Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 29 (2006) 3425–3444
computer communications
www.elsevier.com/locate/comcom

Cross-layer application-specific wireless sensor network design with single-channel CSMA MAC over sense-sleep trees

Rick W. Ha, Pin-Han Ho, X. Sherman Shen *

Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ont., Canada N2L 3G1

Available online 10 March 2006

Abstract

Because of their pervasiveness and autonomy in operation, mesh-based wireless sensor networks (WSNs) are an ideal candidate in offering sustained monitoring functions at reasonable cost over a wide area. However, devising an energy-efficient, cost-effective, and reliable communication strategy for WSNs requires tight collaboration of all of the sublayers, which introduces new technical challenges in the areas of data, network and power management. This paper proposes a cross-layer sleep scheduling-based organizational approach, called Sense-Sleep Trees (SS-Trees), that aims to harmonize the various engineering issues and provides a method of increasing monitoring coverage and operational lifetime of mesh-based WSNs engaged in wide-area surveillance applications. An iterative algorithmic approach is suggested to determine the feasible SS-Tree structures to achieve such design goals. Based on the computed SS-Trees, optimal sleep schedules and traffic engineering measures can be devised to balance sensing requirements, network communication constraints, and energy efficiency. For channel access, a simple single-channel CSMA MAC with implicit acknowledgements (IACKs) is selected to complement SS-Tree implementation because of its flexibility and adaptability in face of the unique operational characteristics of WSNs.

Keywords: Wireless sensor networks; Sleep scheduling; Cross-layer design; Medium access control; CSMA; Implicit acknowledgements

1. Introduction

Recent technological advances in wireless sensor networks (WSNs) [1–3] give inspiration to the development of a new breed of surveillance systems that can offer additional capability and flexibility over existing options for monitoring a wide range of physical phenomena at reasonable cost. The emphasis on energy efficiency is paramount in WSN design because each sensor node is equipped with only a finite amount of battery supply. Aside from avoiding the use of energy-inefficient and complex hardware components, extra energy savings can be achieved through aggressive power management strategies in devising adaptive sleep schedules to minimize the amount of energy lost to needless transceiver idle listening [4–6].

While the main implication of sleep scheduling in the MAC layer is the shortening of the time the radio transceiver is engaged in idle listening, incidences of overhearing can also greatly be reduced as nodes in sleep mode are no longer eavesdropping on the wireless medium. For routing algorithms, however, link table entries will expire prematurely if an intermediate node sleeps and shuts off all links to its neighbors without prior notification, thus forcing frequent recomputation of routing paths [7]. In addition, network maintenance functions such as neighborhood discovery and time synchronization must also operate within the short time frame that the transceiver is active, thereby competing for bandwidth and processing power. In the application layer, real-time data reporting functions are subject to constant and debilitating routing path breakages due to sleeping nodes. Because of these far-reaching

* Corresponding author. Tel.: +1 519 886 4567x2691; fax: +1 519 746 3077.
E-mail addresses: rwkha@bbcr.uwaterloo.ca (R.W. Ha), pinhan@bbcr.uwaterloo.ca (P.-H. Ho), xshen@bbcr.uwaterloo.ca (X.S. Shen).

0140-3664/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2006.01.019

effects, a cross-layer perspective should be taken in devising sleep schedules to collectively tackle the inter-layer issues.

Besides sleep scheduling, the other primary method in minimizing energy wastage is to reduce the amount of data traffic traversing the WSN. The obvious way is to limit the amount of sensing data generated by the nodes through issuing less data requests from the data sink to individual nodes for request-driven data, raising the event reporting threshold for event-driven data, and decreasing data reporting frequency for timer-driven data. Another way to reduce overall data traffic is to implement data aggregation and duplicate suppression schemes to compact similar data packets. On the other hand, energy savings can also be achieved by reducing the amount of control overhead in node addressing, MAC signaling, routing procedures and network maintenance. In any case, such data reduction measures would significantly limit both monitoring capability and network management flexibility of WSNs.

In addition to energy efficiency, another important issue to consider in WSN design is the cost of sensor nodes. Given a WSN can contain thousands of nodes or more, any increase in per unit cost, no matter how minute, would be magnified into a substantial overall cost due to the large number of nodes deployed, which would make WSNs an expensive technology in comparison with other more affordable alternatives and thereby diminishes their economic appeal. Therefore, the prospective WSN design calls for a bare-bones approach in component selection where only the most essential capabilities will be offered.

Given the many WSN design considerations, the real engineering challenge henceforth is to devise a comprehensive yet manageable network organization and communication paradigm that can harmonize the various design criteria without creating significant conflicts in optimization objectives. This paper concentrates solely on the application of WSNs in wide-area surveillance applications, and it is organized in the following manner. First, Section 2 explores issues regarding sleep scheduling and presents a new organizational methodology, called Sense-Sleep Trees (SS-Trees), that aims to maximize energy efficiency in WSN design. Next, operational stages of the SS-Tree scheme are described in Section 3. An iterative algorithmic approach for computing SS-Trees is suggested in Section 4. Section 5 discusses cross-layer sleep scheduling issues and devises a thorough solution to satisfy MAC, routing and application requirements for steady-state WSN operations. Evaluation on the validity and effectiveness of the proposed cross-layer SS-Tree scheme are provided in Sections 6 and 7. Finally, Section 8 offers some concluding remarks and outlook for future research.

2. Sleep scheduling issues and the SS-tree concept

Unlike other types of communication networks, WSNs are characterized by a simple and stable traffic flow pattern, where data is unilaterally streamed from the sensor nodes to the data sink with occasional data requests and network control packets disseminating downstream. So a legitimate approach to organize WSNs is to network all the nodes with a large spanning tree structure that is rooted at the data sink to minimize uplink and downlink communication costs. Forwarding messages in a shortest path spanning tree has the advantage of locating a lowest cost path between each of the nodes and the data sink, which enables minimum cost forwarding and source routing to perform effectively [8,9]. Also, junction points are ideal locations for performing data aggregation and in-network processing to reduce the data traffic traveling upstream. Clustering is the best known example of utilizing the tree structure for WSN formulation [10,11], though it requires higher nodal density for proper cluster formation. The star topology is also an example of a tree structure where all of the nodes are leaves connected to the data sink via one hop, which of course cannot be applied to all types of WSN topologies. A linear chain, which is used in chain-based routing protocols [12,13], is a special case of a tree where the number of descendents per node is 1. For the envisioned wide-area surveillance WSN, the spanning tree structure will be based on the underlying mesh network and no explicit clustering procedure will be pursued because of the lower network density and shorter communication range. However, relying on existing shortest path or minimum-weight spanning tree algorithms may not be adequate since the resultant spanning tree and routing paths have to negotiate around the unique WSN operational constraints such as large nodal population and ultra-low communication duty cycle (i.e., $<1\%$).

Because sleep scheduling is an integral part of the proposed WSN design, compatibility issues of spanning tree management and sleep scheduling should be investigated with prudence. Random sleep scheduling is not recommended because it will exert a detrimental effect on network connectivity and topology control efficiency under an ultra-low duty cycle. On the other hand, while implementing a global sleep schedule for all of the nodes would be relatively uncomplicated on a spanning tree structure, a network-wide communication blackout exists during the long sleep periods where none of the nodes would be active for packet forwarding. This lull in communication will adversely impact the effectiveness and temporal sensitivity of monitoring.

One way to shorten the communication blackout period while maintaining the percentage of sleep time is via forming a spanning tree with the maximum number of leaves such that the non-leaf nodes, often referred to as the dominant set, form a virtual backbone [14]. Groups of leaf nodes can be turned on and off successively to provide interleaved coverage while minimizing energy usage through regular sleep scheduling. The main concern with this approach is that it requires the non-leaf nodes to remain in active mode for longer periods of time to accommodate the varying sleep schedules of the leaf nodes, thereby depleting their battery reserves much sooner.

Thus far, a sleeping node has been viewed simply as a service void in the WSN that disrupts the ability to forward

data packets from neighboring nodes. Yet a sleeping node also ceases to exert radio interference in the wireless channel, which reduces incidences of overhearing and packet collision. Also, a sleeping node removes itself from the active data exchanges, thereby simplifying the WSN topology and making routing procedures less complicated. Therefore, as long as sleep scheduling is implemented in a controlled manner, the benefits of topology simplification can be realized without sacrificing connectivity or sensing capabilities.

The following example describes the principal concepts in using coordinated sleep scheduling for topology simplification. Fig. 1(a) shows a simple WSN with a data sink and nine nodes arranged in a square grid pattern. Suppose that a spanning tree with three branches is logically overlaid on top of the original topology, and all nine nodes follow the same global sleep schedule, as shown in Figs. 1(b) and (c), respectively. Then during the active period, considerable amounts of overhearing and packet collisions, represented as the dashed lines in Fig. 1(b), will occur amongst neighboring node, while none of the nodes will be capable of communicating during the sleep period.

Now suppose that the nine nodes are divided two groups of 3 and 6, respectively, in the manner shown in Fig. 1(d), where each group follows its own sleep schedule such that the active periods of each group alternate, as illustrated in Fig. 1(e). Instead of partitioning the WSN into separate sleep regions, the nodes of each group are arranged to form a tree rooted at the data sink with much sparser branches such that nodes on separate branches cannot communicate with one another. As a result, each node now has much fewer neighbors than those on the logical tree depicted in Fig. 1(b). Therefore, theoretically even with the use of only a single wireless channel, incidences of packet collisions from channel corruption will be drastically reduced. Also since the potential sources of overhearing are limited to just two neighbors instead of up to eight in the previous case, fewer overheard packets will be extraneously processed and less energy will be consumed. Since the nodes on each tree share the same sensing and sleeping cycle, the tree itself is named as Sense-Sleep Tree, or SS-Tree for short.

Besides achieving energy savings from simplifying the WSN topology, notice that in Fig. 1(d) the sleeping nodes, colored as white, are strategically located beside at least one branch of the other SS-Tree. As mentioned before, although the nodes turn off their radio transceivers when they enter sleep periods, they do remain on alert for signs of abnormality or emergency events in the meantime. Those nodes residing on the active SS-Tree during an active period can be viewed as the virtual backbone that links the adjacent nodes that have their transceivers turned off. However, since different SS-Trees rotate in time as the communication backbone, they avoid overburdening any set of nodes from being the sole virtual backbone. Therefore, SS-Trees allow the nodes to increase *monitoring sensitivity* (i.e., greater number of *event reporting windows*) for emergency events that generate event-driven data without altering the communication duty cycle or reporting frequencies on timer-driven data.

Despite this performance advantage for SS-Trees, there exist several issues to be considered regarding its implementation. For example, at least 100 distinct adjacent active paths with correctly interleaved sleep schedules at 1% communication duty cycle are needed to provide continuous real-time event reporting coverage for a single node. While this may not feasible due to limited nodal density and high SS-Tree computation complexity, even having just a few active paths per sensor node would be enough to enhance event reporting capability tremendously. Intuition suggests that the number of SS-Trees to be computed has to be less than the average nodal degree of the WSN topology, but finding the exact relationship between a given WSN topology and the number of SS-Trees is beyond the scope of this paper and is left as future work.

Fig. 2(a) shows the same basic WSN topology as that used in Fig. 1, but the nine nodes are now arranged into three distinctly colored SS-Trees operating under the same low communication duty cycle with the sleep schedules

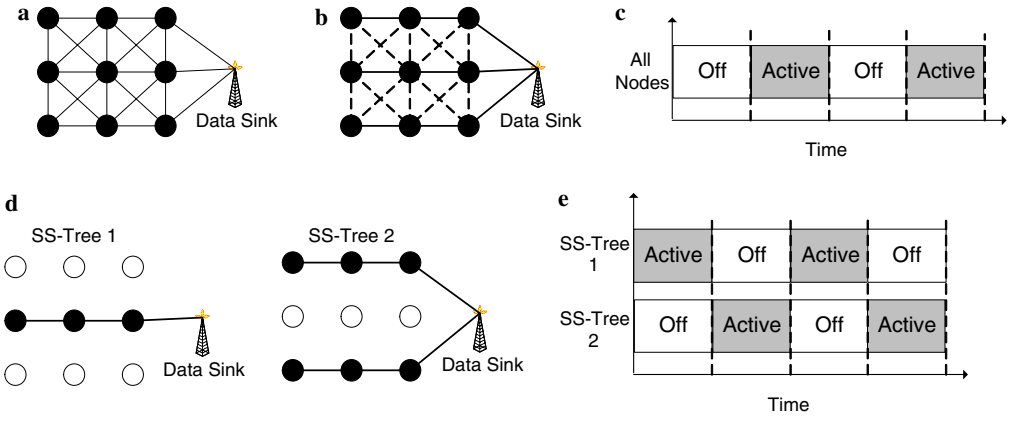


Fig. 1. SS-Tree concept. (a) WSN topology. (b) Logical spanning tree overlay. (c) Global sleep schedule. (d) SS-Tree configuration. (e) Interleaved sleep schedules.

