot, there are totally $\sum_{i=1}^{m} \begin{pmatrix} m \\ i \end{pmatrix} = 2^m - 1$ possible groups of transmissions that need to be checked. Note that $m \leq M - 1$, thus it would require $O(c2^M)$ time. Suppose there are totally N requests and the maximum hop count is C. The work in each time slot can be done in $O(NC2^M)$ time. Note that although there is an exponential term, 2^M, in the bound, M is a fixed small integer. Thus the algorithm is still a linear time algorithm to the input size.

TABLE 1
POLLING ALGORITHM

```
Input : The polling requests.
Output: Polling schedule.

while requests are not all deleted
    if a packet is received
        Delete the request for this packet
    end if
    if a packet that is expected to arrive is not received
        Set the request for this packet to active.
    end if
    while ≤ $M$ requests in the schedule at this time slot
        Add an active request to the schedule to
        current time slot if it does not cause collision.
        Mark the request as idle.
    end while
    wait until next time slot.
end while
```

E. *Joint Routing and Scheduling*

So far, in this section, we have considered the scheduling problem when the relaying paths of sensors have been given. Sometimes we may want to find the optimal solution while not fixing the routing paths. This will give us more flexibilities and possibly better solutions. However, without much difficulty, next we will show that this problem is also NP-hard. This is why we choose to break the bigger problem into two sub problems, the routing problem and the scheduling problem, and solve them one by one.

Define the *polling time* of a cluster as the time needed for the cluster head to finish the polling. The life of a sensor is determined by its transmission load and the polling time of the cluster. We can assume the life of 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 and T is the polling time, and α and β are constants. The problem is: Given the connection pattern and the interference pattern of a cluster, find relaying paths and a polling schedule such that the maximum power consumption rate is minimized. We call it the *Joint Multi-Hop Routing and Polling* problem, or the *JMHRP* problem. JMHRP is clearly also NP-hard since any algorithm that solves it can also give a solution to the TSFRP. This is because that in a TSFR there is no routing problem involved and the JMHRP reduces to TSFRP.

IV. DIVIDING CLUSTER INTO SECTORS

Note that during the data transmission period, if a sensor will not be involved in transmissions occurred later, it can enter the sleep mode immediately to save energy. A sensor can enter the sleep mode in this way if and only if all the packets generated by itself and all the packets it needs to relay have been correctly

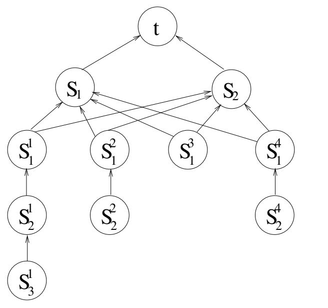


Fig. 6. The construction of the CPAR problem for set $\{3, 2, 1, 2\}$.

received by the cluster head. Therefore, if we want to set a sensor to sleep mode sooner, we can poll it and its dependents first, where a dependent of sensor S is a sensor whose relaying path involves S. Also note that by the time a sensor can enter the sleep mode, all its dependents can also enter the sleep mode.

These observations lead us to consider dividing a cluster into a number of sectors, where a sector consists of a group of sensors and each sensor has a relaying path to the cluster head. We can let each sector wake up and do data transmission in turn. By doing so, the wake up time of sensors can be greatly reduced. However, the maximum load of sensors may increase. Thus we have to divide clusters carefully such that the power consumption rates of sensors can be decreased. Note that sectors can be considered as small clusters, and its routing and polling mechanisms are the same as for clusters described in previous sections.

Managing sensors by sectors has yet another advantage. The number of sensors in a sector can be far less than the number of sensors in the entire cluster. This makes finding and storing the interference patterns much simpler, since far less number of groups of transmissions need to be tested and stored. For example, if we divide a cluster with 80 sensors into 8 sectors with each sector having 10 sensors, for $M = 3$, we need to test 1320 groups, which is far less than 85320 otherwise.

A. Optimal Sector Partition is NP-hard

Question still remains as to how to optimally partition a cluster into sectors. Since our primary goal is to prolong sensor life, we may want to minimize the maximum power consumption rate of sensors. However, finding the optimal partition is NP-hard by this criterion since finding the optimal polling schedule is NP-hard. Another criterion that makes sense is: We say a partition is optimal if the maximum *pseudo power consumption rate* is minimum. The pseudo power consumption rate of S_i is defined as $\alpha R_i + \beta' L_i$, where R_i is the load of S_i and L_i 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.

Problem: Given a cluster and its connection patterns, does there exist a partition such that
$\max\{\alpha R_i + \beta' L_i\} \leq \Theta$ for a given Θ?

