Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Microprocessors and Microsystems 28 (2004) 467–475
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

Energy equivalence routing in wireless sensor networks

Wei Ding[a,*], S.S. Iyengar[a], Rajgopal Kannan[a], William Rummler[b]

[a]Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge LA 70803, USA
[b]Department of Computer Science, Rochester Institute of Technology, USA

Received 19 August 2003; revised 28 April 2004; accepted 12 May 2004

Available online 4 June 2004

Abstract

Energy is a critical resource in wireless sensor networks. In this paper, we propose a new approach to maintain network wide energy equivalence and maximize network lifetime. Compared to existing protocols, our approach emphasizes on route maintenance instead of route finding. This means no critical nodes would become the bottleneck of network lifetime. A reroute request packet is sent out from sinks periodically. When the packet reaches a path node, Common Neighbor Switching (CNS) algorithm checks energy difference between the node and its neighbors outside the routing tree. If the difference goes beyond a threshold, double neighbor switching is performed. Two path-rerouting algorithms, namely, Shortest Rerouting (EERS) and Longest Rerouting (EERL), are also presented to show that neighbor switching is better than path rerouting. Simulation results show that CNS outperforms Directed Diffusion in more than 90% cases, while EERS and EERL show only blurry and conditional advantage over directed diffusion.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Energy equivalence routing; Neighbor switching; Path rerouting; Directed diffusion

1. Introduction

Based upon new advances in VLSI, MEMS, wireless communication, and distributed computing, wireless sensor networks (WSNs) have boomed in past few years [1]. The untethered and unattended nature of WSNs destines energy sources of most sensors to be scanty and impossible to replenish. Energy efficiency and network longevity have dominated WSN design and have occupied a large portion of research effort. Particularly while designing of routing protocols, energy saving is an overwhelming consideration.

When all sensors have equal initial energy and equal chances to become sources and sink, network could maximize its lifetime if all sensors dissipate energy at the same rate, since no loss of connectivity would result from node failure. In this paper, we propose a new method to balance energy consumption and keep approximate network wide energy equivalence by replacing heavily dissipated nodes with their unused or less used neighbors. Many existing protocols in network layer offer solutions that set up optimal routes, but they do not keep routes optimal. Maintaining established routes is a common practice. On the contrary, this paper concentrates on how to readjust established routes to balance energy dissipation. The basic heuristic is to periodically adjust energy dissipation in as small a range as possible. In WSN, smallest topological range is neighborhood, so we concentrated on neighbor switching. Our approach may be used with any route finding protocols, as long as result is a reverse multicast tree. We use Directed Diffusion (DD) in our approach.

2. Related works

Many routing protocols have been designed to enhance energy efficiency and prolong network lifetime in WSNs. At the beginning on-demand protocols for mobile ad hoc networks (MANET) were adopted, but they produce poor results. They are inherently incompatible with WSN in the following aspects [1]. First, WSNs are much larger and have much higher density than MANET. Second, WSNs often uses many-to-one communication model with the topology of a reverse multicast tree. Furthermore, table-driven MANET protocols require too much memory to store

* Corresponding author. Address: Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge LA 70803, USA. Tel.: +1-225-5781249; fax: +1-225-5781465.

E-mail addresses: wding1@lsu.edu (W. Ding), iyengar@bit.csc.lsu.edu (S.S. Iyengar), rkannan@bit.csc.lsu.edu (R. Kannan), war5549@rit.edu (W. Rummler).

routing tables, which WSN cannot afford. Hence, research focus was shifted to tailored protocols for WSNs.

Flooding is natural for multi-hop communication, but it consumes too much energy. Many variants of flooding have been designed as routing protocols for WSN. For example, GRAdient Broadcast (GRAB) [2,3] sets up cost field by flooding. Gossip [2,4] exercises a partial probabilistic flooding to diffuse interests or events. Well-known Directed Diffusion [2,5] uses limited flooding and an acknowledgement scheme to set up route. Data aggregation is integrated to minimize communication cost and maximize energy efficiency. Geographical and Energy Aware Routing (GEAR) [2,6] bounds flooding to a small region. Rumor protocol is an integration of Gossip and GRAB. It combines query flooding and event flooding [2].

Since, EER approach use flooding like DD in route finding, EER could be regarded as a member of flooding family, although its major part is in route maintenance phase.

3. Energy equivalence approach

3.1. Basic concepts

In data-centric routing paradigms, a feasible task is made up of a set of data paths, which fuse at *converging nodes* and form a *routing tree*, a reversed multicast tree. If energy consumption is distributed unevenly in a routing tree, especially if a high energy-consuming subgraph is critical to overall connectivity, the unevenness is disastrous to network lifetime. We call this critical subgraph *topological bottleneck* of lifetime. Typical topological bottlenecks include converging nodes [9] in same task and subnets on which several concurrent tasks overlap. Fig. 1 shows an example of the former case. In the example, radio radius is not long, so listening and transmitting can be considered to dissipate same energy. [7] One after another five sensors transmit data to converging node N. In each 10 time units, every sensing node dissipates 1 unit energy. N receives five data packets, integrates them into one packet and then passes it to sink. So for each round, N dissipates 6 units energy. After eight rounds, N has lost 48 units energy. So its residue is two, while residue of other nodes is 42.

To maximize network survivability by using equal energy among as many nodes as possible, we suggest an approach called neighbor switching, which utilizes density and path redundancy in WSNs. We assume that in the long run all sensor nodes have equal chance to become sources and sink.

