IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received July 20, 2017, accepted September 13, 2017, date of publication September 21, 2017, date of current version October 25, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2755121

Identifying Energy Holes in Randomly Deployed Hierarchical Wireless Sensor Networks

AYESHA NAUREEN[1], NING ZHANG[1], AND STEVE FURBER[2]
[1]Information Management Research Group, School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K.
[2]Advanced Processor Technology Research Group, School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K.

Corresponding author: Ayesha Naureen (ayesha.naureen@manchester.ac.uk)

ABSTRACT This paper proposes a novel protocol, called an aggregation-based topology learning (ATL) protocol, to identify energy holes in a randomly deployed hierarchical wireless sensor network (HWSN). The approach taken in the protocol design is to learn the routing topology of a tree-structured HWSN in real-time, as an integral part of the sensed data collection and aggregation process in the network. The learnt topology is then examined to identify high energy-consuming nodes, that are more likely to create energy holes in the network. The major challenge in designing this protocol is to code topology data in such a way that it can be carried in length-constrained messages supported by current sensor technologies. To address this challenge, three topology coding methods are proposed. A theoretical analysis of the three topology coding methods is carried out to find the optimum method among the three, and this optimum method is used in the ATL protocol. The ATL protocol is tested and evaluated on a real WSN test bed in terms of completeness, correctness and energy costs. Based on the evaluation results, we have identified two classes of high energy-consuming nodes, which are: 1) nodes that carry topology data from more downstream nodes and 2) nodes that more frequently switch between different upstream nodes. This finding is significant as it provides an insight as how topology-learning, as well as data collection, may be used to prolong the life-time of a HWSN. In addition, the evaluation results also show that the energy cost incurred in a data collection process integrated with our proposed topology-learning facility is at a similar level as for the process without the facility, thereby implying that the cost incurred in topology-learning by using our proposed method is negligible. These findings indicate that, by integrating the topology-learning process with the sensed data collection and aggregation process, the ATL protocol can identify high energy-consuming nodes, i.e., nodes that are more likely to create energy holes, in a random HWSN deployment, in an effective and cost-efficient manner.

INDEX TERMS Hierarchical wireless sensor networks, random deployment, energy holes, data aggregation, routing topology.

I. INTRODUCTION

Tracking military vehicles in a battlefield, monitoring environmental phenomenon and tracking real-time events in a smart context, such as smart cities facilitated by Internet of Things (IoTs), are just a few of the many potential applications of Wireless Sensor Networks (WSNs). A WSN typically consists of a base station ($\mathcal{BS}$) and a collection of sensor nodes that are deployed to monitor some physical phenomena and to send the monitored data to the $\mathcal{BS}$ at regular intervals. In such a network, nodes are typically deployed in large numbers ranging from several hundreds to even thousands. With such a large number of nodes, the network deployment is largely random, resulting in a random routing topology. Such a topology often has non-uniform node distribution in different parts of the network, causing some nodes in the network to deplete their energy faster than the others and creating the so called energy holes in the network. In such cases, it would be beneficial to equip the WSN with the ability to identify high energy-consuming nodes in real-time, so that anticipated energy holes in the network could be identified and replacements be deployed before network operations are affected. This will ultimately prolong the lifetime of a randomly deployed WSN.

Sensor nodes are usually organized in a hierarchical tree structure where upstream nodes, that are closer to the $\mathcal{BS}$, relay data from downstream nodes, that are farther away from the $\mathcal{BS}$, and each downstream node is connected to exactly one upstream neighbour [1]. Such a network is often referred to as a hierarchical WSN (HWSN).

In a HWSN, the nodes closer to the $\mathcal{BS}$ often deplete their energy faster than other nodes in the network and energy holes are more likely to be created around the $\mathcal{BS}$. This is because nodes closer to the $\mathcal{BS}$ receive and transmit more messages, and the energy consumed by a node is largely dependent on the communication cost imposed on it (which is, in turn, dependent on the number and length of messages received and transmitted by the node).