We call it *Cluster Partition* problem and refer to it as *CPAR*.

Theorem 5: The Cluster Partition problem is NP-complete.

Proof. The CPAR problem clearly belongs to NP. To see its NP-completeness, we use *Partition* problem, in which a set of positive integers $\{B_1, B_2, \dots\}$ are given, and we partition the integers into two subsets where the sum of the numbers in two subsets should be equal.

Given any instance of the Partition problem, we construct an instance of the CPAR problem as follows. We draw two sensor nodes, S_1 and S_2, which are connected to the cluster head t. For the i_{th} integer, B_i, in the set, we draw B_i sensors, denoted as S_1^i to S_δ^i, where $\delta = B_i$. These δ sensors are a branch, with S_j^i connected only to S_{j-1}^i for $1 < j \leq \delta$. S_1^i is connected to both S_1 and S_2. Θ is set to be $(n/)2(+1 \quad \alpha + \beta')$, where $n = \sum B_i$. For example, the construction for set $\{3, 2, 1, 2\}$ is shown in Fig.6.

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 so, the pseudo power 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 2 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. For example, for the cluster shown in Fig.6, we can let the first and the third branch be in the same sector as S_1 and let the second and the fourth branch be in the same sector as S_2, which is to partition set $\{3, 2, 1, 2\}$ into $\{3, 1\}$ and $\{2, 2\}$. ■

B. Heuristic for Sector Partition

We now give a heuristic for partitioning a cluster into sectors. Since the duty cycle is short as compared to the length of a cycle, we do not need to set a limit to the number of sectors in a cluster. Due to limited space we only outline the ideas of the heuristic here.

We call a sensor that can directly communicate with the cluster head the first level sensor. 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 first level sensor and its dependents. By so doing, the load of sensors remains the same, and the numbers of sensors in the sectors are likely to be evenly distributed.

When the union of the optimal relaying paths is not a tree, which is almost surely the case, we will first try to make it a tree. Note that there must be some flow splitting sensors, i.e., 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 minimum. To make sure that there are no flow splitting sensors along this path, we start flow merging at flow s-plitting sensors closest to the cluster head. Eventually, the union of the relaying paths will become a tree.

However, we cannot simply let a first level branch of this tree be a sector, because at this time it is very unlikely that the load of first level sensors are evenly distributed. To remedy this, we can combine two first level branches into one sector. There are three rules for pairing up branches. First, there should be connections between the two branches such that some traffic can be redirected to the first level sensor with less load. Second, a branch with more sensors should be paired up with a branch with fewer sensors. Third, let two first level sensors be S_1 and

S_2, when S_1 is sending a packet to the cluster head, S_2 should be able to receive a packet from other sensors, and vice versa, so that the polling can be done more efficiently. Note that we assume sensors are simple and cannot receive and send at the same time. In a sector with only one first level sensor, say, S_1, S_1 has to receive a packet in the first time slot and then sends the packet to the cluster head in the second time slot. In a sector with two first level sensors, if the third rule is satisfied, S_1 and S_2 can send packets to the cluster head in consecutive time slots, and the polling time will not increase much as compared to sectors with only one first level sensor.

V. Implementation Issues

In this section, we discuss some implementation issues in the two-layered heterogeneous sensor networks.

A. Cluster Forming

Initially, the sensor network should be partitioned into clusters. The cluster head should know which sensors are in its cluster and sensors should also know which clusters they belong to. This is *Cluster Forming*, and is a research topic of its own [5] [3] [13].

In this paper we assume that the clusters have been formed and focus on in-cluster controls. However, one possible way of cluster forming is to let cluster heads compute the Voronoi diagrams and let sensors in the same Voronoi cell belong to the same cluster. This requires sensors to know their locations, which is a legitimate assumption since for many applications, information will be meaningless without knowing the locations of where they come from.

After cluster has formed, the cluster head should know which sensors belong to its cluster. We can first let the cluster head discover sensors that can directly communicate with it, then let these sensors find sensors that are two hops away, then use the new sensors to find sensors three hops away, until no new sensors are discovered. Each sensor can remember the first sensor that discovered it as its parent, who will be in charge of forwarding its packets to the cluster head. Note that this is only for setting up a temporary data relaying path.

B. Knowing the Connectivity

The cluster head should first find out the connectivity patterns in its cluster. We can let sensors broadcast in turn, then poll each sensor to see which sensor it has heard from. This will need $O(n)$ time where n is the number of sensors, and is only needed at the initialization phase. After the connectivity pattern is known the cluster head can find the optimal relaying path using the network flow algorithms.