3.2. Neighbor switching

Neighbor switching substitutes one node with a neighbor outside the original routing tree according to a given criterion. Neighbor switching is essential in EER approach. It changes routing tree at very small scale, so the energy uniformization is achieved with least energy cost. Fig. 2 shows how neighbor switching prolongs the lifetime. Its setting is same as Fig. 1. Fig. 2a is identical to Fig. 1b. Fig. 2b shows how network lifetime is extended from 80 without neighbor switching to 500 with neighbor switching. The red nodes are exhausted replacing neighbors.

Source and sink nodes could not be switched, so a switched node N always has a preceding node P and a set of succeeding nodes SS. If $|SS| > 1$, N is a converging node. The prerequisite of neighbor switching is adequate density or redundancy in vicinity of replaced node. Two types of neighbor switching exist. In *single neighbor switching* shown in Fig. 2, N is replaced by one *single neighbor* with most energy residue, which is common neighbor of N, P and every node in SS. It is used by EERS and EERL. In *double neighbor switching*, N is replaced by $|SS| + 1$ *double neighbors*; one is common neighbor of P and N, the rest are common neighbors of N and every $S \in SS$. Double neighbor switching is used by common neighbor switching (CNS).

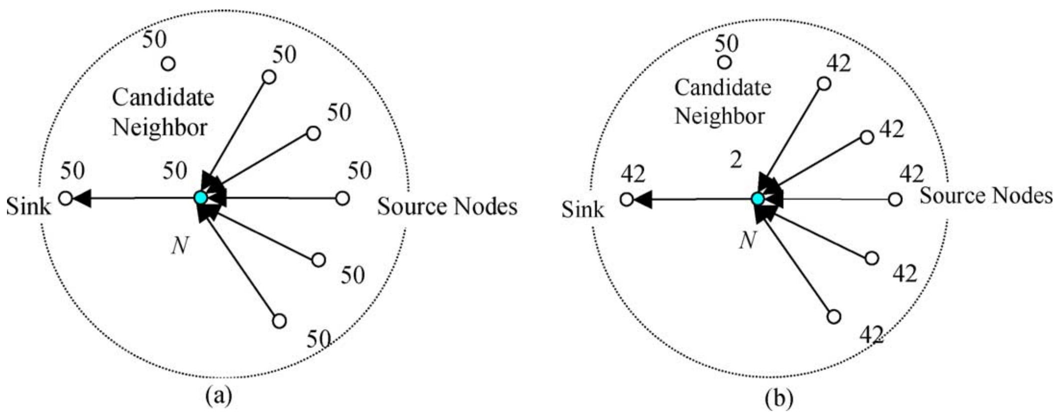


Fig. 1. Demonstration of unevenness in energy dissipation. (a) Initial energy 50 units for all sensor and (b) residual energy after 80 time units. N is a converging node, sensing interval = 10 time units, transmission energy = reception energy = 1.

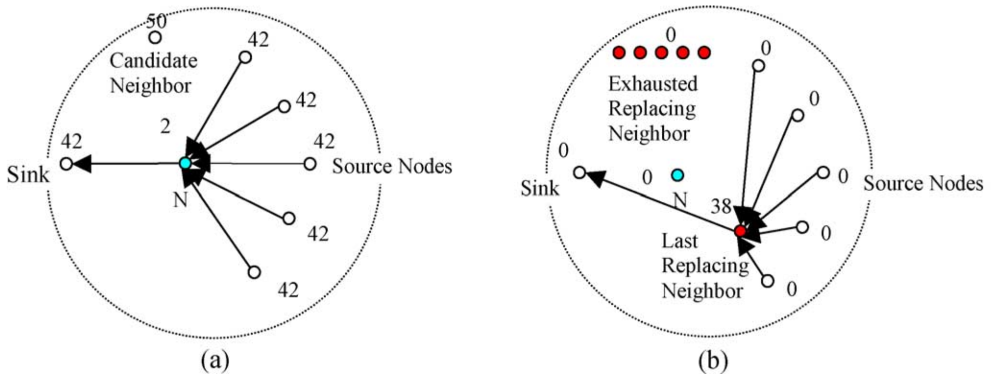


Fig. 2. Lifetime of a single task network is prolonged via single neighbor switching. (a) Lifetime is 80 time units without switching and (b) lifetime is 500 if sufficient replacing neighbors exist. N is a converging node, sensing interval = 10, energy cost for transmission and reception = 1.

3.3. Path rerouting

Path rerouting links an outside substitute neighbor back to the original routing tree, in which single neighbor switching is used. Rerouting is still a flooding. It costs much more energy than neighbor switching. Delay is also longer than neighbor switching. *Shortest rerouting* links the replacing neighbor to the nearest descendent node, while *longest rerouting* links to the farthest descendent node. To avoid energy waste in the chain reaction of a series of adjacent nodes, we define the nearest descendent node as the first descent in the path which does not need to be replaced. In *longest rerouting* it is better to reject the new path which intersects with original path at a non-rerouted node other than source and destination. Our rerouting procedures follow the DD protocol, but with the constraint that every node on rerouting path should need no rerouting, that is, it does not have a neighbor with energy difference beyond the threshold. However, the path may contain other nodes of the original routing tree, so the new path is not necessarily disjoint with the original path. Nodes immediately following it in the original routing tree also need to set up new paths to its substitute neighbor.