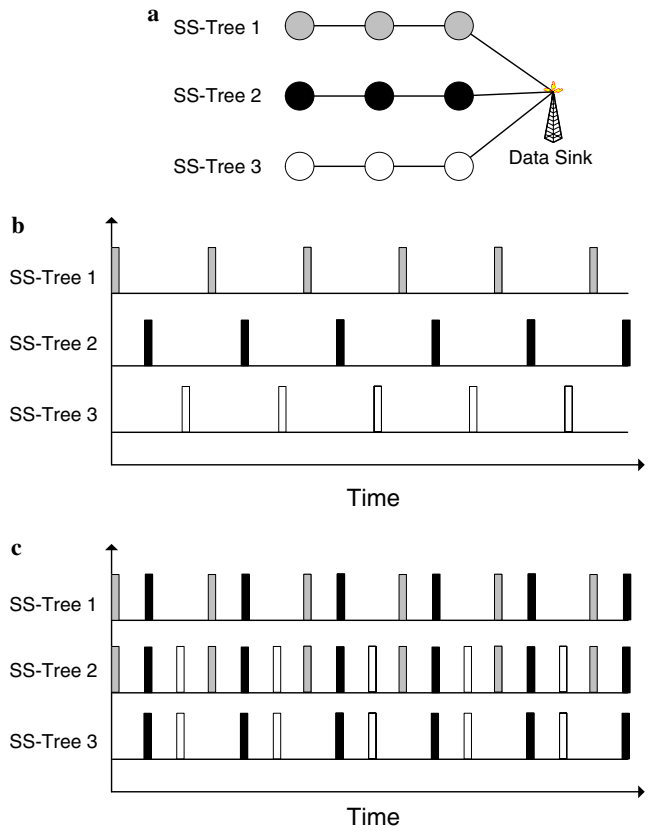


Fig. 2. Impact of SS-Tree on communication duty cycle and event-driven data reporting. (a) SS-Tree assignment. (b) Staggered sleep schedules. (c) Event reporting windows.

arranged in a staggered manner, as shown in Fig. 2(b). With SS-Tree 1 and SS-Tree 3 serving as active paths during their respective active periods, SS-Tree 2 is able to increase its monitoring sensitivity by 3 times, as demonstrated in Fig. 2(c). On the other hand in the same figure, SS-Tree 2 acts as active paths during its active period for the other two trees to increase their monitoring sensitivity by two times, though the event reporting windows are not equally distributed in time. In between the event reporting windows, the sensor nodes have the option to reduce the sampling rate or even turn off their sensing unit during the communication blackout periods to further minimize energy consumption.

While SS-Trees can provide extended monitoring coverage for event-driven data for about the same amount of energy, one minor drawback is that timer-driven data cannot be simultaneously gathered from all SS-Trees since each follows its own sleep schedule and only one SS-Tree may be active at any given time. For surveillance applications though, the impact of having unsynchronized periodic reports of ambient conditions and operational status of sensor nodes is far less significant than experiencing any delays in alerting the data sink of urgent events. Therefore, besides requiring the WSN application to tolerate a higher delay in timer-driven data reporting, event-driven data is to be given a higher priority in packet delivery that allows it to be expedited to the data sink on the minimum cost path when both types of data are injected into a SS-Tree during the active period.

3. SS-Tree operational stages

Fig. 3 shows the complete operational stages throughout the WSN's life cycle using SS-Trees. Soon after initial nodal deployment, the WSN will enter the *Network Initialization* stage, which allows the data sink to gather network connectivity information from individual sensor nodes, compute the SS-Trees, and disseminate the sleep schedules to every sensor node. The sensor nodes will then alternate between *Active* and *Sleep* stages for the majority of their lifetime in providing constant physical monitoring and performing the necessary data reporting tasks. During prolonged periods when sensing services are not needed, the entire WSN would enter *Hibernation* mode to conserve the maximum amount of battery power. To preserve network integrity, sensor nodes need to undergo the *Neighborhood Update* process periodically for keeping informed of any status changes of adjacent nodes in sleep schedules or hardware failure. Finally, sensor nodes and the data sink will enact the *Failure Recovery* procedures in case node failures are detected. The following paragraphs will further explain the operational dynamics in each of the stages.

To realize the benefits of SS-Trees, it is important to devise an efficient method for determining and disseminating the sleep schedule to all of the nodes during the *Network Initialization* stage. Distributed approaches for sleep schedule computation offer better scalability and robustness against single point of failure [4,5]. However, because of the need to adapt to different monitoring sensitivity requirements in response to varying environmental conditions, the optimal sleep schedules should be prepared by the data sink or the more powerful processing center since they are most sophisticated to handle scheduling decisions in a global manner. With direct source routing and efficient broadcast trees, any rescheduling commands issued by the data sink can be delivered to nodes swiftly. Also, since the

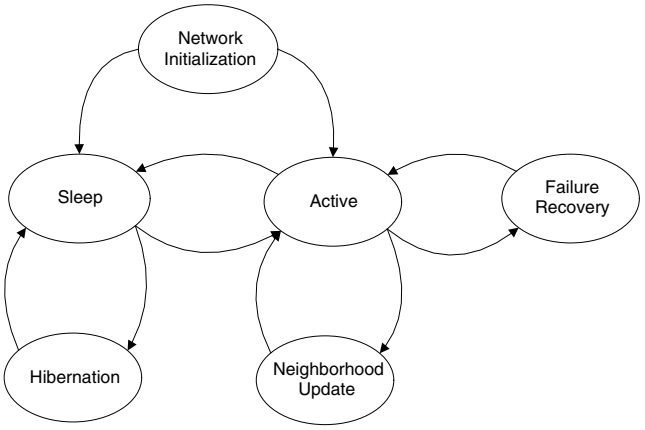


Fig. 3. WSN operational stages with SS-Trees.

data sink possesses global knowledge of network connectivity and link costs, the same sleep schedule can be assigned to nodes located on the same routing path, thereby ensuring a green light for packet delivery all through the path during active periods and guaranteeing the minimal end-to-end propagation delay.

During the *Sleep* state, the sensor node shuts down the radio transceiver to conserve power, thereby excluding it from intra-WSN communication. However, other hardware components such as the processor and the sensing unit can remain active to allow the sensor node to monitor the surrounding area for signs of abnormalities. In case an emergency situation arises, the sensor node which sensed the abnormality will search its neighborhood sleep scheduling list and find out which nodes are scheduled to be active next. When that active period commences, the sensor node will turn on its transceiver and sends a notification message to the active neighbor. That neighbor will in turn forward the urgent message to the data sink through the active SS-Tree.

When the WSN is expected to undergo an extended period of inactivity, the entire sensor node population should enter *Hibernation* state by shutting off all hardware components except for a tiny low-power wakeup timer. While the nodes will gain a few months of rest during hibernation, they should periodically wake up to participate in global synchronization sessions to minimize clock drifts. These sessions are also ideal for notifying the nodes of any changes to the hibernation schedule, as well as to detect any changes to the network topology from nodal failure or newly added nodes.

Given the ultra-low communication cycle, sensor nodes will spend only a tiny fraction of their lifetime in the *Active* state. Nonetheless, it is the most important state with respect to the overall operation of the WSN where all the necessary data reporting and network maintenance tasks are performed within this short time span. In order to complete all packet transmission and forwarding activities within a single short active period, the data reporting process should also be carefully choreographed instead of letting the nodes transmit packets at will to further reduce management overhead. In addition, data aggregation and duplicate suppression will be enforced so that the processing load will be distributed as evenly as possible along routing paths and across the sensor field to prevent premature battery depletion. Such issues related to the sensing application requirements will be discussed later in Section 5.

A potential issue with SS-Trees is their vulnerability toward nodal failures since any failed intermediate node will instantly sever the end-to-end path on the spanning tree. Since each sensor node does not keep the full network connectivity information, it is difficult to route around the failed nodes via some distributed algorithm alone. Also, the *Failure Recovery* process is further complicated by the fact that neighboring nodes often cannot reach each other during steady state operations as they belong to separate SS-Trees with different sleep schedules. Therefore, in order to recover from nodal failures without merely resorting to classic flooding, it is important to let neighboring nodes be aware of each other's sleep schedule through exchanging local information during *Neighborhood Update* sessions from time to time. Whenever a node senses an upstream breakage on its SS-Tree branch, it can refer to the stored neighborhood sleep schedules and reconnect to the data sink via the next neighboring node that is scheduled to become active. More importantly, the data sink would assume a central role in permanently repairing SS-Trees with the help of the global connectivity and sleep scheduling information it possesses. Such neighborhood update and failure recovery issues are beyond the scope of this paper and they will be instead delegated as part of the ongoing research work on SS-Trees.

4. SS-tree computation

Since SS-Tree is a novel concept in WSN organization and management, issues such as sleep schedule determination, data dissemination dynamics, neighborhood discovery process, and failure recovery procedures remain to be explored. However, the core problem for realizing the SS-Tree concept is the actual determination of how the sensor nodes can assigned to a fixed number of SS-Trees on a given WSN topology. With multiple SS-Trees coexisting in a WSN comes the possibility of tree overlapping, where a selected number of sensor nodes may have to belong to multiple SS-Trees to maintain tree connectivity. Such nodes, called *shared nodes*, need to follow multiple sleep schedules since each SS-Tree maintains its own sleep schedule. Therefore, the shared nodes constitute the weak points of the network where they will deplete their batteries much sooner than the rest of the WSN population. Since no existing tree computation algorithm or general approach has been suggested regarding SS-Trees, the following formal definition of the SS-Tree problem will address the various objectives described thus far.

Symbols - Let:

V – set of all sensor nodes plus the data sink
E – set of all bidirectional links between nodes in V
K – set of all SS-Trees
s – symbol representing the data sink in V

Problem Definition: Given an undirected connected graph $G = (V, E)$ with node s denoted as the data sink, form $|K|$ connected subgraphs (SS-Trees), all rooted at node s, with the following main objectives:

1. Minimize the number of shared nodes (i.e., nodes belonging to multiple SS-Trees).
2. Minimize the number of co-SS-Tree neighbors of each node.
3. Minimize the cost of forwarding messages between the data sink and each node.

Since the presence of shared nodes on SS-Trees has the most adverse impact on the expected lifetime of a given WSN, it becomes the top priority in the proposed computation approaches. For the second objective, each node should preferably be adjacent to the maximum number of neighbors that reside on other SS-Trees in order to take advantage of a larger number of available event reporting windows. Also, specifying a smaller number of co-SS-Tree neighbors per node would decrease the amount of overhearing interference produced. The third objective requires that all nodes on each SS-Tree should reside on the minimal cost path to the data sink, where the cost can be interpreted in terms of energy usage, transmission distance or hop count.

The objective of the proposed SS-Tree computation algorithm is to offer a fast approach to compute SS-Trees while balancing the three objectives outlined in the problem definition. The algorithm follows a greedy depth-first approach that constructs SS-Trees from the bottom-up on a branch-by-branch basis. The general idea is to construct the SS-Trees based on the underlying shortest path tree rooted at the data sink as determined by Dijkstra's algorithm. The SS-Tree computation algorithm proceeds in a number of *iterations*, where in each iteration an end-to-end minimum cost path is appended to one of the SS-Trees. At the start of the algorithm, all the nodes in the WSN topology are deemed *unselected*, and each path is built starting from the node with the highest path cost and uses as many unselected nodes as possible along the way. Each iteration is then divided into a number of *steps*, where in each step the path grows by one hop by adding a single unselected upstream node belonging to the next lowest hop level to the currently selected set of nodes. For picking the ideal unselected upstream node among multiple candidates, the selection criteria favor those with the fewest number of neighbors in an effort to reduce the number of shared nodes in future iterations. If no unselected upstream node is available for selection, then a node is picked from those that were already selected, which carries the risk of creating a shared node if the selected node belongs to a different SS-Tree. Path construction for a given SS-Tree in the current iteration stops when either the data sink or a selected upstream node belonging to the same SS-Tree is reached.