C. Source Routing

To the best of our knowledge, all previous works on load balanced routing focused on the theoretical aspects of the problem, that is, how to find the optimal relaying paths. Practically, after finding the optimal relaying paths, we should find a way to make traffic move along it. This problem was realized in [12] but not solved. A possible solution to this is to use *source routing*: Each sensor remembers its relaying path and adds it to the header of the packet it sends. The sensor that receives this packet will then relay it to the sensor that appears next in the relaying path. This will ensure that traffic will flow according to the optimal relaying paths. However, source routing will also add length to the data packets and waste energy. Equivalently, we can let each sensor remember a one-hop routing table for all its dependents. Since only one hop information is needed for each dependent, this will not need too much storage space.

D. Multiple Paths Rotation

Note that a sensor may have several relaying paths. For example, if on average 3 packets are generated by sensor S_1 in a cycle, the maximum flow algorithm may find two paths, with path 1 carrying 2 units of flow and path 2 carrying 1 unit. To simplify the control, in a duty cycle, we would like sensors to send packets only on one path. Thus, to balance the load, we can let sensors send packets on these paths alternatively, in proportion to the units of flows the paths carry. In this example, we can let S_1 send packets along path 1 for two duty cycles and along path 2 for one duty cycle.

E. Knowing the Interference Pattern

After the optimal routing path is found, to determine the polling schedule, the cluster head needs to know the interference pattern. As mentioned earlier, we can do so by testing each group of no more than M transmissions in the optimal relaying paths. We can poll each pair of sensors involved in the transmissions at the same time, then poll the sensors that are supposed to receive the packets in turn. This can be done in $O(n^M)$ time.

F. Acknowledgments Collecting

As discussed earlier, after the cluster head has received all the packets, sensors enter the sleep mode. Ideally, later, all sensors should wake up at exactly the same time. However, due to possible drift of clocks in sensors, this might not be the case. Therefore, to make sure that all sensors have waken up before data transmission, the cluster head should broadcast an inquiry message. All sensors receiving this message should then reply an acknowledgment. Only after receiving acknowledgments from all sensors will the cluster head start polling. In addition, along with this acknowledgment message, a sensor can inform the cluster head of the number of packets it intends to send.

One possible way of doing this is to let cluster head poll every sensor. However, note that since the network is multi-hop, some sensors will have to relay the ack packet for other sensors, and while relaying, sensors can add in their own ack to this packet. Thus, actually only one ack packet is needed for sensors along a path to the cluster head, and as a result, only the sensor at the beginning of the path needs to be polled.

Using similar arguments to that in Section III-E, we can show that problem of completing the acknowledgments collecting process in minimum time is also NP-hard. To solve it, we can break it into two subproblems: first, find a set of paths that covers all sensors; then, find a schedule to finish polling the sensors at the beginning of the paths in minimum time. The second problem can be solved by the algorithm for multi-hop polling described earlier.

To solve the first subproblem, we use the relaying paths for the data packets as "candidate" paths, and choose among them a set of paths that cover all sensors with minimum total hop counts. This problem can be solved by regarding the sensors as elements in a whole set, and regarding the paths as subsets. Each subset has a cost equal to its hop count. Then find a group of subsets that cover the whole set with minimum total cost. This problem is exactly the *Weighted Set Cover Problem*, and is also NP-hard, but a fast greedy algorithm can be used to give sub-optimal solutions. This greedy algorithm iteratively chooses a subset that has the minimum covering cost, where covering cost of a subset is defined as the ratio of the cost of the subset divided by the number of uncovered elements it contains.

G. Removing Inter-Cluster 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, if their cluster heads decide to poll them at the same time. This is called the inter-cluster interference.

The simplest way to remove inter-cluster interference is to allow data transmission in only one cluster at a time. A token can be rotated among the cluster heads, and only the cluster head that captures the token can start data transmission in its cluster. It works for the case when the number of clusters are not large, and 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 adjacent clusters different colors. This is the planar graph coloring problem and we know that 4 colors, or 4 radio channels will be sufficient. To find a coloring, a cluster head can be elected to run a centralized algorithm for the coloring problem. There exists a simple algorithm that uses at most 6 colors, using the property that in a planar graph, there must be a vertex with degree no more than 5 [14].