To reduce communication costs in HWSNs, a technique called data aggregation is often used with a data collection process. With data aggregation, each non-leaf node (i.e. an aggregator $\mathcal{A}$) collects and aggregates data from its downstream neighbours and forwards the aggregated data in a single transmission to an upstream neighbour. This results in a reduced number of data transmissions by each node in the network. However, this also means that now the energy holes are no longer limited to the nodes closer to the $\mathcal{BS}$. Any node, at any hop in the network, can create an energy hole if it receives and transmits more messages than other nodes in the network.

To identify nodes creating energy holes in a HWSN that uses data aggregation, the number of messages received and transmitted by each node in the network need to be determined. In other words, to identify energy holes, the $\mathcal{BS}$ needs to know the number of downstream and upstream neighbours of each node in the network. The more downstream and upstream neighbours that a node has, the more messages the node will receive and transmit. Since, in a HWSN, each downstream node is connected to exactly one upstream neighbour, the $\mathcal{BS}$ only needs to know the number of downstream neighbours for each node to identify energy holes. One way of doing this is for each node in the network to record and transmit the IDs of its 1-hop downstream neighbours and their positions in the tree hierarchy to its 1-hop upstream neighbours. In other words, the topology data 'seen' by each node should be collected in a bottom-up manner. The major challenge, however, is how to record (or code) the topology data such that it can be carried in length-constrained messages supported by sensor nodes.

To address this challenge, this paper proposes three topology data coding methods, a Full Mapping (FM) method, Partial Mapping (PM) method and Revised Partial Mapping (RPM) method. With the FM method, each node codes the topology data of all its downstream neighbours in the form of their IDs and hop counts, and transmits the data to one of its 1-hop upstream neighbours without aggregation or any further processing. This, however, incurs high transmission costs as the messages carrying the topology data could be long. In the PM method, the concept of data aggregation is applied to topology data collection. Here each non-leaf node acts as an aggregator and codes the topology of the tree headed by itself using two sets of values. The first set lists the IDs of the nodes in the tree in an orderly manner, and the position occupied by each node's ID in the set is captured by using an index number. The second set lists the index numbers of the aggregator's 1-hop downstream neighbours. In this way, its 1-hop upstream neighbour could map the IDs present in the first set with the index numbers contained in the second set. As the second set only contains the index numbers of the aggregator's 1-hop downstream neighbours, rather than the index numbers of all the nodes in the tree headed by the aggregator, the PM method requires shorter messages to carry topology data than the FM method. However, the PM method only works for a network with a hop count of no more than 4. The RPM method is designed to overcome the limitation of the PM method by using the second set of the topology data to capture the index numbers of the leaf nodes of the tree headed by the aggregator, rather than the index numbers of its 1-hop downstream neighbours. Among the three coding methods, the RPM method is the most preferable one, as it generates short messages to carry the topology data and works for a network with any number of hops and nodes. We have carried out a theoretical analysis to evaluate the RPM method against the other two methods.

Using the RPM method, we have designed and implemented a novel protocol, called an Aggregation based Topology Learning (ATL) protocol, that facilitates real-time learning of a HWSN's routing topology. The implementation is done on a real WSN test bed. We have carried out a number of experiments on the test bed and have made some interesting discoveries from the experiments.

The remainder of this paper is organized as follows. The related work is discussed in Section II. Section III describes the three topology coding methods and carries out a theoretical analysis of the three methods. Section IV presents the design preliminaries, namely notations, assumptions and design requirements, for the ATL protocol. Section V gives an overview of the ATL protocol, followed by a detailed description of the protocol. Section VI describes the experiments carried out on a real WSN test bed and discusses the results and findings from the experiments. Finally, Section VII concludes the paper.

II. RELATED WORK

This section gives an overview of the related work under two categories. In the first category, it discusses the work relevant to the topology learning problem in WSNs and in the second category, it discusses the work relevant to the energy hole problem in WSNs.

A. ON LEARNING WSN TOPOLOGY

There is very little related work published in literature which directly addresses the topology learning problem in WSNs. In our literature research, we were able to identify only a few proposals [2]–[5] that are most relevant to our work.