Across the WSN topology, two constructed paths are said to be *adjacent* if all of the vertices on one path are adjacent to at least one vertex on the other path, and vice versa. If each path is assigned to a different SS-Tree, then the nodes on both paths will enjoy the advantages of increased sensing duty cycle and added protection from nodal failures. However, suppose in every iteration the algorithm constructs the paths by selecting candidates from the same pool of unselected nodes, then it would be very difficult to maintain path adjacency among different SS-Trees because the constructed paths can crisscross each other in an unordered fashion. An intuitive approach to maximize path adjacency is to construct a path from the set of nodes that are neighbors to the nodes belonging to a different SS-Tree.

To illustrate this idea, Fig. 4 shows the successive iterations in computing 2 SS-Trees for a 25-node square grid WSN, where path cost of each node is represented by its hop count to the data sink. Here, the nodes colored in solid black and solid white represent that they are selected for SS-Tree 1 and SS-Tree 2, respectively, whereas nodes colored in solid grey indicate they are shared nodes, which means they belong to both SS-Tree 1 and SS-Tree 2. For example, after a path is constructed for SS-Tree 1 in Fig. 4(a), all of the selected nodes' neighbors, which are colored in vertical stripes pattern, become the set of candidate nodes, or *candidate set*, from which the next path for SS-Tree 2 is based upon. Subsequently when a path is constructed for SS-Tree 2 in Fig. 4(b), all of its selected nodes' neighbors will form the candidate set for SS-Tree 1, which

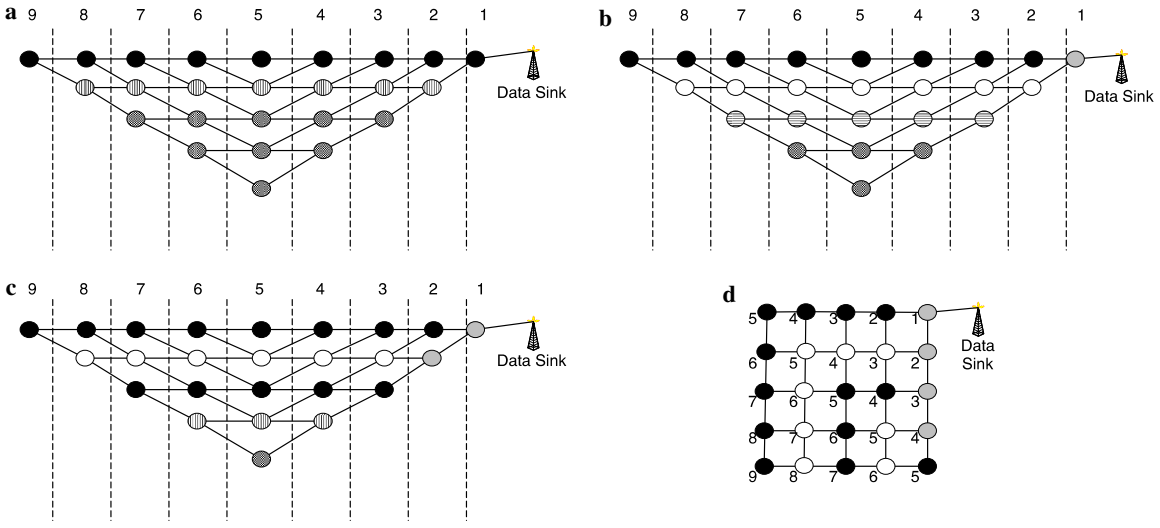


Fig. 4. Successive iterations in SS-Tree computation. (a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Final results.

are colored in horizontal stripes. The path construction process continues until every node is selected, and the final SS-Tree configuration is shown in Fig. 4(d).

The psuedocode for computing k SS-Trees on a given WSN topology is shown below, where the algorithm is divided into the main program and two subroutines:

Input: An adjacency list or matrix describing the complete WSN topology, where each node is aware of its hop count to the data sink.

Output: $|K|$ sets, each named S_{Sk} for $k = [1..|K|]$, where each set contains the nodes that belong to each of the SS-Trees.

Variables: S_{Sk} – selected set for SS-Tree k, S_{Ck} – candidate set for SS-Tree k, S_U – unselected set.

Initialization: $S_U \leftarrow V - s$, $S_{Sk} \leftarrow s$, $S_{Ck} \leftarrow$ NULL, for $\forall k \in K$

Program COMPUTE_SST

```
1. while (|S_U| + Σ_{k∈K}|S_Ck|) > 0 do
2.   if Σ_{k∈K}|S_Ck| = 0
3.     run MAKE_NEW_PATH
4.   end if
5.   for k counts from 1 to |K|
6.       if (|S_Ck| > 0)
7.       run MAKE_SSTk_PATH
8.     end if
9.   end for
10. end while
```

Subroutine MAKE_NEW_PATH

```
1. if (|S_U| = 0)
2.   exit subroutine
3. end if
4. i ← select a node in S_U with the maximum path cost
5. while (true) do
6.    search in S_U for an upstream neighbor of node I
7.   if such a node can be selected
8.     i ← newly selected node
9.   else
10.    for k counts from 1 to |K|
11.       search in S_Sk for an upstream neighbor of node I
12.       if such a node is found in S_Sk
13.         move all previously selected nodes from S_U to S_Sk
14.         move all unselected peer and upstream neighbors of the selected nodes from S_U to S_Cj for ∀j ∈ K, j ≠ k
15.         exit subroutine
16.       end if
17.    end for
18.  end if
19. end while
```

Subroutine MAKE_SSTk_PATH

```
1. if (|S_Ck| = 0)
2.   exit subroutine
3. end if
4. i ← select a node in S_Ck with the maximum path cost
5. while (true) do
6.    search for an upstream neighbor of node i
7.    if such a node can be selected in S_Ck
8.      i ← selected node
9.      remove node i from S_Ck
10.   else if such a node can be selected in S_Sk
11.      place all the previously selected nodes to S_Sk
12.      move all unselected peer and upstream neighbors of the selected nodes from S_U to S_Cj for ∀j ∈ K, j ≠ k
13.      exit subroutine
14.   else if such a node can be selected in S_U
15.      i ← selected nodes
16.      remove node i from S_U
17.   else if such a node can be selected in S_Cj for ∀j ∈ K, j ≠ k
18.     i ← selected node
19.     remove node i from S_Cj
20.   else if such a node can be selected in S_Sj for ∀j ∈ K, j ≠ k
21.     i ← selected node
22.   end if
23. end while
```

In the iterative algorithm, all of the neighbors of a given node need to be searched after it has been selected for a particular SS-Tree (e.g., Line six of both subroutines MAKE_NEW_PATH and MAKE_SSTk_ PATH). So for a given WSN and assuming the WSN topology is implemented using adjacency lists, the worst case running time of the SS-Tree computation algorithm is in the order of $O(|V||D_{\max}|)$, where $D_{\max}$ denotes the maximum degree per node in the WSN topology. Otherwise, the running time complexity could reach $O(|V|^2)$ if an adjacency matrix is used. Similarly, the memory requirement for running this algorithm is in the order of $O(|V||D_{\max}|)$ when using adjacency lists, $O(|V|^2)$ for adjacency matrices.

5. SS-tree operational specifics and sleep scheduling

After SS-Trees are computed, the next major task is to determine an optimal sleep schedule that maximizes energy efficiency. Because of the phenomenally high transmission latency when using sleep schedules with short active period duration that only permits data to travel over one or a few hops per active period, the use of longer active periods is preferred so that all of the packet exchanges can be completed within fewer cycles for a given duty cycle. Furthermore, if all of the end-to-end data exchanges between the data sink and the nodes can be completed within a single active period, then network control functions such as time synchronization and sleep schedule updates can be implemented with less difficulty. While longer active period

lengths have the additional advantage of requiring less stringent time synchronization requirements, they will increase the amount of sleep time between two consecutive active periods, which in turn affects how often sensing applications can generate their data. Therefore, it is imperative to determine an upper bound of active period duration in order to balance the low communication duty cycle, monitoring sensitivity, and end-to-end packet transmissions.

5.1. Network layer routing

Since periodic link state updates for all of the sensor nodes are very expensive in terms of energy usage for large WSNs, a more energy-efficient packet delivery solution is preferred where different routing strategies may be employed to exploit the asymmetric upstream and downstream WSN traffic patterns. The proposed SS-Tree design streamlines the routing procedures by restricting individual sensor nodes to only maintain local connectivity information of its immediate 1-hop neighbors, whereas the data sink is given the sole right to compute the global network connectivity map from the link state information gathered from the sensor nodes.

Without the availability of global link state information nor geographical location through onboard GPS or radio ranging techniques due to the high costs involved, sensor nodes will rely on minimum cost forwarding [16] for sending packets to the data sink via its SS-Tree. Through locally exchanged connectivity information, each sensor node becomes aware of the cost of forwarding data packets, usually its hop count to the data sink, via each of its neighbors. On the other hand, the data sink can use source routing [17] for all types of downstream traffic, namely unicast, multicast, and broadcast, where the routing path is to be explicitly listed in the packet header. However, because of the potentially high header size for long end-to-end paths, source routing should be reserved only for special occasions such as network initialization and failure recovery. For more energy-efficient downstream dissemination, SS-Trees, which are essentially spanning tree structures, can be adapted for multicast and broadcast communications.

Since accurate time synchronization cannot be guaranteed in large WSNs, guard bands should buffer each active period to compensate for potential clock drifts along the entire path, as shown in Fig. 5. The first guard band, T_{G1}, compensates for common time synchronization errors amongst sensor nodes, clock drifts occurred during the preceding sleep period, and hardware switching times. After T_{G1}, the necessary packet forwarding activities would commence and should be completed within a length of T_{PP}, where the exact packet exchange sequencing will be discussed in Section 5.2. The second guard band, T_{G2}, accounts for timing overshoots of the packet forwarding period due to packet collisions and other unexpected events. Since T_{G1} is deterministic in nature as it depends on hardware selection and synchronization method, it would very much be predictable and likely to be the shortest in duration out of the three active period partitions. Both T_{PP} and T_{G2} depend on monitoring requirements, traffic patterns, processing overhead, and network topology, which makes calculating the perfect timing allocation for active periods difficult because of the high variability of the different factors. Therefore, the active period should be given a more liberal share of time in the initial sleep schedule and its length is to be adjusted dynamically in response to ongoing network performance measurements such as round-trip time and packet collision rate.

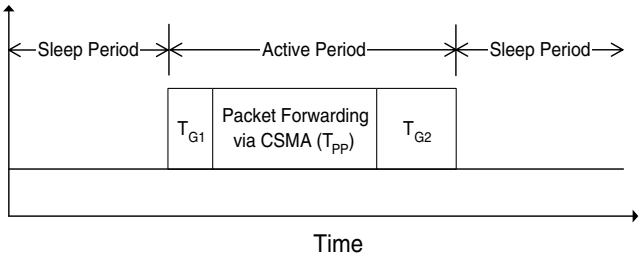


Fig. 5. Organization of active periods.