4. EER algorithms—CNS and others

CNS, a route maintenance algorithm using double neighbor switching, is most efficient in EER algorithms. CNS has minimal additional energy cost and minimal time complexity, since it never uses flooding in route maintenance. This has been supported by the simulation results. EERS and EERL are also presented to show that neighbor switching is better than path rerouting. Rerouting algorithms are inefficient. Simulation results show that they have almost same lifetime as DD, which means rerouting overhead balances out their gain in lifetime.

EERS and EERL both use single neighbor switching, but differ in destination selection. For EERS and EERL, the routing tree is divided into *line segments*, which are bounded by converging nodes. The major difference between EERS and EERL is the way they determine their rerouting paths on the same line segment. EERS divides each segment into as many sections as possible, with each section to be shortest to minimize topology variation. On the other hand, EERL chooses the longest possible substituent section in a hope to find a better path to equalize energy dissipation.

4.1. Assumptions

We assume that network has been initialized and routing tree for tasks has been set up before EER algorithms are called. During network initialization, the network topology is discovered and basic parameters are set. All nodes obtain important data about their neighbors after initialization, like energy, position and ID. Radio radius is fixed and identical for all nodes. Wake up mechanism is used to save lavish energy dissipation in active listening. Waking up uses a paging channel with negligible energy dissipation [8]. Note that in following three algorithms every node N except source and sink has a preceding node P and a set of succeeding nodes SS.

The rerouting algorithms postpone current data transfer. When sensing interval of a task is long as in usual applications and rerouting interval is set long, this delay is trivial. On the other hand, the delay may be a draw back and may not be neglected when network is dense or large.

Although algorithms are distributed, with communications limited to neighbors and via packet exchange, their control framework is centralized. Sink is the coordinator. Every execution starts at sink, and then goes through the routing tree like falling blocks in a domino game. They are activated by receiving reroute request packets: Extended Reroute Request (ERR) packet for CNS, Regular Reroute Request (RRR) packet for EERS and EERL. To make the call explicit and conventional, we separate receiving operation and execution. In fact these two operations are integrated tightly. An EER Common Entry Algorithm is executed by sink. It sends out corresponding reroute request packets to all succeeding nodes in routing tree.

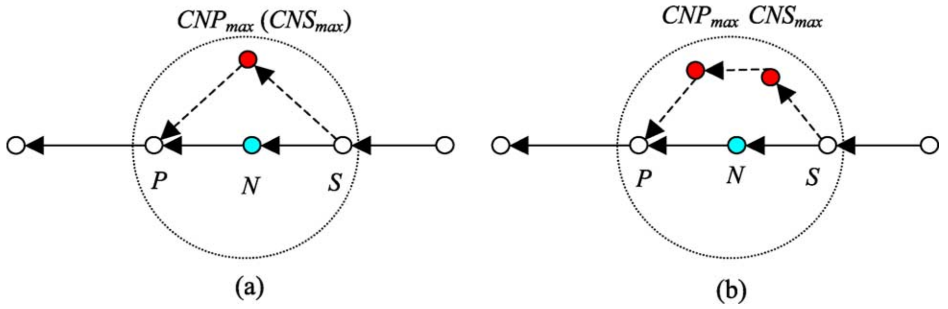


Fig. 3. Common neighbor switching. When (a) $CNP_{\max} = CNS_{\max}$ and (b) $CNP_{\max} \neq CNS_{\max}$.

4.2. Common neighbor switching EER algorithm (CNS)

Let $S \in SS, \{P, N, S\}$ forms a basic unit in Common Neighbor Switching. *CNP* is the set of double neighbors of *N* and *P* such that: (1) it excludes nodes on the original path; (2) a neighbor has more energy residue than *N*; (3) the energy difference between *CNP* and *N* is more than a predefined threshold. Similarly, *CNS* is a double neighbor of *N* and *S*. $CNP_{\max}$, $CNS_{\max}$, are elements in *CNP*, *CNS* with maximum energy residue, respectively. The basic idea of CNS, as shown in Fig. 3, is to replace *N* with $CNP_{\max}$ and $CNS_{\max}$, like a standard double neighbor switching. When $CNP_{\max}$ and $CNS_{\max}$ overlap, original path segment $S \rightarrow N \rightarrow P$ is replaced by $S \rightarrow CNP_{\max}$ (or $CNS_{\max}$)$\rightarrow P$, otherwise, it is replaced by $S \rightarrow CNS_{\max} \rightarrow CNP_{\max} \rightarrow P$. Below is the simplified CNS algorithm

```
CNS(N, CNSOption) {
    if ($N \in$ sources) return; // Source nodes cannot be switched
    for (every $S \in SS$) {
        N sends S a Call for Neighbors Information packet;
        S sends back a Neighbors Information packet;
        N calculates $CNP_{\max}$, $CNS_{\max}$;
        If ($CNP_{\max}$ and $CNS_{\max}$ exist) {
            if ($CNP_{\max}$, $CNS_{\max}$ overlap) Replace $S \rightarrow N \rightarrow P$ with $S \rightarrow CNS_{\max}$ (or $CNP_{\max}$) $\rightarrow P$;
            else {
                if ($CNP_{\max}$ and $CNS_{\max}$ can be connected)
                    Replace $S \rightarrow N \rightarrow P$ with $S \rightarrow CNS_{\max} \rightarrow CNP_{\max} \rightarrow P$;
                else Recover connection failure according
                    to CNSOption;
            }
            N removes itself from the routing tree
        }
        N sends an Extended Reroute Request packet to S;
        CNS(S, CNSOption);
    }
}
```

Most calculation happens on node *N*. When *N* is a converging node, that is, $|SS| > 1$, old path and new path may both exist. The original data stream is shared by $S_i \rightarrow N \rightarrow P$ and $S_j \rightarrow CNS_{\max} \rightarrow CNP_{\max} \rightarrow P$. Above algorithm may be adapted to concurrently process each *S* in set *SS*. There is no need for communication. This will save energy and time, but require that *N* have adequate memory and processing capacity.