Deb *et al.* [2] have proposed a TopDisc algorithm for a $\mathcal{BS}$ to learn approximate topology in a cluster-based WSN. In this proposal, each cluster is monitored by a cluster head and each node is a member of at least one cluster. To learn the WSN topology, the $\mathcal{BS}$ (i.e. the monitoring node for the entire network) initiates a topology discovery request and sends it to all the cluster heads in the network. The cluster heads, upon receipt of this request, collect and aggregate responses from the nodes in their respective clusters before sending the responses to the $\mathcal{BS}$. One of the challenging issues with this approach is how to discover a set of cluster heads in a network that could collectively provide a complete topological picture for the network. The TopDisc algorithm uses a greedy strategy to discover such a set of cluster heads. This strategy may not always produce an optimal solution in terms of the completeness of the learnt topology. In addition, the topology learning process makes use of separate control messages (i.e. topology discovery requests and responses), which will impose additional energy costs in a WSN.

Staddon *et al.* [3] have proposed a method that allows a $\mathcal{BS}$ to identify failed nodes in a WSN. In this method, routing messages are used to carry topological information from the nodes in the network to the $\mathcal{BS}$. In other words, when a routing message passes through a node, the node also adds the topological information about its neighbours into the routing message before forwarding the message further. The $\mathcal{BS}$ applies a divide and conquer strategy to identify any failed nodes in the network, based on the received information. With respect to communication costs, the $\mathcal{BS}$ requires $O(logn)$ routing messages to identify one failed node and $O(dlogn)$ routing messages to identify all the failed nodes in the network, where n is the number of probably failed nodes and d is the number of actually failed nodes.

Marinakis *et al.* [4] have proposed a Monte-Carlo Expectation Maximization (MCEM) based algorithm to determine the state of a WSN in a managed indoor environment. The WSN under consideration is assumed to be deployed at the junctions of the hallways of a large building. The algorithm uses the sequence of events generated by people passing within view of different nodes to determine a connectivity map and travel times between nodes.

Peralta *et al.* [5] have proposed the use of a Collaborative Wireless Sensor Network (CWSN) model to represent the state of a network and to capture its properties before the network is actually deployed. CWSN is a mathematical approach for the modelling and analysis of a WSN. The graphical representation of the model allows the visualization of the communication interfaces between nodes, state of the nodes and state of the links. This allows for a better organization and management of the network when it is actually deployed. The CWSN model may only be applicable to non-random deployments of WSNs and does not support real-time learning of a routing topology.

B. ON IDENTIFYING ENERGY HOLES

The analysis of energy consumption by each node and the network lifetime of a WSN is a widely studied problem in literature. Several proposals have been published to address this problem [6]–[25]. We here discuss the most notable proposals.

Chen *et al.* [6] have used a mathematical model to estimate the communication load on each node in a multi-hop WSN. The findings from this work confirm that a node closer to the $\mathcal{BS}$ consumes more energy than a node farther away from the $\mathcal{BS}$ due to the increased communication load on the nodes closer to the $\mathcal{BS}$. The authors suggest that the mathematical model can be used to predict the operational lifetime of a network and to identify nodes with heavy communication load in the network.

Cheng *et al.* [7] have proposed two models: a network lifetime model and a deployment cost model for network lifetime analysis. The two models are used to establish the estimated network lifetime against different network deployment strategies. The authors suggest that, using the two models, suitable network deployment strategies can be formulated to help with the actual deployment of a WSN.

Zhao *et al.* [8] have described an idea of a residual energy scan (eScan) that allows a $\mathcal{BS}$ to acquire information about the resource distribution in terms of remaining energy level in the network. Each node regularly carries out local scans to get an update information about its remaining energy level. When a request is received from the $\mathcal{BS}$, the nodes aggregate and forward their local scans towards the $\mathcal{BS}$. The scans received by the $\mathcal{BS}$ provide a summary information of the geographical distribution of energy resources in the network. However, the eScans do not provide detailed information about the energy level at any particular node.