Fig. 6(b) illustrates a timing example for coordinating sleep schedules for a 3-hop routing path shown in Fig. 6(a), where the arrows at the bottom indicate potential transmission times at Node 3 for packets destined to Node 1 via Node 2. Arrows A and D are definitely dreadful timing choices for packet transmission because they reside in times when all of the nodes are asleep. Then around the start of a particular active period, all three nodes will wake up at roughly, but not exactly, the same time because of imperfections in time synchronization and clock drifts happened in the previous sleep period. The optimal transmission window occurs around arrow B, where all the nodes

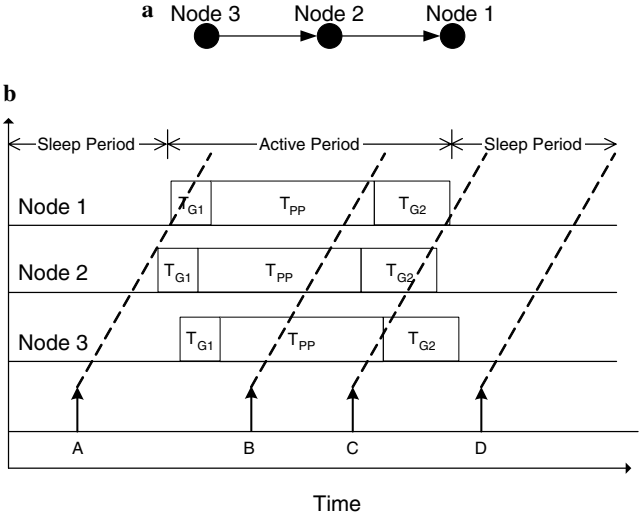


Fig. 6. Coordinated sleep scheduling for multihop routing paths. (a) Routing path configuration. (b) Sleep schedules and packet transmission timing.

have entered the active period and the packet should have enough time to traverse the two hops within T_{PP} as shown by the slanted dashed lines, assuming the assorted transmission delays are much shorter than the active period duration. Since arrow C begins transmission at the latter part of the active period, its delivery cannot be guaranteed within a single active period even with the extra buffering of T_{G2}. Therefore, the packet may have to be intermediately cached at Node 2 and forwarded to Node 1 at the next active period, which may be many minutes away. On the other hand, if sporadic packet losses can be tolerated, Node 2 can discard the packet at the end of the active period if it cannot be delivered to Node 1 in time, thereby cutting down energy usage. To prevent packet losses of this type, an accurate assessment of T_{PP} and T_{G2} as well as careful data traffic coordination in response to sleep schedules should be implemented.

5.2. Sensing requirements and traffic engineering

As briefly mentioned earlier, the nature of WSN data generation can be classified into three types: request-driven, event-driven, and timer-driven. For request-driven data reporting, a request is generated by the processing center to query specific nodes within the WSN for interested data [18]. Upon receiving the request packet, the sensor nodes would reply with the necessary data. Although the request and data exchange is simple, the extra overhead in executing the query can be significant in terms of energy consumption. If environmental data is to be recorded in regular intervals, then a better approach is to let the participating sensor nodes generate uplink data reports at times indicated by a timer without waiting for any query requests from the processing center. Timer-driven data reporting achieves limited energy savings by eliminating the need for downlink request notifications, but uncoordinated uplink message forwarding could offset any gains as periodic bursts of packets are concentrated within a short time frame, thus potentially causing excessive bit collisions and buffer overflows. Therefore, further energy savings are guaranteed only if meticulous nodal coordination in message forwarding can be established.

For WSNs engaging in surveillance applications, the primary objective is to trigger alarms whenever interested events are detected. Similar to timer-driven cases, event-driven data reporting does not require any request packets to initiate message forwarding. Given that very few messages would be generated by a subset of the entire sensor node population, the likelihood of energy wastage as a result of collisions and other adverse traffic effects is thus minimized. However, maintaining the real-time nature of event-driven data reporting implies that all of the sensor nodes should be constantly monitoring the wireless channel for incoming packets, which entails significant energy expenditure.

Since the objective of WSN applications is to oversee all sensor nodes cooperating together to execute common data reporting tasks instead of cater to thousands of terminals with individual QoS guarantees as in other types of networks such as the Internet and wireless LANs, more flexibility and control exist in manipulating application requirements and data flow patterns to suit dynamic operational situations. For example, in order to reduce hop-by-hop transmission time, each data reporting packet can shrink in size by formatting data types to represent Boolean answers (e.g., Is ambient temperature above 30 °C? → yes/no) rather than absolute values (e.g., What is the ambient temperature? → 25.75 °C). Besides speeding up data transmission, a smaller packet size would not require segmentation and reassembly services in lower layers, as well as decreases the instances of buffer overflow at intermediate nodes and reduces packet loss as a result.

In addition to compact query formats, aggressive data aggregation and duplicate suppression will be used for reducing unnecessary data packet exchanges. For example, suppose that each sensor node is required to report their circuitry well-being to the data sink every so often, where a positive response would indicate the node is alive and operational. Instead of generating separate packets, nodes on the same routing path can collectively rely on a single *seed reply* packet and transmit the same copy upstream, starting from the node furthest away from the data sink. If the node is experiencing no operational problems, then it will simply forward the incoming seed reply packet to its immediate upstream neighbor on the routing path without altering the packet's contents. Since the data sink possesses the global connectivity knowledge, receiving this single positive response packet would show that all the nodes along that path are faring well. Again, the SS-Tree structure would be ideal for performing in-network processing, and some timing coordination in data generation such as timer-driven data reporting would be very helpful in achieving efficient data aggregation and duplicate suppression.

While timer-driven data reporting is favored because of its simplicity and periodicity, event-driven and request-driven types should also be accommodated within the traffic engineering framework. In fact, WSNs for surveillance applications should give a high priority to event-driven data such as intrusion detection and abnormal data readings such that their deliveries can be expedited to the data sink with high fidelity at minimal packet loss. In contrast, occasional losses of periodic data readings and hardware status reports, though also undesired, would not seriously jeopardize surveillance capabilities. Still, such packet losses, regardless of request or timer-driven nature, do signify possible nodal failures or traffic load imbalance that require the immediate attention of the data sink for network modifications or repairs.

In face of aggressive data aggregation and duplicate suppression in the proposed WSN design, providing end-to-end ACKs in the transport layer by the data sink for regular data reporting packets is rather unnecessary, especially under an ultra-low communication duty cycle. Also, buffer

overflow, the common culprit in causing packet loss in the Internet, is deemed virtually non-existent in the envisioned WSNs because of small packet size and low traffic volume, thereby drastically diminishes the role of end-to-end ACKs. Therefore, energy expenditure can be further reduced by advocating the use of hop-by-hop ACKs in the MAC layer instead of end-to-end ACKs in the transport layer. However, hop-by-hop ACKs cannot always guarantee packet delivery success because a packet will get stranded on an intermediate node due to the premature expiration of the active period or a path blockage from nodal failures upstream. To limit energy use and to simplify the recovery process, packets with low priority such as periodic sensor readings and hardware status reports will be discarded, while high priority packets such as intrusion alert messages will be cached and forwarded to the data sink at the next available active period. If the processing center indeed wishes to obtain the data contained in the discarded packets, it can always reissue data requests to the corresponding sensor nodes. Given the assertion of meticulous sleep schedule planning and the low probability of nodal failures occurred during the short active period, such blatant packet discards will likely be uncommon.

Using low-rate communications, each active period could last up to several seconds or more to let packets pass through the entire path without obstruction, which implies that the temporal separation of active periods will be much longer, perhaps 15 min or more, in order to maintain the low communication duty cycle. Consequently, data reports for environmental monitoring and network maintenance cannot be generated at high frequencies, and some delay will need to be tolerated between report generation by the sensor nodes and actual reception at the data sink. Similarly, event-driven data packets indicating emergency events during the sleep period will be forwarded to the data sink at the first available active period with high priority either via the node's SS-Tree or that of neighboring nodes. Fig. 7 shows those events triggers not coinciding with active periods are to be relegated to the next available transmission slot, thereby introducing the need of buffering at the involved sensor nodes and some overall reporting delay. Apparently, there exists a trade-off between providing timely notifications of emergency events and extending overall system lifetime.

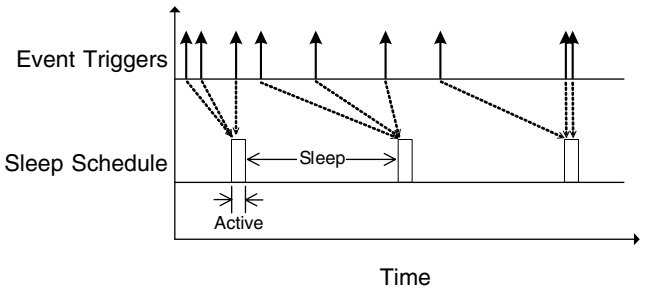


Fig. 7. Delays in event-driven reporting due to sleep scheduling.

Unlike event-driven data reporting, the reporting frequency of timer-driven types can be preset so that it coincides with each active period, depending on application requirements. After every node along a routing path wakes up at the start of each active period, intuitively the node with the highest hop count would automatically send a seed reply packet to initiate the data aggregation and duplicate suppression processing along the path. However, since the WSN may encounter unexpected nodal failures, experience considerable clock drifts, or detect new sensor node additions during the long sleep periods, the first task the sensor nodes should perform at the start of the active period is to assess any changes in the neighborhood topology and to keep each other's onboard clock in sync. Still, even the simple act of exchanging *Hello* and *Sync* messages with neighbors will occupy a sizable portion of the active period, not to mention the possibility of further delays due to packet collisions with legitimate data packets and other control traffic.

To better streamline the active period initialization process, the data sink should assume a much more involved role in coordinating topology maintenance and time synchronization functions, and the following example illustrates the proposed approach in doing so. After the start of the active period, the data sink should send a network probing packet called a *token* down every routing path to detect any link breakages, as shown in Fig. 8(a). At the tree junction points, the token will be broadcast to all downstream nodes, effectively splitting the single token into multiple copies to be pushed down each branch. When a node cannot reach its immediate downstream neighbor to forward the token, this would indicate a nodal failure has occurred due to depleted battery, hardware malfunction, off-sync sleep scheduling, or worsened radio conditions. The node should then instantly report back the failure discovery to the data sink, where the appropriate recovery procedures will be undertaken but are beyond the scope of this paper. Otherwise if every node is functioning properly, then the nodes at the fringes of the network will transmit back seed reply packets right after they receive the tokens, as in Fig. 8(b), where upstream and junction nodes will perform traffic merging and in-network processing on

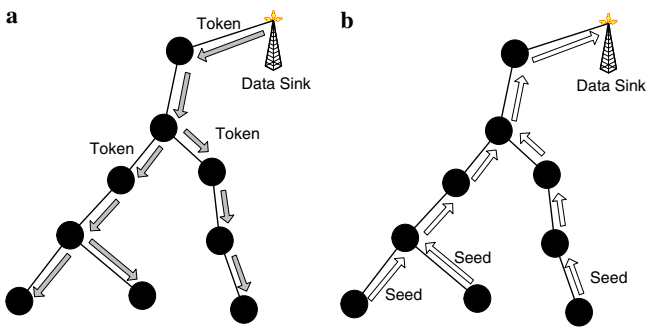


Fig. 8. Push–pull traffic sequencing for control and data packets in active period. (a) Downstream token. (b) Upstream seed reply.

the seed reply packets. Similarly, since time synchronization packets have to be periodically distributed throughout the WSN to offset clock drifts, they can be combined with the tokens to minimize transmission overhead, and the resultant packet size will be very short for accelerated forwarding and processing as they only contain a small packet header and a global timestamp. The act of sending the tokens downstream and the subsequent response of the seed reply packets will be referred to as the push–pull traffic sequencing.

In addition to accommodating control packets for failure detection and time synchronization, push–pull traffic sequencing can certainly incorporate other control message types, such as data requests, link state updates, low battery notification, and sleep schedule updates, with slight procedural changes. Normally during steady state operations, these control messages are to be mixed in with upstream data reporting traffic to contend for limited bandwidth during the short active periods. Since these control packets are vital in maintaining monitoring capabilities and network connectivity, any transmission delay or packet loss should be minimized or avoided if possible. For example, a sleep schedule update packet would become stale and useless if its delivery is bogged down by packet collisions and channel access delays. To minimize delays, data requests and sleep schedule updates can be piggy-backed or even incorporated into the token packets as well at the start of the active period. Although this may increase transmission and processing overhead if these additional control messages are not intended for a broadcast or multicast audience, their infrequency in generation deftly offsets the negative effects. Similarly, infrequent uplink control messages such as link state updates and low battery notification as well as timer-driven and request-driven data packets can also latch onto the seed reply packet and be sent upstream as a data packet burst for maximum communication efficiency.

Fig. 9 shows the partitioning of the T_{PP} portion of the active period into two phases, where each part has enough time allocated for the push–pull traffic sequencing to traverse end-to-end on the longest routing path in either downstream or upstream direction, with the exact timing demarcation of the two halves not explicitly defined. The push–pull traffic sequencing begins when the data sink sends the token downstream T_{G1} seconds after the start of the active period. The direction of traffic flow on the SS-Tree reverses sometime in between when the fringe nodes receive the tokens and respond with the seed reply packets. Hopefully all of the data reporting activities can be completed before the active period encroaches into the T_{G2} buffering area. The advantage of having downstream control and data request traffic preceding upstream data reporting and status update traffic is that the former can always verify entire path is free of node failures via forwarding tokens, thereby performing data forwarding and network maintenance simultaneously. Also, since the simultaneous convergence of unorganized downstream and upstream traffic over the WSN will inevitably raise the likelihood of packet collisions, allowing the packet traffic to only flow in one direction during a particular time slot would greatly alleviate the packet collision problem, though the improvement may come at the expense of potential channel under-utilization, which is of lesser priority in WSN design.