Replacing is implemented by node's child attribute. For example, replacing $S \rightarrow N \rightarrow P$ with $S \rightarrow CNS_{\max}$ (or $CNP_{\max}$) $\rightarrow P$ is implemented as $CNP_{\max}.child \leftarrow N.child$, $S.child \leftarrow CNP_{\max}$, $N.child \leftarrow$ null. Replacing $S \rightarrow N \rightarrow P$ with $S \rightarrow CNS_{\max} \rightarrow CNP_{\max} \rightarrow P$ is implemented as $CNP_{\max}.child \leftarrow N.child$, $CNS_{\max}.child \leftarrow CNP_{\max}$, $S.child \leftarrow CNS_{\max}$, $N.child \leftarrow$ null.

If $CNP_{\max}$ or $CNS_{\max}$ cannot be connected (they are not neighbors), a *switching failure* occurs. Following options are provided to recover from such a failure (Fig. 4):

- CNS(Plain): use the original path $S \rightarrow N \rightarrow P$ and change nothing.
- CNS(Alternative): find another suboptimal $CNP_{\max}$ and $CNS_{\max}$ pair which are connected.

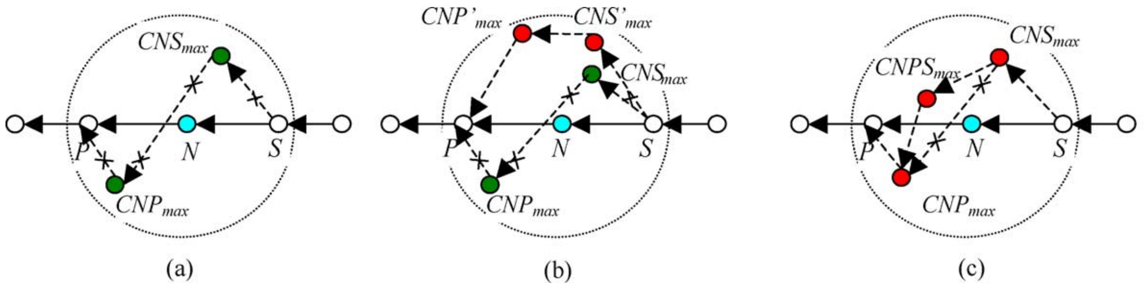


Fig. 4. Switching failure recovery. (a) CNS plain, no recovery; (b) CNS alternative, find another pair; and (c) CNS intermediate, find $CNPS_{\max}$.

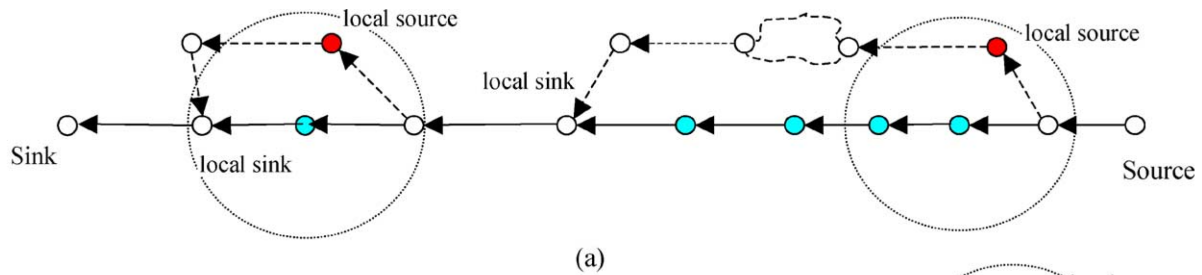
- CNS(Intermediate): find an optimal common neighbor $CNPS_{\max}$ of $CNP_{\max}$ and $CNS_{\max}$, use path $S \rightarrow CNS_{\max} \rightarrow CNPS_{\max} \rightarrow CNP_{\max} \rightarrow P$ to replace path $S \rightarrow N \rightarrow P$.

4.3. Shortest rerouting EER algorithm (EERS)

EERS switches an over dissipated sensor node to a neighbor and replacing original path with a new shortest rerouting path. Fig. 5a shows EERS on a path without converging nodes. To avoid expensive and redundant rerouting, successive nodes that need rerouting are grouped together into one rerouting. Unlike in EERL, more than one rerouting in a line segment is possible and encouraged in EERS. An example of EERS in a path with converging nodes is shown in Fig. 5b. Only one more situation needs to be handled, that is, a converging node, which needs rerouting. The leftmost converging node is simply the sink of a line segment in demand of rerouting. The rightmost one in need of rerouting is source of middle line segment and sink of rightmost line segment. Below is EERS algorithm

```
EERS (N, threshold, RCTimeout) {
    if ($N \in$ sources) return;
    N finds $N_{\max}$, the neighbor with highest energy outside the routing tree;
    if ($E(N_{\max}) > E(N) + threshold$) { // N needs rerouting
        N sends a Rerouting Confirm (RC) packet to P;
        if (N is a converging node) {
            Reroute from $N_{\max}$ to RRR.destination;
            RRR.destination $\leftarrow N_{max}$;
            N switch to $N_{\max}$;
            N multicasts a Regular Reroute Request packet to all $S \in SS$;
            for (all $S \in SS$) EERS (S, threshold, RCTimeout); // Concurrently trigger EERS on SS
        }
        else { // N is not a converging node (only one S in SS)
            N sends a Regular Reroute Request packet to S;
            // RRR.destination has not been changed
            EERS(S, threshold, RCTimeout);
            Wait for the Rerouting Conform packet from S until RCTimeout expires;
            if (not received) {
                Reroute from $N_{\max}$ to RRR.destination;
                RRR.destination $\leftarrow N_{\max}$;
                N resends a Regular Reroute Request packet to S;
                EERS(S, threshold, RCTimeout);
            }
        }
    else { // N does not need rerouting
        RRR.destination$\leftarrow$N;
        N multicast a Regular Reroute Request packet RRR to all $S \in SS$;
        for (all $S \in SS$) EERS (S, threshold, RCTimeout);
        // Concurrently trigger EERS on all $S \in SS$
    }
}
```