Zhao *et al.* [9] have proposed to use aggregates of network properties, called the network digests, to indicate node failures and resource depletions in a network. Each node regularly computes digests of network properties like packet loss rates, remaining energy levels, etc. and uses these digests to alert the $\mathcal{BS}$ about any erroneous conditions within the network. To improve the accuracy of the computed digests, the authors suggest to exclude data links with heavy packet loss and asymmetry from the digest computations.

L. Zhang *et al.* [10] have proposed a generic distributed progressive algorithm (DPA) to maximize network lifetime for a WSN. They define network lifetime as the time period from the deployment of the network to the time when $\mathcal{BS}$ can no longer receive data from any node in the network. In DPA, a lifetime vector is used to define the lifetimes of all active nodes in an ascending order. DPA runs iteratively in a distributed manner to produce the values in the lifetime vector. During each iteration, each node makes localised estimations for its lifetime based on its remaining energy level and communicates the estimated lifetime to its neighbours using flooding. With each iteration, the $\mathcal{BS}$ gets a better lifetime vector than the previous one and this continues till DPA stabilises, at which point the $\mathcal{BS}$ gets the lifetime vector with the maximum values.

Kacimi *et al.* [11] have discussed load-balancing techniques to maximize the network lifetime. To do so, a distributed heuristic solution is proposed that adjusts the transmission power of sensor nodes in order to balance their energy consumption.

Rout *et al.* [12] designate the area around the $\mathcal{BS}$ as a bottleneck zone and propose the use of duty cycling and network coding to enhance the lifetime of the nodes in this area. Each node in the bottleneck zone combines all the data it receives and sends the combined data in a single transmission.

This reduces the number of transmissions to the $\mathcal{BS}$ and improves the lifetime of the nodes in the bottleneck zone.

Liu *et al.* [13] have proposed three algorithms to reduce hotspots or energy holes in a network and increase the network lifetime. To optimise network lifetime using the first algorithm (GlobalSame), they propose to adjust the values of network parameters (e.g. transmission radius, transmission rate, etc.) globally. In the second algorithm (RingSame), the authors propose that the network lifetime can be further increased by adjusting the values of the network parameters based on the network conditions in different regions. Using the third algorithm (NodeDiff), they propose that, to further increase the network lifetime, the values of the network parameters should be adjusted for each individual node in the network based on its distance from the $\mathcal{BS}$.

Different from the above mentioned proposals, this paper describes a real-time routing topology learning approach by which high energy-consuming nodes, and the likely energy holes, in a HWSN can be identified by using the same procedure and same set of communication messages that are widely used for data collections in HWSNs.

III. TOPOLOGY DATA CODING METHODS

The task of learning the routing topology of a HWSN involves finding answers to the following questions: (i) how to express a node's topology data, and (ii) how to get such topology data to the $\mathcal{BS}$. With respect to question (ii), we can consider topology data collection in a HWSN as a normal data (i.e. sensed data) collection process. Since a sensed data collection process often makes use of data aggregation to reduce communication costs, therefore topology data collection can also make use of data aggregation for the same reason. Applying data aggregation to topology data collection, each node (starting from the first non-leaf nodes) collects topology data from its 1-hop downstream neighbours, aggregates the collected topology data with its own topology data and transmits the aggregated topology data to a 1-hop upstream neighbour. Each 1-hop upstream neighbour then learns the downstream routing topology using the topology data received from its 1-hop downstream neighbours. In this way, each upstream node, and finally the $\mathcal{BS}$, can learn the downstream routing topology as part of the data collection process.

The questions that remain are how to express a node's topology data and how such topology data may be aggregated. To address these questions, we have designed three methods, i.e. the Full Mapping (FM) method, the Partial Mapping (PM) method and the Revised PM (RPM) method.