On the other hand, the use of push–pull traffic sequencing makes timer-driven data reporting behaving much like request-driven data reporting running on a regular schedule. Instead of letting sensor nodes spontaneously transmit data packets according to a predetermined timer, the tokens transmitted by the data sink in the push phase act as data requests to solicit the data reporting packets to be sent upstream in the pull phase, which happens periodically according to the sleep schedule. While timer-driven data reporting incurs fewer messaging overhead because of its periodicity and the absence of downlink control messages, keep in mind that other network control aspects such as routing path integrity, time synchronization, and sleep scheduling have to be accounted for in the overall design. As a result, a compromise is reached to incorporate data reporting tasks and network maintenance functions into the push–pull traffic sequencing model, where both will adhere to a periodic sleep schedule but with the timer-driven data generation spontaneity notion removed.

With respect to end-to-end delivery reliability in face of premature ending of the active period, lost downstream control packets can always be resent by the data sink if no seed reply has been received. However, for sleep schedule updates, since some packet loss would cause nodes on the same routing path to lose partial connectivity, nodes involved in updating sleep schedules will not enact upon them until the nodes themselves receive the seed reply from downstream to confirm the receipt of updated sleep sched-

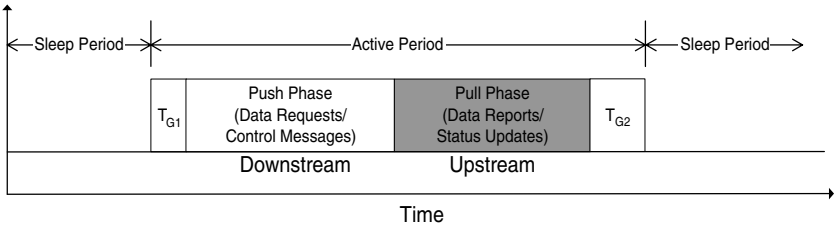


Fig. 9. Active period time slot partitioning for push–pull traffic sequencing.

ules of downstream peers. On the other hand, upstream control messages will need to be cached in intermediate nodes if they cannot be delivered within a single active period since no upstream end-to-end transport mechanism is to be implemented. Nevertheless, to prevent packet loss from happening at all, the active period should be allocated with plenty of time to let every packet to be delivered within a single active period.

5.3. Medium access control and sleep scheduling

Due to the extra messaging overhead in maintaining accurate time synchronization and managing channel assignment in face of the low communication duty cycle and hardware cost restrictions, single-channel unslotted CSMA is preferred over contentionless methods such as FDMA, TDMA, and CDMA for channel access during active periods of the sleep schedule because of its simplicity, greater scalability, and looser time synchronization requirements. While the RTS/CTS mechanism in CSMA/CA is effective in preventing the hidden node problem during channel contention, it increases the overall end-to-end propagation delay which in turn affects the monitoring sensitivity, especially when the length of data reporting packets is less than RTS packets [15]. Because of the long end-to-end propagation delay and the low volume of traffic in the envisioned WSN for wide-area surveillance, steady state data exchanges can bypass the RTS/CTS handshake as traffic management techniques discussed in Section 5.2 effectively reduces packet collisions through streamlining overall data traffic flows.

Putting all of the cross-layer considerations described in Sections 5.1 and 5.2 together, the timing components that constitute a single active period shown in Fig. 5 can be determined. Assuming the duration of the active period, T_{AP}, is the same for each SS-Tree, then T_{AP} can be represented by

$$T_{AP} = T_{G1} + T_{PP} + T_{G2}, \quad (1)$$

where T_{G1} accounts for clock drifts, time synchronization errors and hardware switching times, T_{PP} deals with time expended during push–pull traffic sequencing, and T_{G2} buffers the any timing overshoots. For standard crystal oscillators with well-known time synchronization methods, T_{G1} is largely deterministic. On the other hand, T_{PP} can be approximated according to round-trip time calculations at the data sink during the *Network Initialization* phase such that

$$T_{PP} \cong \max_{i \in V}(\mathrm{RTT}^i), \quad (2)$$

where RTT^i is the round-trip time recorded for node i on its respective SS-Tree. Due to the use of downstream flooding and upstream minimum cost forwarding over a large and dense WSN, the initially collected RTT values may not reflect a concise round-trip time measurement since it includes delays caused by random back-off timers and packet collisions. This timing inaccuracy will affect how T_{G2} is determined because the purpose of T_{G2} is to compensate for all the abnormalities during push–pull traffic sequencing such that packet loss due to premature completion of the active period can be minimized. While a longer T_{G2} will certainly diminish the chances of encountering timing overshoots, it will reduce monitoring sensitivity such that the event reporting windows will appear less frequently for adjacent SS-Trees. On the other hand, constant fine-tuning of the sleep schedule through issuing sleep schedule update packets downstream would produce high messaging overhead, thereby consuming considerable amounts of energy. Therefore, empirical *LRTT* measurement data should be complemented with some mathematical guidelines for calibrating the timing allocation of T_{PP} and T_{G2}. The ultimate goal is to minimize T_{AP} in order to increase the monitoring sensitivity while ensuring push–pull traffic sequencing and event reporting can be accomplished within a single active period.

To produce a rough mathematical estimate of T_{PP}, a routing path of N_h hops is first subdivided into single hops, where the sources of delay in packet delivery over each hop are further decomposed into individual components, as shown in Fig. 10 [19]. In a 1-hop transmission, the delay components are:

- *Preparation*: Before actually sending the packet, some time is spent by the sender in handling software commands and setting hardware interrupts for data preparation. Its nature is highly variable as it depends on the type of Operating System (OS) software, packet type, and packet size.
- *Channel access*: Since no RTS/CTS scheme is to be used because of its high messaging overhead and drastic increase in end-to-end delays, this component can be much reduced as the node instantly gains access to the wireless channel, assuming no ensuing packet collisions. On the other hand, channel contention introduces some variability in the time used, which increases along with the number of immediate neighbors.
- *Transmission*: This largely deterministic component concerns the time needed to transmit every bit of the packet through the sender's radio transceiver, which can be estimated using radio speed and packet size.
- *Propagation*: This deals with the minute amount of time needed for each bit to traverse the wireless link from sender to receiver, which is negligible in comparison to other delay components.
- *Reception*: This refers to the time spent in receiving every bit from the wireless channel and reconstructing the packet for further processing, which is also mainly deterministic as it depends on radio speed and packet size.

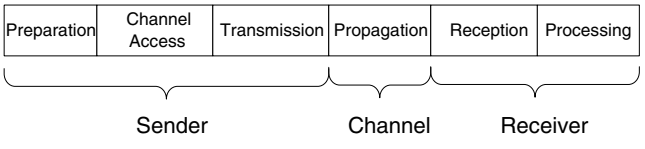


Fig. 10. Sources of delay in packet delivery over wireless link.

- *Processing*: After the validity of the received packet has been verified, the processor will decode the packet information and decide on appropriate actions. This last timing attribute is highly variable as it depends on the packet type and the software commands executed during processing.

Even though uncertainties exist in the choice of processor, OS software and radio components to be implemented on the sensor nodes, the control or data packets (C/D packets) traversing the WSN are likely to be extremely short (<10 bytes) in order to minimize preparation, transmission, reception, and processing delays. The short C/D packet size, along with topology simplification through SS-Trees and aggressive traffic engineering, also helps to drastically reduce the likelihood of packet collisions. On the other hand, for an SS-Tree whose branches are shaped in the form of linear chains, the cause of packet loss is mostly due to channel errors during the push–pull traffic sequencing.

Fig. 11(b) demonstrates the timing sequence of a simple Control/Data (C/D) packet exchange along a multihop 3-node chain shown in Fig. 11(a), where a data-acknowledgement-timeout mechanism ensures safe packet delivery in face of poor radio channel conditions. Due to the open wireless medium, the dotted arrows indicate overhearing by adjacent nodes of packets not intended for them. When a corrupted C/D packet is received by Node 3, as shown in Fig. 11(c), no ACK will be sent back to Node 2. As a result, another copy of the C/D packet is transmitted by Node 2 after a timeout period. Summarizing all of the associated delays shown in Fig. 10 for analyzing timing delays in Fig. 11, let the total time for processing and delivering each C/D packet be T_{CD}, the total time for processing and delivering a single ACK be T_{ACK}, and the duration of each timeout period be T_{TO}. If there is no packet loss present, then the time to complete the exchange over N_h hops on a linear chain, T_h, can be simply referred to as

$$T_h = N_h(T_{CD} + T_{ACK}). \tag{3}$$

When packet corruption occurs, the receiver will not generate an ACK and the sender would automatically retransmit the same C/D packet after a random timeout with mean value of T_{TO}. To analyze the impact of packet corruption on end-to-end propagation delay, let the generalized packet delivery success rate is p. Then the probability that the C/D packet transmission will be successful on the jth try can be represented as a simple geometric distribution. The expected number of tries before a successful delivery, E, can therefore be given by

$$E = \sum_{j=1}^{\infty} j(1-p)^{j-1}p = \frac{1}{p}. \tag{4}$$

Factoring in parameters T_{CD}, T_{ACK}, and T_{TO} and assuming zero correlation of packet delivery success rate for consecutive hops, the expected duration of each C/D packet delivery, E_{PD}, is

$$E_{PD} = \frac{2T_{CD} + T_{TO}}{p} - T_{TO} - T_{CD} + T_{ACK}. \tag{5}$$

For a regular push–pull traffic sequencing exchange with a downstream token and an upstream seed reply, suppose both types of packets are equally sized and require the similar preparation and processing time. Then the estimated RTT on a path of N_h hops, RTT_h, is

$$RRT_h = 2N_h\left(\frac{2T_{CD} + T_{TO}}{p} - T_{TO} - T_{CD} + T_{ACK}\right). \tag{6}$$

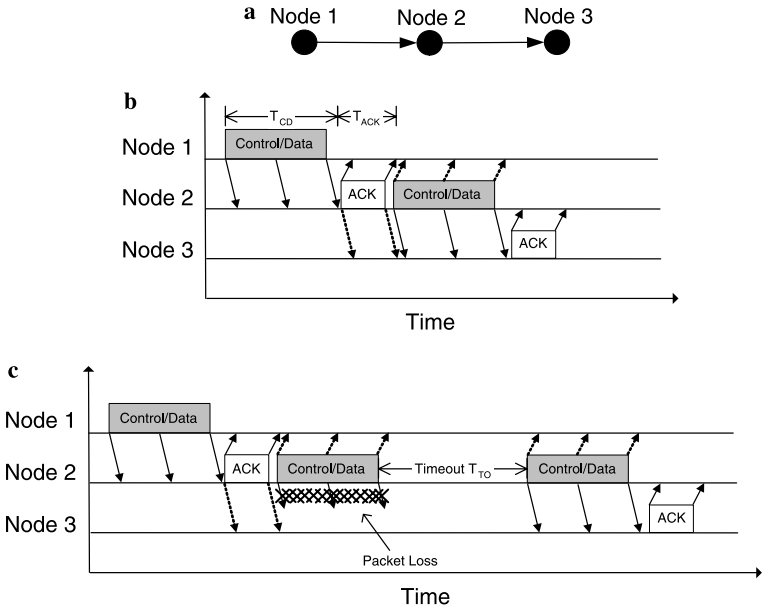


Fig. 11. C/D packet delivery delay analysis over multihop links. (a) Network configuration. (b) Timing diagram of a successful delivery. (c) Timing diagram of a successful delivery with one retransmission.