For any node, RRR.destination is the nearest preceding node which needs rerouting. If no such node exists, the destination is the nearest preceding converging node.



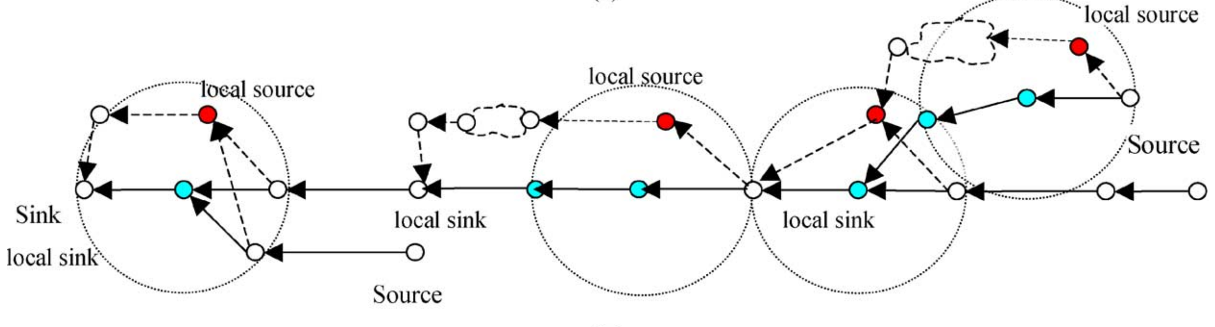


Fig. 5. Demonstrations of EERS. Blue nodes need rerouting, replaced by red nodes. Dashed line is the new path. Circles mark the radius of blue nodes. (a) Rerouting in a path without converging nodes and (b) rerouting in a path with converging nodes (for interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

If a node N does not need rerouting, it sets itself to RRR.destination and sends the RRR packet to all its succeeding nodes.

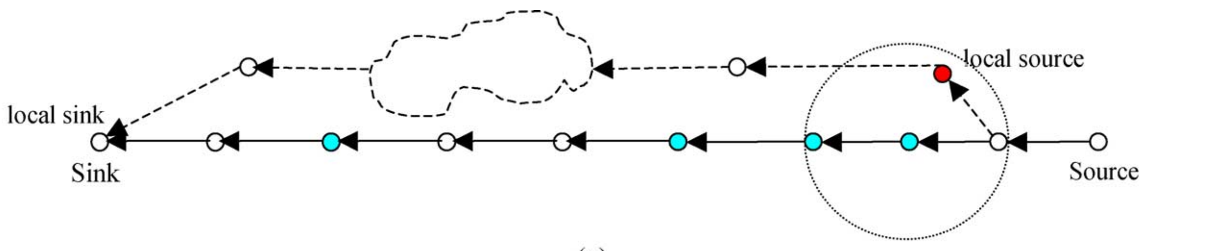
If N needs rerouting, it first sends back a Rerouting Confirm packet to P to suppress Ps attempt to reroute. Then it finds replacing neighbor $N_{\max}$ outside the routing tree. If N is a converging node, it sets off rerouting from $N_{\max}$ to its RRR.destination. Then N is replaced by $N_{\max}$. At the same time, N generates a new RRR packet with the destination set to $N_{\max}$, then N multicasts the RRR packet to all its succeeding nodes and EERS algorithm is set off on them.

If N needs rerouting but is not a converging node, it sends the only S in SS a RRR packet with same destination as its own. Afterwards it keeps waiting for the Rerouting Confirm (RC) packet from S till *RCTimeout*. If N receives RC from S within *RCTimeout*, it gives up its rerouting attempt. If no RC packet is received till *RCTimeout*, N starts rerouting procedure and switches to $N_{\max}$. N also generates a new RRR packet with the destination set to $N_{\max}$, and N resends the packet to S and sets off EERS algorithm on S.