A. FULL MAPPING (FM) METHOD

The basic idea behind the FM method is that if the $\mathcal{BS}$ knows the identity and hop count of each node in the network, then the BS can work out the tree topology of the network. In other words, in a bottom-up fashion starting from the first non-leaf node layer, if a node (referred as a tree-head) can pass the identity of each of its 1-hop downstream neighbours along with their positions in the tree hierarchy to a 1-hop upstream neighbour, then the 1-hop upstream neighbour can learn the topology of the tree headed by this tree-head.

To explain this method and for the sake of clarity, we first give two definitions.

Definition 1 (Topology Identity (TID)): TID is used to express a node's topology data. It comprises of two attributes, i.e. Node_ID and Hop_Count, where Node_ID is an integer that uniquely identifies a node in the network and Hop_Count is an integer that refers to the node's hierarchical position in the network in terms of its hop count from the $\mathcal{BS}$. TID for a node X (expressed as TID_X) is given as $<\text{Node\_ID}_X, \text{Hop\_Count}_X>$.

Definition 2 (Neighbourhood Topology Data With FM (NT_Data_FM)): NT_Data_FM is used to express aggregated topology data at a node. NT_Data_FM for a leaf node is the same as its TID, as it does not have any downstream neighbours. For a tree-head X with m downstream neighbours, NT_Data_FM_X is given as:

$$\begin{aligned} NT\_Data\_FM_X = &< Node\_ID_X, Hop\_Count_X >, \\ &< Node\_ID_1, Hop\_Count_1 >, \ldots, \\ &< Node\_ID_m, Hop\_Count_m > \end{aligned} \tag{1}$$

With the use of above definitions, we now describe the FM method. Each node in the network, starting from the first non-leaf node layer, collects the TIDs of its 1-hop downstream neighbours, appends its own TID to the collected TIDs to construct the so-called NT_Data_FM of this node, and sends it to a 1-hop upstream neighbour. Similarly, the 1-hop upstream neighbour performs the same operations. Once all such data are delivered to the $\mathcal{BS}$, the $\mathcal{BS}$ can learn the topology of the entire tree.

Now let us use the network structure shown in Fig. 1 to further illustrate how the $\mathcal{BS}$ learns the routing topology using the FM method. Based on Definition_1, the TID of node J is: <J, 5>. According to Definition_2 and assuming that the TID_J has been collected by G, G generates $\text{NT\_Data\_FM}_G = \text{TID}_G, \text{TID}_J$ = <G, 4>, <J, 5>. Next, G sends this data to D, which is G's 1-hop upstream neighbour. Similarly, D appends its own TID, <D, 3>, to the received NT_Data_FM_G. As D does not have any other downstream neighbours, so NT_Data_FM_D = <D, 3>, <G, 4>, <J, 5>. D then sends this data to its 1-hop upstream neighbour, i.e. B. B carries out similar steps and generates NT_Data_FM_B = <B, 2>, <D, 3>, <G, 4>, <J, 5>. Next, B sends this data to its 1-hop upstream neighbour, i.e. A. By using a similar process, A also collects the topology data of the right branch of the tree from node C, i.e. NT_Data_FM_C = <C, 2>, <E, 3>, <F, 3>, <H, 4>, <I, 4>. Using the topology data from both B and C, A generates NT_Data_FM_A = <A, 1>, <B, 2>, <D, 3>, <G, 4>, <J, 5>, <C, 2>, <E, 3>, <F, 3>, <H, 4>, <I, 4> and sends it to the $\mathcal{BS}$. Using the topology data from A, the $\mathcal{BS}$ can learn the topology of the tree.

The main limitation of the FM method is that length of NT_data_FM generated at each hop increases proportional