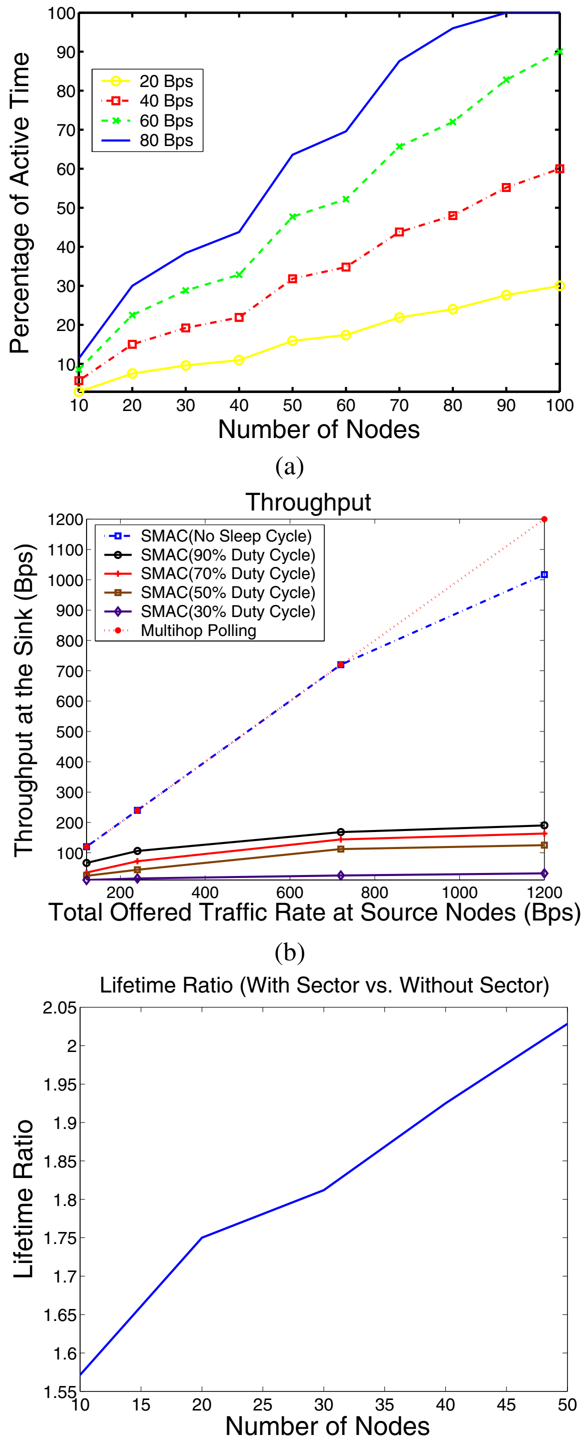
VI. Simulation Results

In this section we evaluate the performance of the proposed polling scheme. We have implemented the proposed algorithm based on the NS-2 simulator. We assume that all sensor nodes are uniformly deployed within a $200 \times 200m^2$ two-dimensional square. The cluster head is placed at the center of the square. Two-ray propagation model is used to describe the feature of the physical layer. With the maximum transmission power $0.858mw$, each node can communicate with other nodes as far as $40m$ away. The radio bandwidth is 200kbps. CBR traffic on the 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 1000 seconds which contains 100 seconds warming-up period. All simulation data are collected from 100 seconds to 1000 seconds.

A. Percentage of Active Time

The major goal of our polling scheme is to reduce the active time of sensors. In Fig. 7(a) we plot 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 data generating rate ranges from 10 to 80 Bps. We can see that for a cluster with 30 sensors, when the data generating rate is 60 Bps, sensors need only to be active for about 30% of the time.

We can also see that when the number of sensors increases, or when data generating rate increases, the active time of sen-



(c)

Fig. 7. (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. (c). Life time ratio of a cluster when divided into sectors v.s. no sector.

sors will also increase to ensure packet delivery. We notice that for a given data generating rate, for example, 80 Bps, when the number of nodes increases to 90, all sensor in the cluster must be active for *all* time. This implies that there is a maximum size for a cluster under a certain data generating rate, and above this threshold, packets will be lost. Thus we should choose a suitable size for a cluster so that no packets are lost while sensors can also enjoy long sleeping time.

B. Comparison with SMAC

To show the effectiveness of the proposed polling scheme, we compare its performance with an energy efficient MAC protocol, named SMAC [8]. Similar to our polling scheme, SMAC also allows each sensor 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 MAC layer, to find the relaying path for each sensor, we use 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. 7(b) shows the throughput of a cluster with 30 sensors, when the total offered load is 210, 750 and 1200 Bps (the data generating rates at individual sensors are 7, 25 and 40 Bps, respectively). We can see that under all offered loads, our polling scheme achieves 100% throughput, i.e., all packets generated by sensors are correctly received by the cluster head. Note that the percentage of active time of sensors under the polling scheme is less than 20%, as can be derived from Fig. 7(a). We also measured the throughput of SMAC+AODV, when active time of sensors are configured as 30% , 50%, 70%, 90% and 100%. These ratios are all higher than 20% which is the active time of sensors under the polling scheme. Quite surprisingly, the throughput of SMAC+AODV is far less than the total offered load when the active time is not 100%. Thus, we can say that our polling scheme has much better throughput performance than SMAC even with much less sensor active time. We believe one of the major reasons for this is that in SMAC+AODV, a lot of control packets were generated for routing, since sensors must frequently use AODV to find relaying paths as the path used previously may not be valid any more if some sensors have entered the sleep 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 according to the back-off algorithm. 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% when the total offered traffic rate becomes as high as 1200 Bps.