While any lost C/D packets can be easily recovered after a timeout, the retransmission mechanism will not be effective in an open multihop environment if each timeout period is too short to combat ACK packet losses. Fig. 12 shows how a single ACK packet loss at point A can trigger future packet collisions with the absence of the RTS/CTS mechanism. After Node 1 fails to receive an ACK from Node 2 for the correctly received C/D packet, it will retransmit another copy of the C/D packet after a short timeout at point B in which it also has to wait for a clear channel through carrier sensing. On the other hand, Node 2 has already forwarded the C/D packet to Node 3, where the corresponding ACK is replied at point C. Without using the inefficient RTS/ CTS mechanism to prevent the notorious hidden node problem, the retransmitted copy of the C/D packet will collide with the incoming ACK reply from Node 3, thus wasting the entire packet exchange sequence. Therefore, in order to prevent such circumstances from happening on a linear chain, the timeout period T_{TO} has to be at least

$$T_{TO} \geqslant T_{CD} + 2T_{ACK}. \quad (7)$$

Another concern with the open wireless medium is that additional care has to be taken when passing tokens and other packets downstream across SS-Tree junction points to bypass the hidden node problem. Figs. 13(a) and (b) respectively show a simple SS-Tree junction configuration and its corresponding timing diagram where a C/D packet originated at Node 1 will be broadcast at junction Node 2 to its downstream descendants Nodes 3 and 4. Afterwards, Nodes 3 and 4 will forward the C/D packet to their respective downstream descendents. Without any coordination in the acknowledgement sequence and assuming both Nodes 3 and 4 cannot detect each other through carrier sensing, the ACK packets from Nodes 3 and 4 would collide at Node 2. Since Node 2 subsequently cannot confirm if the C/D packet deliveries were successful or not, it has to rebroadcast the same packet after a timeout, thereby consuming more energy and creating even more potential collisions. Other than to minimize the number of junction points on the SS-Trees or resort to expensive RTS/CTS mechanism, solutions such as separating the C/D packet broadcasts into unicast links and introducing random timers for ACK replies are worth exploring further, though

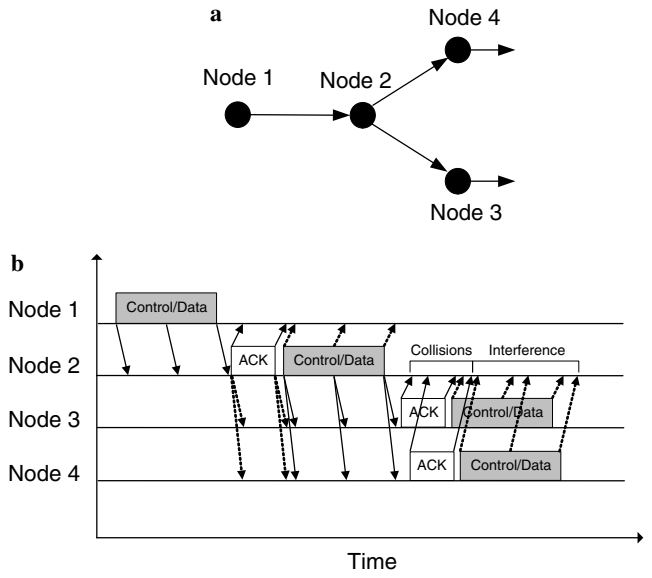


Fig. 13. Effects of packet collision at SS-Tree junction point. (a) Network configuration. (b) Timing diagram.

they may also increase propagation delay and control overhead without completely eliminating the hidden node problem.

Since the combination of wireless medium openness and multihop communications introduce the unpleasant effect of hidden terminal problem that will eventually lead to a decrease in monitoring sensitivity through the lengthening of active periods, it would be conducive to explore other approaches to expedite the push–pull traffic sequencing mechanism while maintaining hop-by-hop packet acknowledgements. One way is to implement *implicit acknowledgements* (IACKs), where an overheard data packet also acts as an acknowledgement when it is being forward along a path. Fig. 14 shows an example of mixing IACKs and explicit acknowledgements (EACKs) when passing a single C/D packet along the same short chain as in Fig. 11(a). After Node 1 passes the C/D packet to Node 2, Node 2 in turn will directly forward it to Node 3 without sending an EACK. Because of the open wireless medium, Node 1 will overhear the C/D packet sent to Node 3, thereby indirectly acknowledging its safe arrival earlier at Node 2.

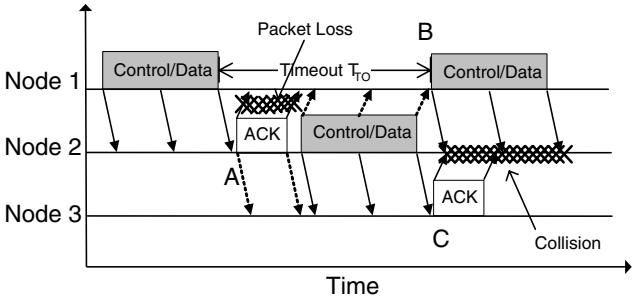


Fig. 12. Impact of ACK losses in face of short timeout periods over multihop links.

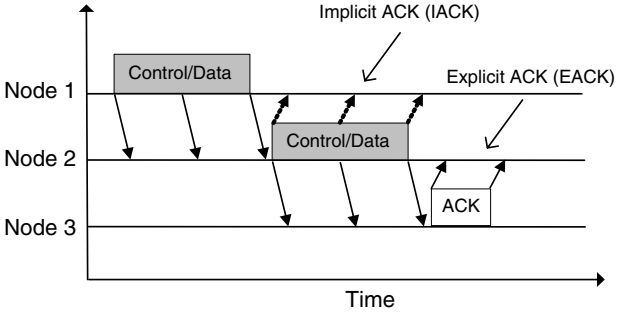


Fig. 14. Use of implicit ACK (IACK) and explicit ACK (EACK).

On the other hand, Node 3 will need to send an EACK back to Node 2 because it is at the end of the routing path and has no other neighbor to send the C/D packet to.

Both IACK/EACK modes can be explicitly set by toggling a particular control bit in the C/D packet header by the sender, and the receiver of the packet will respond accordingly. However, the decision to use either ACK mode also depends on routing path topology and application requirements, hence involving yet another kind of cross-layer considerations. In fact, instances where EACK must be used is when the C/D packet reaches the end of a routing path like the previous example, when the C/D packet has already been forwarded to the next hop, and when the packet is traveling through a junction point on the SS-Tree in the upstream direction where the parent needs to wait and perform data aggregation on incoming packets collected from different branches. Fig. 15 describes the second case on the same topology as in Fig. 11(a). Initially, Node 1 did not receive the initial IACK and retransmits after a timeout. Since the C/D packet has already been successfully passed from Node 2 to Node 3, Node 2 responds to the retransmitted C/D packet by an EACK back to Node 1.

To simplify the process of message passing over SS-Tree junction points, the upstream sender will treat the wireless multicast situation as multiple unicast links. Fig. 16 shows its operation on the same topology as in Fig. 11(a), but only this time Node 3 replies Node 2 with an EACK because it is made to have no downstream descendents to forward to. On the other hand, Node 4 acknowledges the C/D packet with an IACK back to Node 2.

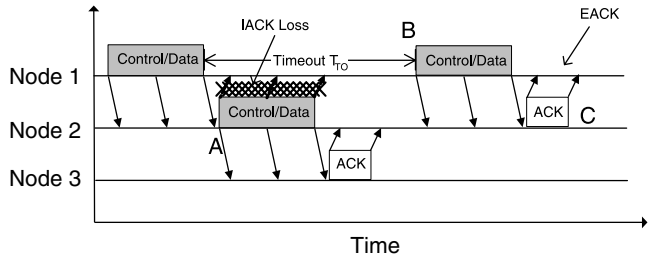


Fig. 15. Interchange of IACK and EACK in face of packet loss.

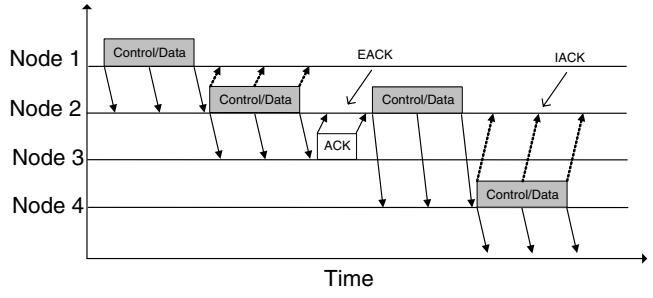


Fig. 16. Timing diagram of mixed IACK and EACK use at SS-Tree junction.

With mixed use of IACKs and EACKs, the time to complete the push–pull exchange over N_h linear hops, T'_h, can be reduced to

$$T'_h = N_h T_{CD}. \tag{8}$$

Similarly, the estimated RTT on a linear path of h hops with IACKs, RRT'_h, becomes

$$RRT'_h = 2(N_h - 1)\left(\frac{2T_{CD} + T_{TO}}{p} - T_{TO} - T_{CD}\right) + 2(T_{CD} + T_{ACK}). \tag{9}$$

To prevent the disruptive effects a corrupted IACK can cause in multihop communications, a new timeout value, T'_{TO}, is defined to be

$$T'_{TO} \geqslant 2T_{CD}. \tag{10}$$

Comparing Eqs. (7) and (10) shows the IACK mechanism would work better in reducing the time dwelled in push–pull traffic sequencing when the size of C/D packets is comparable to that of EACKs, which can be achieved through data reduction and data aggregation schemes mentioned earlier.

Since the traffic flow is light and coordinated during push–pull traffic sequencing, the likelihood of collision between IACKs and other packets is low. Still, with the introduction of spontaneous event-driven data to the push–pull traffic sequencing stream come increased chances of packet collisions and timing overshoots that exceed the T_{PP} period. To reduce the effects of event-driven data and random channel errors that can invalidate the IACKs, the original sender needs to only decode the first few bytes or just the header portion of the IACK and accept as legitimate packet acknowledgement when C/D packets are long. Nevertheless, both factors will affect the determination of the packet delivery success rate p. The exact derivation of p will also depend on the underlying wireless channel model assumptions, and this will be delegated as part of future work.

6. Performance evaluation – SS-tree computation

The proposed iterative algorithmic approach for computing SS-Trees is coded into ANSI C programming language and is tested by running simulation cases on a Dell Precision 450 machine with two 2.8 GHz Pentium 4 Xeon processors and 1 GB RAM. The preliminary SS-Tree computation results were performed under the following assumptions:

1. Two types of WSN topology are used: 8-neighbor square grid (8-N, see Fig. 1a) and 4-neighbor planar square grid (4-N). The number of nodes per grid edge ranges from 3 to 15 (i.e., the number of nodes in the WSN is the square of the number of nodes per grid edge). Further research into SS-Trees will consider the more practical cases of random and pseudorandom distributions of nodes.

2. The number of SS-Trees to be computed on each test topology ranges from 2 to 4 (2-SST–4-SST).
3. The data sink is represented as a node located at or near the center of the grid. Specifically, given n is the number of nodes per grid edge, then the coordinates of the data sink is $(0.5n, 0.5n)$ if n is even, $(0.5(n+1), 0.5(n+1))$ otherwise.
4. The link cost between adjacent nodes is 1, which implies that the total shortest path cost from each node to the data sink is simply its hop count.

6.1. Integrity of computated SS-trees

To evaluate the effectiveness of the iterative algorithm, several types of metrics are collected from the computed SS-Tree configurations on the first attempt in the different test cases and then presented into graphical form for easier comparison. In terms of computation speed, the iterative algorithm arrives at SS-Tree computation results at a very quick pace, typically in less than 1 s for all of the test cases. As an unpleasant trade-off for achieving faster solution times, however, the algorithm produces a considerable number of shared nodes. Fig. 17 shows the number of shared nodes computed in the different test cases using the iterative algorithm. In all of the test cases, the number of shared nodes computed increases more or less in a linear trend corresponding with the number of nodes per grid edge. The slight deviations along the linear trend can be attributed to the shifting of the data sink's coordinates in response to odd/even changes of the number of nodes per grid edge in individual test cases. Out of all test scenarios, cases with a 8-neighbor grid and 4 SS-Trees (8-N 4-SST) produce the most shared nodes, while scenarios with 2 SS-Trees (both 4-N 2-SST and 8-N 2-SST) generated the fewest shared nodes on average.

Another measure of how well SS-Trees are computed is the number of *protected* nodes, which refers to nodes with at least one non-co-SS-Tree neighbor. This measure is important as it indicates how large the event reporting window of each node will be and how well nodes can recover from failures with help from such non-co-SS-Tree neighbors. Despite the large number of shared nodes computed, the iterative algorithm does enhance the degree of neighbor-to-neighbor protection and therefore, increase the capabilities for event reporting and failure recovery of each node. Fig. 18 shows the proportion of nodes protected in each case, and all of the test cases achieved 100% node protection except for a couple of instances.