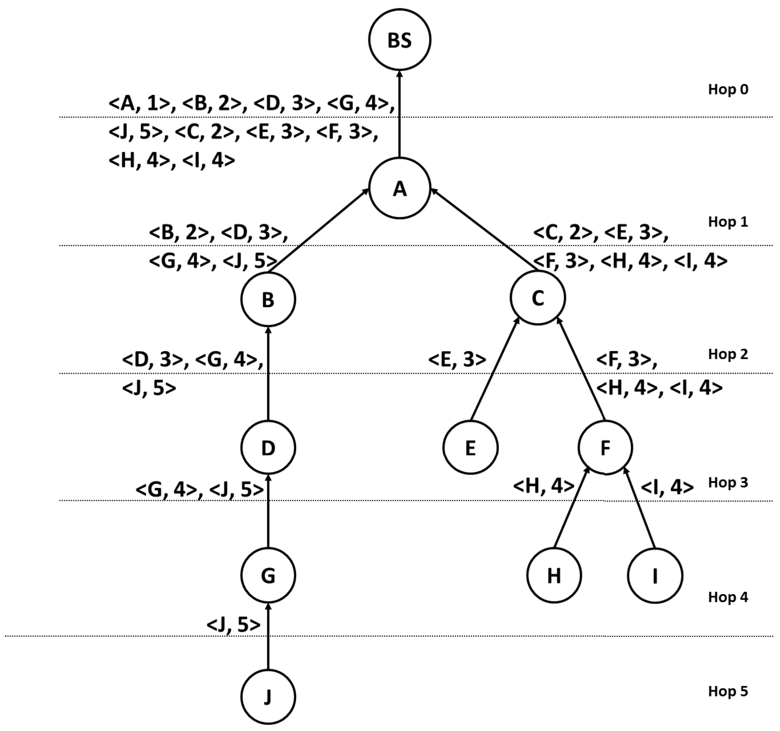

FIGURE 1. Full mapping method explained.

to the number of downstream neighbours. This is because, with FM, there is actually no aggregation of topology data during the data collection process. At each hop, new topology data are simply appended (i.e. concatenated) to the received NT_Data_FM. This will lead to two issues.

The first issue is the level of communication overhead introduced by the FM method. A node with many downstream neighbours requires a longer message to carry its NT_Data_FM. In addition, the length of this message increases as it propagates hop-by-hop towards the $\mathcal{BS}$. The longer the message is, the higher the transmission cost and thus the higher the communication overhead. This communication overhead is particularly high if long messages are generated at the leaf end of the tree, the network is very large and/or the hierarchy of the tree has many levels.

The second issue is the limited network size that could be supported by the FM method. Based on the current sensor technology, sensor nodes typically support a message length of around a hundred bytes. For example, a standard TinyOS message is just 36 bytes long with 5 bytes of header, 29 bytes of data payload and 2 bytes of CRC [26]. The CC2420 radio used by the sensor nodes allows a maximum packet length of 128 bytes, so the data payload length in a TinyOS message can be increased up to 121 bytes [27]. Let's assume a data payload length of 40 bytes. Using 1-byte to code each Node_ID and 1-byte to code each Hop_Count, a node can code the topology data for a maximum of 20 downstream neighbours over a maximum of 20 hops (20 nodes can be spread over a maximum of 20 hops) where 20 bytes are used to code the Node_IDs for 20 downstream neighbours and 20 bytes are used to code the Hop_Counts.

To reduce the length of the messages carrying the collected topology data, we need to devise a different way of expressing a node's topology data that allows the aggregation of topology data as it propagates towards the $\mathcal{BS}$. In the following, we describe such a method, which is called the Partial Mapping (PM) method.

B. PARTIAL MAPPING (PM) METHOD

As mentioned above, the PM method is aimed at expressing the topology of a tree with a shorter message than the FM method. The idea used is to express the topology data at a tree-head using two sets: the first set lists the identities of the nodes in the tree in an orderly fashion, starting with the tree-head Node_ID and followed by the Node_IDs passed by its 1-hop downstream neighbours in a left-to-right fashion. Each Node_ID in the first set is given an index number that starts from 0 and increments by 1 for each subsequent Node_ID in the set. The second set of the topology data lists the index numbers for the 1-hop downstream neighbours of the tree-head. When a 1-hop upstream neighbour of the tree-head receives this topology data, it can infer the topology of the tree headed by this tree-head by mapping the Node_IDs present in the first set with the index numbers contained in the second set. That is to say, it can work out which node is the tree head, which node is the first child of the tree head (indicated by the first index number in the second set), which node is the second child of the tree head (indicated by the second index number in the second set), and so on. Different from the FM method, that encodes both the Node_IDs and the positions (hop counts) of all the nodes in a tree, the PM method encodes the Node_IDs of all the nodes in a tree but only the positions of the 1-hop downstream neighbours of the tree-head. In this way, the PM method cuts out the position information of the nodes that are not 1-hop downstream neighbours of the tree-head from the topology data, thus reducing the size of a message carrying the topology data.