C. Effects of Dividing a Cluster into Sectors

As Section IV shows, by dividing a cluster into sectors, sensor life time can be further prolonged. We studied the life time of a cluster both when divided into sectors and not divided into sectors while sustaining 100% throughput, and showed the life time ratio of the former over the latter in Fig. 7(c). We can see that the ratio is always larger than 1, which means that by dividing a cluster into sectors, sensor life time will always increase. Also, because usually larger clusters can be divided into more sectors, the increase of life time is more for larger clusters.

VII. 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 each cluster. We mainly focused on energy efficient design of clusters to prolong network life. We used polling to get data from sensors instead of letting sensors send data randomly so that less energy is consumed. We have showed that the problem of finding a contention-free polling schedule that uses minimum time is NP-hard, and then gave a fast on-line algorithm. We also conducted simulations on the NS-2 simulator, and the results show that our polling scheme achieves 100% throughput even when the ratio of active time of sensors is very low.

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd and R. Morris, “Link-level measurements from an 802.11b mesh network,” *in Proceedings of ACM SIGCOMM 2004*, Portland, Aug. 2004.

[2] A. Woo and D. E. Culler, “A transmission control scheme for media access in sensor networks,” *in Proceedings of ACM MOBICOM 2001*, Rome, Italy, June 2001.

[3] W. Rabiner, A. Chandrakasan and H. Balakrishnan, “Energy efficient communication protocols for wireless microsensor networks,” *in Proceedings of the Hawaii International Conference on System Sciences*, Jan. 2000.

[4] J.H. Chang and L. Tassiulas, “Energy conserving routing in wireless adhoc networks,” *in Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.

[5] A. Amis, R. Prakash, D. Huynh and T. Vuong, “Max-min D-cluster formation in wireless ad hoc networks,” *in Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.

[6] Y.-D. Lin and Y.-C. Hsu, ”Multihop cellular: a new architecture for wireless communications,” *in Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.

[7] S. Banerjee and S. KhullerA “Clustering scheme for hierarchical control in multi-hop wireless networks,” *in Proceedings of IEEE INFOCOM 2001*, Anchorage, April 2001.

[8] W. Ye, J. Heidemann and D. Estrin, “An energy-efficient MAC protocol for wireless sensor networks,” *in Proceedings of IEEE INFOCOM 2002*, New York, June 2002.

[9] 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, March 2002.

[10] V. Kawadia and P. Kumar, “Power control and clustering in ad hoc networks,” *in Proceedings of IEEE INFOCOM 2003*, San Francisco, April 2003.

[11] J. Chou, D. Petrovic and K. Ramchandran, “A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks,” *in Proceedings of IEEE INFOCOM 2003*, San Francisco, April 2003.

[12] A. Bogdanov, E. Maneva and S. Riesenfeld, “Power-aware base station positioning for sensor networks,” *in Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[13] O. Younis and S. Fahmy, “Distributed clustering in ad-hoc sensor networks: a hybrid, energy-efficient approach,” *in Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[14] D. B. West, “Introduction to graph theory,” *Prentice-Hall*, 1996.

[15] X. Hong, M. Gerla, H. Wang and L. Clare, “Load balanced, energy-Aware communications for Mars sensor networks,” *In Proceedings of IEEE Aerospace 2002*, Bigsky, Mar. 2002.

[16] W. Hu, N. Bulusu and S. Jha, ”A Communication paradigm for hybrid sensor/actuator networks,” *To appear in Proceedings of the 15th Annual IEEE Conference on Personal, Indoor and Mobile Radio Communication (PIMRC 2004)*, Barcelona, Spain, Dec 2004.

[17] P. Gupta and P. R. Kumar, “The capacity of wireless networks,” *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388-404, March 2000.

[18] C. Florens, M. Franceschetti and R. J. McEliece, “Lower bounds on data collection time in sensory networks,” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 6, pp. 1110-1120, Aug. 2004.

[19] “The Network Simulator - NS-2, http://www.isi.edu/nsnam/ns/”.