If a protected node is adjacent to neighbors that separately belong to all other SS-Trees, then this node is deemed *fully protected*, which means it possesses the most frequent event reporting window and it receives the maximum protection from neighbors during failure recovery. Fig. 19 shows the proportion of fully protected nodes in each test case. For a given topology, the degree of full protection decreases as the number of SS-Trees to be computed increases. On the other hand for a fixed number of SS-Trees, an increase in network density would also increase the degree of full protection. Cases of 8-N 2-SST configuration all achieved 100% protection, though at the expense of each node having multiple co-SS-Tree neighbors. This will lead to increased overhearing and packet collisions during steady state WSN operations. Notwithstanding WSN topologies with small number of nodes, degree of full protection remains quite stable or even increases along with the increase in nodal population. This is especially true for test cases with higher number of SS-Trees, where a larger and denser WSN topology permits

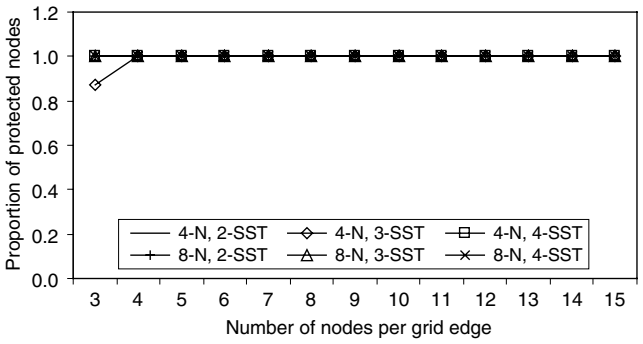


Fig. 18. Proportion of nodes protected in the test cases.

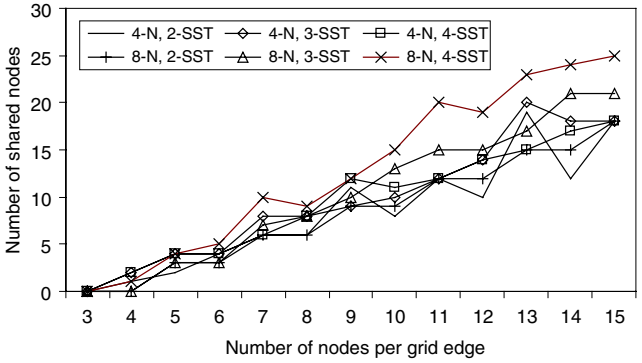


Fig. 17. Number of shared nodes computed in the test cases.

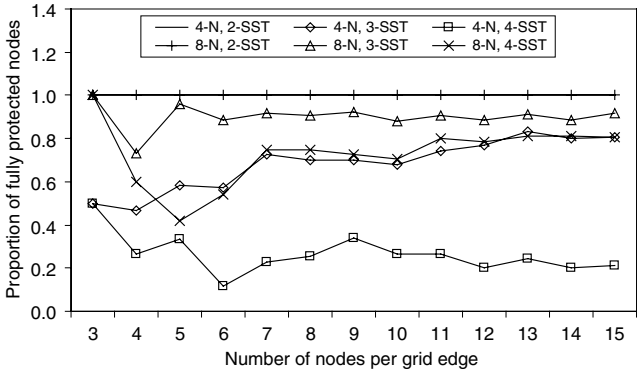


Fig. 19. Proportion of nodes fully protected in the test cases.

better SS-Tree construction. On the other hand, smaller topologies give less leeway in allowing multiple SS-Trees to become neighbors to a particular node. Again, careful design consideration should be given when balancing the optimal nodal deployment strategy, sensing requirements, wireless communication range and the number of SS-Trees involved.

6.2. WSN lifetime increase and energy utilization

To evaluate the expected lifetime of a given WSN topology, the standard approach is to first construct a number of SS-Trees and then generate simulated traffic over the network and measure the energy consumption until network connectivity is no longer sustainable or the sensing coverage becomes inadequate due to nodal failures. However, because of the complexity in building a suitable MAC-level simulation environment that can model system lifetime changes lasting several years for thousands of nodes, detailed packet transactions for message exchanges over the wireless medium cannot be simulated at this time. While a precise measure of the expected WSN lifetime is not yet available, a rough estimation can still be obtained through relating the expected lifetime to the number of times, or *rounds*, the iterative SS-Tree computation algorithm can be run consecutively over a given topology. The logic behind this is that after all the shared nodes have drained their batteries, the data sink will have to recompute the SS-Trees based on the revised WSN topology with the shared nodes removed, thereby extending the WSN lifetime. This SS-Tree recomputation process continues until less than 50% of all nodes remain connected to the data sink, which is the benchmark for evaluating the expected operational lifetime of a particular WSN.

Simulations on determining the expected operational lifetime and other related performance metrics of a given WSN are based the following idealized assumptions:

1. Every node begins operation with the same amount of battery power.
2. Battery depletion is the only cause for nodal failures.
3. The energy cost in recomputing SS-Trees is negligible compared to that incurred during normal WSN operations.
4. Because of efficient data aggregation and duplicate suppression procedures, the energy depletion rate for all nodes across the WSN is approximately the same.

If no shared nodes were ever computed on a given WSN, then theoretically the expected operational lifetime of the WSN is equal to the product of its normal operational lifetime under a particular duty cycle and the number of SS-Trees involved. Therefore, for test cases with four SS-Trees computed, the upper bound on expected lifetime becomes four times that of cases without using SS-Trees. Fig. 20 shows the expected increase in WSN operational lifetime, and from the numerical results it can be seen that denser

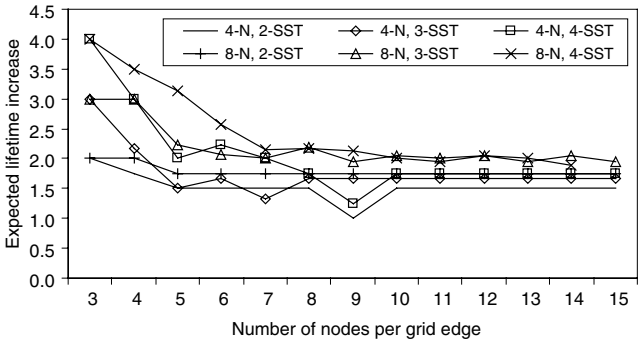


Fig. 20. Expected WSN lifetime increase through SS-Trees.

topologies and a higher number of SS-Trees will generally extend WSN lifetime longer. Still, only tiny WSNs (i.e., 3×3 to 4×4) are able to achieve the theoretical limit of expected lifetime since their SS-Tree configurations do not contain any shared nodes. On the other hand, a few cases do not yield any lifetime extension at all because the WSN topology becomes prematurely disconnected below the 50%-threshold in the first round of SS-Tree computation by shared nodes that appear at the right places, which leaves the rest of the nodal population with plenty of battery power left to waste. In fact, shared nodes that are concentrated at a close path distance to the data sink will lead to a shorter expected lifetime since they expedite the occurrence of topology disconnection.

Another useful metric for evaluating the effectiveness of the SS-Tree computation algorithm is energy utilization, which measures the total amount of energy expended across the WSN versus the total amount initial energy given to every node. Fig. 21 shows the degree of energy utilization occurred in the test scenarios, and again small WSNs are able to achieve maximum energy utilization since their SS-Tree configurations do not contain any shared nodes. On the other hand, the test scenario 8-N 2-SST is able to attain over 90% energy utilization in addition to excelling in extending WSN operational lifetime. Future research work using full-scale WSN data traffic simulations will verify this combination's prowess in computing SS-Trees by taking into account MAC-level effects such as overhearing and packet collisions as well as other degradation effects.

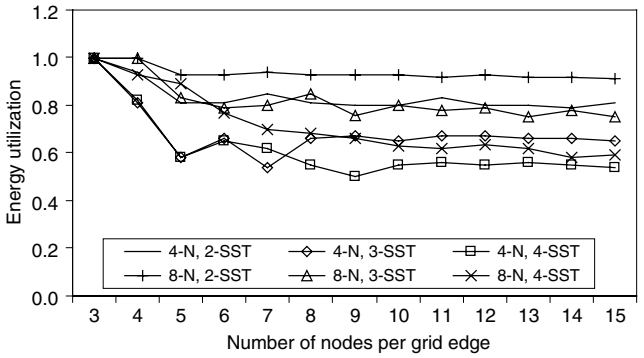


Fig. 21. Energy utilization of the WSN nodes before end of operation.

7. Performance evaluation – Sleep scheduling

The performance evaluation for sleep scheduling with respect to MAC dynamics is conducted on the same machine in Section 6 using a custom-built discrete event simulator written in ANSI C. By forgoing existing wireless network simulator choices (e.g., ns2, Glomosim, Omnet++, Qualnet, etc.), more flexibility can be achieved in manipulating traffic generation and MAC-layer signaling to better portray the cross-layer SS-Tree approach. Comparison is to be made between the normal MAC acknowledgement procedures without RTS/CTS and the proposed combined IACK/EACK approach described in Section 5.3 for minimizing the time used in end-to-end push–pull traffic sequencing under different WSN operating conditions. Table 1 shows the parameters to be used in the performance evaluation, whose definitions were given in Section 5.3.

For simplification, the WSN topology used for performance evaluation is assumed to be a linear path with no junction points, where the hop count is determined by parameter N_h in each test case. RTT measurements are to be taken at the data sink, which is at one end of the linear path, for the amount of time to execute pull–pull traffic sequencing from end to end. Each test case is run 10 times in Monte Carlo style and final results are taken as the average of all recorded data. Simulation data are in turn compared with analytical results obtained from computing Eqs. (6) and (9). Given the various transceiver choices and operational requirements in WSN design, generalized time units are used in time measurements for better independence from actual data rate and packet size. For example, time to send a C/D packet, T_{CD}, is given as 1, 2, or 5 time units, which can be easily converted to metric units for a given modulation scheme and packet length. Timeout values for EACK and IACK schemes, T_{TO} and T'_{TO}, are equal to the values given in Eqs. (7) and (10), respectively.

7.1. Packet loss effects

The following test cases demonstrate the effects in varying p while T_{CD} is fixed at 2. The selected values of p of 0.99, 0.95 and 0.9 are adequate for simulating wireless channel conditions with bit error rates from 10^{-3} to 10^{-6} given that very short packets are passed within the WSN. Figs. 22 and 23 show how different p values affect the eventual RTT measurements in EACK only and IACK/EACK schemes, respectively, where the dotted and solid lines

Table 1
Parameters for sleep scheduling performance evaluation

Parameter	Range
N_h	2–10
T_{CD}	1, 2 and 5
T_{ACK}	1
T_{TO}/T'_{TO}	See Eqs. (7) and (10)
p	0.9, 0.95 and 0.99

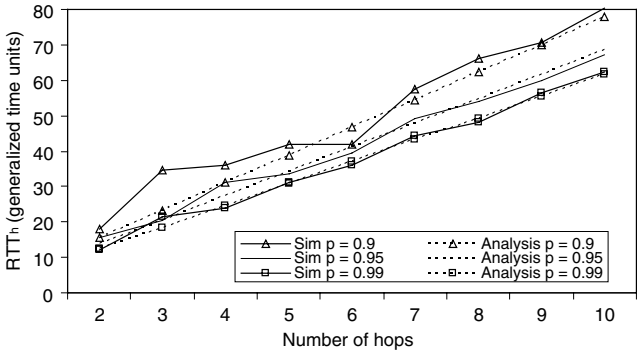


Fig. 22. Packet loss effects in EACK only scheme.

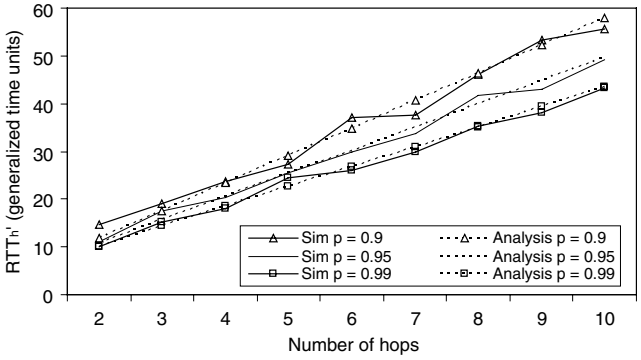


Fig. 23. Packet loss effects in IACK/EACK scheme.