4.4. Longest rerouting EER algorithm (EERL)

EERL switches an over dissipated sensor node and replaces original path with longest rerouting path. Fig. 6a shows EERL on a path without converging nodes. On each rerouting line segment, EERL uses a big flooding instead of several small flooding in EERS. We do not have enough information to compare flooding energy cost in EERS and EERL. EERL does not only skip redundant rerouting in a series of successive nodes in need of rerouting, but also combine all such series of nodes into one task of rerouting. Therefore, at most one rerouting in a line segment is needed in EERL. An example of EERL in a path with converging nodes is shown in Fig. 6b. The leftmost converging node is simply the sink of a line segment that needs rerouting. The rightmost one that needs rerouting is both source and sink of line segments

```
EERL (N, threshold, waitTimeout) {
  if ($N$ ∈sources) return;
  $N$ finds $N_{\max}$, the neighbor with highest energy outside
    the routing tree;
  if ($N$ is a converging node) {
    if ($E(N_{\max})>E(N)+threshold$) RRR.destination←
    $N_{\max}N$; // $N$ needs rerouting
    else RRR.destination←$N$;
    $N$ multicasts the RRR (Regular Reroute Request)
    packet to all $S \in SS$;
    for (all $S \in SS$) EERL($S$, threshold, waitTimeout);
    // Concurrently
    if ($E(N_{\max}) > E(N) + threshold$) { // $N$ needs
    rerouting
      $N$ sends a No Wait packet to $P$, which is passed
      until it reaches RRR.destination;
      Reroute from $N_{\max}$ to RRR.destination;
    else $N$ sends an End of Segment packet to $P$, which
      is passed along the segment until it
      reaches the first node that needs rerouting, the
      first converging node, or sink;
    else { // $N$ is not a converging node with only one $S$
    in $SS$
      $N$ sends a Regular Reroute Request packet $S$;
      EERL($S$, threshold, waitTimeout);
```



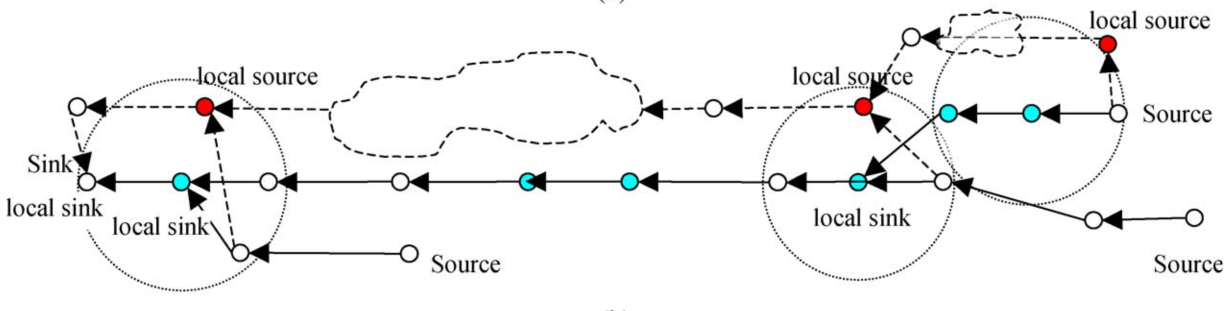


Fig. 6. Demonstrations of EERL. Blue nodes need rerouting, replaced by red nodes. Dashed line is the new path. Circles mark the radius of blue nodes. (a) Rerouting for a path without branches and (b) rerouting for a path with branches (for interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

```
        if ($E(N_{\max}) > E(N) + threshold$) { // $N$ needs rerouting
            Wait for an End of Segment or No Wait packet from $S$ until waitTimeout expires;
            if (ES packet received) Reroute from $N_{\max}$ to RRR.destination;
            else if (NW packet received) Give up rerouting;
            else Error exit;
            $N$ sends a No Wait packet to $P$;
        }
        else {
            Wait for an End of Segment packet or a No Wait packet from $S$ until waitTimeout expires;
            if (received) passes it to $P$;
            }
    }
}
```

In EERL RRR.destination is always the starting node of current line segment, so it must be a converging node or sink. Like in EERS N first finds out $N_{\max}$. If N is a converging node, N generates a new RRR packet with the destination set to $N_{\max}$ if N needs rerouting, otherwise set it to N. Then N multicasts the RRR packet to all succeeding nodes and EERL algorithm is set-off on them. If N needs rerouting, N sends P a No Wait (NW) packet and P passes it until it reaches RRR.destination. Next N starts rerouting from N_{max} to RRR.destination. If N does not need rerouting, N sends an End of Segment (ES) packet to P, and P passes it along the line segment until it reaches the first node that needs rerouting, the first converging node, or sink.

If N is not a converging node, N sends a RRR packet to S. If N needs rerouting it would wait for an ES or NW packet from S until *waitTimeout* expires. Before *waitTimeout*, if N receives ES packet, it sets off rerouting from $N_{\max}$ to RRR.destination; otherwise it give up rerouting attempt. At the end, N sends a NW packet to P. If N does not need rerouting, it waits for an ES or NW packet from S until *waitTimeout* expires. If N receives the packet, it passes it to P.

5. Simulation

5.1. Basic procedure

Our simulator is coded in Java 2 with Borland JBuilder 8 IDE It is set up as following:

(1) Network Topology Generation
In a 100×100 square, n nodes with equal initial energy and radio radius are independently generated and randomly distributed. Their connections are decided by distance. The same topology is used for all simulated protocols (or algorithms) like DD, CNS, EERS and EERL.

(2) Task generation
Tasks are generated with parameters like duration and generation interval. A 30×30 square source area is selected by randomly choosing its upper left corner. All nodes falling in the area are source nodes. Sink is randomly located. For simulations with same sensor number, no matter what initial energy level and EER algorithm are, same task series is used. The task series is used repeatedly, so it works for whatever possible network lifetime.

(3) Path generation
Flooding and reinforcement are used to generate initial paths for a task. All protocols use DD [5] for path setting up. For each source node a shortest path is generated. Then all shortest paths are aggregated at converging nodes.