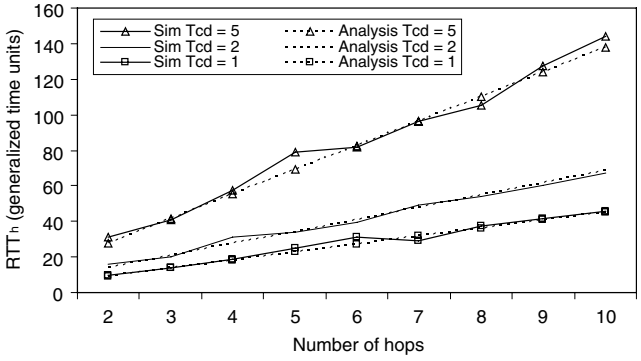


Fig. 24. Effects of packet length variations in EACK only scheme.

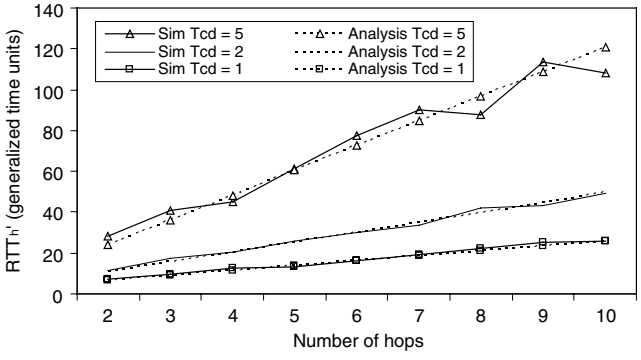


Fig. 25. Effects of packet length variations in IACK/EACK scheme.

denote analytical and simulation results, respectively. Simulation data figures adhere well to the linear profiles of the analytical results in all test cases with less than 3% difference except for the EACK only scheme at $p = 0.9$, where the deviation is over 10%. A closer inspection on the actual packet exchange log revealed that more frequent instances of corrupt ACK packet at $p = 0.9$ created havoc in the MAC signaling as noted in Fig. 12, thus leading to timing prolongation and uncertainties in the eventual RTT measurements.

In both schemes, RTT measurements increase along with packet losses because of the extra time spent recovering from packet losses. However, when compared to the EACK only scheme, the IACK/EACK scheme achieved an over 25% reduction in the time required for running push–pull traffic sequencing for all p values, thereby increasing monitoring sensitivity through scheduling shorter and more frequent active periods for each SS-Tree. Conversely, the amount of time used in the active period for the EACK scheme can remain the same, but the extra 25% of time can be designated as the T_{G2} portion of the active period as depicted in Fig. 9 and Eq. (1) for better protection against all the abnormalities and timing overshoots during push–pull traffic sequencing, as well as increasing the available time to safely deliver event-driven data to the data sink.

7.2. Packet length variations

Previously, a 2:1 relationship is assumed in the time used for processing and delivering ACK and C/D packets. Therefore, it would be helpful to investigate the effects on RTT measurements with respect to changes to this ratio. The following test cases study the effects in changing T_{CD} (i.e. the length of C/D packets) with p fixed at 0.95 and T_{ACK} set at 1, and the results are shown in Figs. 24 and 25 for both acknowledgement schemes, respectively. T_{CD} takes on values of 1, 2, and 5, and its relationship to T_{ACK} will determine how much control information and data can be incorporated into C/D packets without affecting active period scheduling and monitoring sensitivity.

From both Figs. 24 and 25, it is apparent that RTT measurements increase at a faster rate as T_{CD} becomes greater. The reasoning is that since the timeout value T_{TO} increases along with T_{CD}, any increase of T_{CD} will put double pressure to increase the RTT values. On the other hand, increasing T_{CD} will diminish the performance advantage achieved by the IACK/EACK scheme as its main feature of reduction in explicit ACK use become a lesser factor in influencing RTT values when C/D packets get longer. Specifically, the timing reduction of using the IACK/EACK scheme is nearly 40% over the EACK only approach when $T_{CD} = 1$, whereas is the same performance metric drops to about 25% for $T_{CD} = 2$ and then further to only about 12% for $T_{CD} = 5$. Therefore, WSN designers using SS-Trees need to keep in mind of the ramifications of trading off C/D packet size for sleep scheduling and monitoring sensitivity.

8. Conclusions and future work

This paper discussed a number of cross-layer design issues that are pertinent to the implementation of a mesh-based wireless sensor network for wide-area surveillance applications. In light of the constraining effects of an ultra-low communication duty cycle in the range of 1%, a sleep scheduling-based organizational approach called SS-Trees is suggested to minimize energy usage while providing sufficient monitoring capabilities. An iterative algorithm based on a greedy depth-first bottom-up approach is proposed to tackle the core problem of determining how the sensor nodes can assigned to a fixed number of SS-Trees on a given WSN topology. An MAC-layer implicit acknowledgement scheme is suggested to maximize monitoring sensitivity in the subsequent sleep scheduling. Performance evaluation has shown that the proposed approach is capable of computing SS-Trees that adhere to the original goals, albeit to various degrees. Specifically, the iterative algorithm approach returns a solution quickly with excellent inter-node protection, but it produces a higher number of shared nodes. On the other hand, sleep scheduling is better served when combined with the IACK/EACK scheme at the MAC layer and the push–pull traffic sequencing concept because of their better adaptation to the application-specific multihop WSN environment.

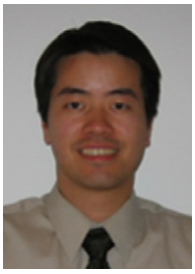
In the next stage of research, the existing approach for computing SS-Trees and sleep schedules will be applied over a larger variety of network configurations and more realistic wireless channel models for further performance improvements. In addition, the following list of issues will be explored:

1. For a given random topology, what is the maximum number of SS-Trees that can be constructed to minimize the number of shared nodes?
2. For a given number of nodes, what is the optimal method of deployment that ensures 100% coverage of the subject area while maximizing the number of available SS-Trees with minimum number of shared nodes?
3. What are the suitable neighborhood discovery and failure recovery strategies for the SS-Tree design?

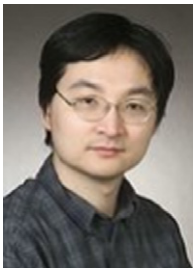
References

[1] C.-Y. Chong, S.P. Kumar, Sensor Networks: Evolution, Opportunities, and Challenges, Proc. IEEE 91 (8) (2003) 1247–1256.
[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on wireless sensor networks, IEEE Commun. Mag. 40 (8) (2002) 102–114.
[3] G.J. Pottie, W.J. Kaiser, Wireless integrated network sensors, Commun. ACM 43 (5) (2000) 51–58.
[4] W. Ye, J. Heidemann, D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, Proc. IEEE INFOCOM 3 (2002) 1567–1576.
[5] T. van Dam, K. Langendoen, An adaptive energy-efficient MAC protocol for wireless sensor networks, Proc. ACM SenSys'03 (2003) 171–180.
[6] M.L. Sichitiu, Cross-layer scheduling for power efficiency in wireless sensor networks, Proc. IEEE INFOCOM (2004).

[7] C.-F. Hsin, M. Liu, Network coverage using low duty-cycled sensors: random and coordinated sleep algorithms, Proc. ACM IPSN (2004) 433–442.
[8] A. Boukerche, X. Cheng, J. Linus, Energy-aware data-centric routing in microsensor networks, Proc. ACM MSWiM'03 (2003) 42–49.
[9] U. Cetintemel, A. Flinders, Y. Sun, Power-efficient data dissemination in wireless sensor networks, Proc. ACM MobiDE'03 (2003).
[10] W.B. Heinzelman, A.P. Chandrakasan, H. Balakrishnan, An application-specific protocol architecture for wireless microsensor networks, IEEE Trans. Wireless Commun. 1 (4) (2002) 660–669.
[11] A. Manjeshwar, D. Agrawal, TEEN: a routing protocol for enhanced efficiency in wireless sensor networks, in: Proceedings of the International Parallel and Distributed Processing Symposium, 2001, pp. 2009–2015.
[12] S. Lindsey, C. Raghavendra, K.M. Sivalingam, Data gathering algorithms in sensor networks using energy metrics, IEEE Trans. Parallel Distributed Syst. 13 (9) (2002) 924–935.
[13] K. Du, J. Wu, D. Zhou, Chain-based protocols for data broadcasting and gathering in the sensor networks, Proc. Int. Parallel Distributed Proc. Sym. (2003).
[14] P.-J. Wan, K.M. Alzoubi, O. Frieder, Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks, Mobile Networks and Applications, Vol. 9, Kluwer Academic Publishers, 2004, pp. 141–149, issue 2.
[15] ANSI/IEEE Std. 802.11 1999 Edition, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE, Piscataway, NJ, 1999.
[16] F. Ye, A. Chen, S. Liu, L. Zhang, A scalable solution to minimum cost forwarding in large sensor networks, in: Proceedings of the Seventh Annual International Conference on Computer Communications and Networks, 2001, pp. 304–309.
[17] C.E. Perkins, E.M. Royer, S.R. Das, M.H. Marina, Performance comparison of two on-demand routing protocols for ad hoc networks, IEEE Pers. Commun. Mag. 8 (1) (2001) 16–28.
[18] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, F. Silva, Directed diffusion for wireless sensor networking, IEEE/ACM Trans. Networking 11 (1) (2003) 2–16.
[19] S. Ganeriwal, R. Kumar, M.B. Srivastava, Timing-sync protocol for sensor networks, Proc. ACM SenSys'03 (2003) 138–149.



Rick W. Ha received his B.A.Sc. in Computer Engineering and M.A.Sc. in Electrical and Computer Engineering in 2000 and 2002, respectively, from University of Waterloo, Canada, where he is currently pursuing his Ph.D. degree. His research interests include cross-layer design of wireless ad hoc and sensor networks.



Professor Pin-Han Ho received his B.Sc. and M.Sc. Degree from the Electrical and Computer Engineering Department of National Taiwan University in 1993 and 1995, respectively. He started his Ph.D. study in the year 2000 at Queen's University, Canada, focusing on optical communications systems, survivable networking, and QoS routing problems. He finished his Ph.D. in 2002, and joined the Electrical and Computer Engineering Department of University of Waterloo, Canada, as an Assistant Professor in the same year. Professor Ho is the first author of more than 40 refereed technical papers and book chapters, and the co-author of a book on optical networking and survivability. He is a recipient of Early Researcher Award (ERA) in 2005.

Xuemin (Sherman) Shen received his B.Sc. degree (1982) from Dalian Maritime University (China) and his M.Sc. (1987) and Ph.D. degrees (1990) from Rutgers University, New Jersey (USA), all in Electrical Engineering. From September 1990 to September 1993, he was first with the Howard University, Washington D.C., and then the University of Alberta, Edmonton (Canada). Since October 1993, he has been with the Department of Electrical and Computer Engineering, University of Waterloo, Canada, where he is a Professor and the Associate Chair for Graduate Studies. Dr. Shen's research focuses on mobility and resource management in interconnected wireless/wireline networks, UWB wireless communications systems, wireless security, and ad hoc and sensor networks. He is a coauthor of two books, and has published more than 200 papers and book chapters in wireless communications and networks, control and filtering. Dr. Shen was the Technical Program Co-Chair for IEEE Globecom'03 Symposium on Next Generation Networks and Internet, ISPAN'04, IEEE Broadnets'05, QShine'05, IEEE WirelessCom, and is the Special Track Chair of 2005 IFIP Networking Conference. He serves as the Associate Editor for IEEE Transactions on Wireless Communications; IEEE Transactions on Vehicular Technology; ACM/Wireless Networks; Computer Networks; Wireless Communications and Mobile Computing (Wiley); and International Journal of Computers and Applications. He also serves as Guest Editor for IEEE JSAC, IEEE Transactions Vehicular Technology, IEEE Wireless Communications, and IEEE Communications Magazine. Dr. Shen received the Outstanding Performance Award from the University of Waterloo in 2004, the Premier's Research Excellence Award (PREA) from the Province of Ontario, Canada for demonstrated excellence of scientific and academic contributions in 2003, and the Distinguished Performance Award from the Faculty of Engineering, University of Waterloo, for outstanding contribution in teaching, scholarship and service in 2002. Dr. Shen is a registered Professional Engineer of Ontario, Canada.