(4) Communication and energy simulation
This is the major part of simulation program. Discrete time is used and a time unit is defined as interval needed to transmit or receive a packet. We assume that a node transmits a received packet in the next unit. Rerouting paths calculation is discrete, imitating the real world.

At any time unit, the simulator first checks if any task expires, then checks if a new task is generated, at last checks if it is time to call EER algorithm (which specific algorithm is called depends on an input parameter). If yes, do the EER algorithm along the path tree; if no, data transfer is executed. At any step, corresponding energy cost of each node is deducted. We assume no protocol has capability to throttle the data transfer of available paths, which means, all possible data paths are working as long as topology connectivity and energy residue permit (Fig. 7).

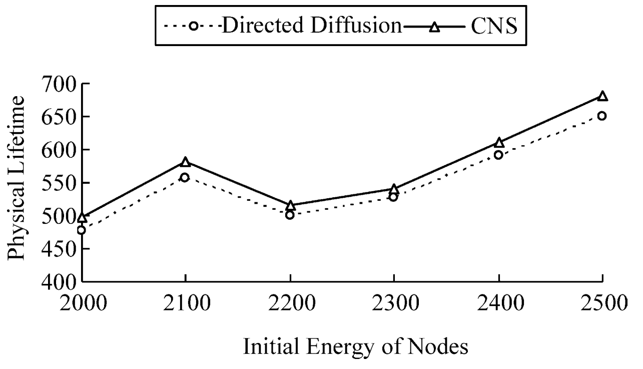


Fig. 7. Simulation result of CNS (Plain). Connection radius = 10.0; rerouting interval = 60; max time no tasks generated = 200; number of nodes = 300; difference threshold = 20; task failure ratio when ending = 0.6; rerouting starting period = 80; task generation interval = 10; reception energy = 0.5; task duration = 200; sink node has endless energy; transmission energy = 3.0; use the same series of randomly generated tasks. Starting period = 80; and end condition: Option 3 (test both the ratio of exhausted nodes and the ratio of failed tasks).

5.2. Lifetime and end condition

In our simulation, two lifetimes are defined. One is *physical lifetime*, which measures the simulation time from the beginning to its end for specified end condition. Physical lifetime includes every second in simulation. It makes no difference whether 100 tasks are running or no active task is running. However, the energy dissipation differs greatly. By this metric, for some protocol that saves and balance energy well, much more tasks may set up paths and become active than protocols with poor energy efficiency. Thus during the same time, much more energy is dissipated by the good protocol, so its physical lifetime may even be shorter than a poor protocol.

Physical lifetime does not correctly reflect energy consumption. We found another metric called *collected lifetime*, which is the sum of run time of all active tasks. It overcomes the deficit of physical lifetime. However, it is not perfect. For instance, routing trees for tasks vary remarkably in size, length, nodes involved, and branches. So during the same time interval, different tasks may consume wildly different energy.

The definition of network lifetime is the key for many research topics in WSNs. For a single task, it has been well defined [7], however, much effort is needed to find a definition that offers a picture for all task (Fig. 8).

One thing is very interesting in our simulation results and deserves more attention and further study. That is, lifetime of network is not linear with the initial energy level of sensors. For a single sensor or a network with only a few sensor nodes it is impossible. However, for a large and dense WSN, it is possible and reasonable. Actually, most our simulations reflect the same phenomenon with one accord. Probable explanation is that for a same deployment of sensor nodes, as time goes by, different initial energy levels result in considerably different residual network topologies. Higher level would give a more connected topology than lower level. Suppose for these two topologies, current task sets yield same task assignment, i.e. same subset of succeeding tasks; however, a same succeeding task may have totally different routing trees in different topologies if there are many source nodes. Again, higher-level network gives bigger routing tree than lower level network. Nonetheless, bigger tree may cost much more energy for same collected lifetime. The additional energy expenditure may be so much that it could balance out all the lifetime benefit of higher initial level and even more, and hence, results in a shorter collected lifetime and physical lifetime. Fig. 9 shows the phenomenon clearly. In fact, Figs. 7 and 8 also reflect some degree of same phenomenon, but since its metrics is physical lifetime only, it may be result from other causes like different successive task subset.

In our simulation, lifetime, physical or collected, is decided by the simulator. We give four options for end condition. The first is the rate of failed tasks in task set, if the rate goes over a certain threshold, simulator will stop. This is the strictest option, sometimes the simulator stops even when a big portion (50–70%) of nodes have remarkable energy residue. Second is the rate of exhausted nodes, it is looser than first, but does not always work when threshold is high. In many topologies, especially sparse ones, some critical nodes dominate almost all possible paths; no path

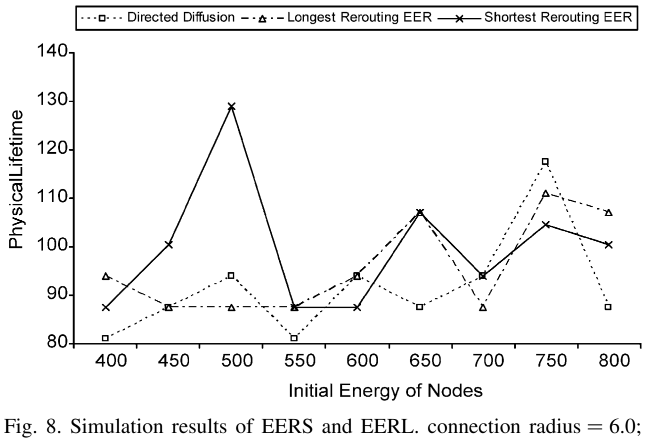


Fig. 8. Simulation results of EERS and EERL. connection radius = 6.0; rerouting interval = 80; task failure ratio when ending = 0.7; number of nodes = 50; difference threshold = 50; reception energy = 1; rerouting starting period = 80; task generation interval = 10; transmission energy = 3; task duration = 60; starting period = 80; sink node has same initial energy; and end condition: Option 1. Result is the average of 20 runs with different network topology.

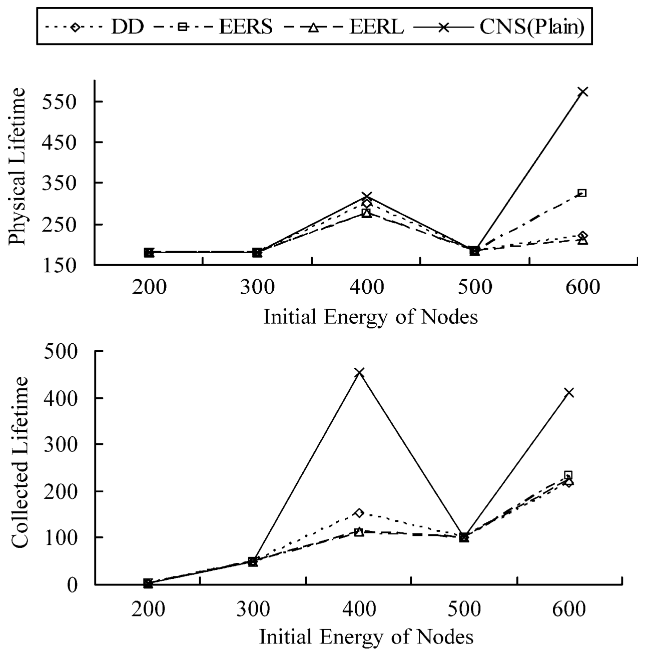


Fig. 9. Simulation result with physical and collected lifetime. Connection radius = 10.0; Rerouting interval = 50; max time no tasks generated = 100; Number of nodes = 150; Difference threshold = 10; Task failure ratio when ending = 0.9; Rerouting starting period = 50; Task generation interval = 10; Reception energy = 0.5; Task duration = 300; Sink node has endless energy; transmission energy = 2.0; use the same series of randomly generated tasks; starting period = 80; and end condition: Option 4.

can be set up when they are exhausted. If only these critical nodes are exhausted, simulation may run forever without any change in parameters, because most tasks have failed, but rate of exhausted sensor nodes keeps very low. The third is loosest; it requires both rate of failed tasks and rate of exhausted sensor nodes be over certain threshold. Very long lifetime is reached for given threshold. The last one is more practical. Under this end condition, simulator stops when the life time is longer than another limit while the rate of failed tasks is more than a given threshold.

6. Conclusion

In this paper, we proposed CNS algorithm as the best EER approach to balance network wide energy consumption and prolong network lifetime. There are two ways to reach energy equivalence, that is, neighbor switching and path rerouting. According to performance of CNS, EERS, and EERL, neighbor switching is much better. The simulation results indicate that CNS algorithm outperforms typical existing protocol. It is a promising approach and deserves more future research. Future research topics could be:

- Find algorithms or heuristics to determine appropriate rerouting interval and difference threshold.
- For CNS algorithm further research is needed to (1) find the best parameters which give best performance, especially for network density and initial energy; (2) investigate two remedies for connection failure—CNS Alternative and CNS Intermediate, find out if they have advantage over CNS Plain and identify specific circumstances; (3) give a complexity analysis of lifetime and energy. It is still possible to find better algorithms because of many unexplored details in the algorithm.
- Find mathematical or algorithmic model for EER, give upper bound and lower bound of lifetime in complex real WSNs.
- Try to find means that carry out neighbor switching and rerouting without interrupting normal data transfer. Investigate possible conflict between rerouting of co-existing tasks.

Acknowledgements

This project is partially funded by NSF under contracts ITR-0312632 and IIS-0329738.

References

[1] I.F. Akyldiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (March 2002) 393–422.

[2] D. Braginsky, D. Estrin, Rumor routing algorithm for sensor networks, WSNA'02, Atlanta, GA, September, 2002.

[3] F. Ye, S. Lu, L. Zhang, GRAdient Broadcast: a Robust, Long-lived Large Sensor Network. http://irl.cs.ucla.edu/papers/grab-tech-report.ps.

[4] M.-J. Lin, K. Marzullo, S. Masini, Gossip versus deterministic flooding: low message overhead and high reliability for broadcasting on small networks. UCSD Technical Report TR CS99-0637. http://citeseer.nj.nec.com/278404.html.

[5] C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, Proceedings ACM MobiCOM 2000, Boston MA, August 2000.

[6] Y. Yu, R. Govindan, D. Estrin, Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks, UCLA Computer Science Department Technical Report UCLA/CSD-TR-01-0023, May 2001.

[7] M. Bhardwaj, A.P. Chandrakasan, Bounding the lifetime of sensor networks via optimal role assignments, Proceedings IEEE INFOCOM 2002, New York, June 2002.

[8] M. Singh, V.K. Prasanna, Optimal energy-balanced algorithm for selection in a single hop sensor network, First IEEE International Workshop on Sensor Network Protocols and Applications, Anchorage, AK, May 11, 2003.

[9] B. Krishnamachari, D. Estrin, S. Wicker, Modelling data-centric routing in wireless sensor networks, Proceedings IEEE INFOCOM 2002, New York, June 2002.