To further explain this method, we first define four terms used in the PM method.

Definition 3 (Family): A family is formed by a node and its 1-hop downstream neighbours. The node is called the parent and its 1-hop downstream neighbours are the children of the parent.

Definition 4 (Extended Family): An extended family is formed by a node, called the tree-head, and all the downstream nodes (i.e. children, grandchildren, greatgrandchildren and so on) in the tree headed by this node.

Definition 5 (Extended Family Node_IDs (EF-ID)): This is a set of Node_IDs for the nodes in an extended family. The Node_IDs are listed in the set based on the 'family positions' of the nodes, such that a tree-head's 1-hop downstream neighbour is followed by its children, grandchildren and so on.

Definition 6 (Nodes' Positions in the Extended Family (NP-EF)): This is a set of integers, called index_numbers, assigned just to the 1-hop downstream neighbours (i.e. the children) of a tree-head to indicate their respective positions in EF-ID.

Definition 7 (Neighbourhood Topology Data With PM (NT_Data_PM)): NT_Data_PM uses a node's EF-ID and NP-EF to express aggregated topology data at a node. NT_Data_PM for a leaf node only contain its own Node_ID in EF-ID and a null value in NP-EF. For a tree-head X with m 1-hop downstream neighbours, NT_Data_PM$_X$ is given as:

$$\begin{aligned} &NT\_Data\_PM_X \\ &\quad = EF-ID_X, NP-EF_X \\ &\quad = \{Node\_ID_X, Node\_IDs\ of\ Child1 \\ &\qquad and\ its\ children, \ldots, Node\_IDs\ of\ Childm \\ &\qquad and\ its\ children\}, \{index\_number\ of\ Child1, \\ &\qquad \ldots, index\_number\ of\ Childm\} \end{aligned} \tag{2}$$

With the use of above definitions, we now describe the PM method. Each node in the network, starting from the first non-leaf node layer, collects the Node_IDs of all its downstream neighbours, appends its own Node_ID to the collected Node_IDs and constructs the so-called EF-ID of this node. It then determines the positions occupied by its 1-hop downstream neighbours in EF-ID and puts these positions in NP-EF. It then constructs NT_Data_PM using EF-ID and NP-EF and sends it to a 1-hop upstream neighbour. Once the 1-hop upstream neighbour gets this data, it uses NP-EF to determine the family positions in EF-ID. The 1-hop downstream neighbours of the sending node are determined directly from the index_numbers present in NP-EF, while the other downstream neighbours are conveyed using the agreed order of the Node_IDs' appearances in the data structure 2.

Using Fig. 2, we now illustrate how the $\mathcal{BS}$ learns routing topology using the PM method. At the lowest level (i.e. hop 5), there is only one leaf node J, so its EF-ID only contain its own Node_ID and its NP-EF is null, i.e. NT_Data_PM$_J$ = EF-ID$_J$, NP-EF$_J$ = {J, { }}, { }.

At hop 4, there are three nodes; G is a tree-head and H and I are leaf nodes. G has one 1-hop downstream neighbour J, so NT_Data_PM$_G$ ={G, {J}}, {1} where 1 indicates the family position of its 1-hop downstream neighbour J. Since H and I are leaf nodes so NT_Data_PM$_H$ ={H, { }}, { } and NT_Data_PM$_I$ ={I, { }}, { }.

At hop 3, there are three nodes; D and F are tree-heads and E is a leaf node. D has one 1-hop downstream neighbour G, so NT_Data_PM$_D$ ={D, {G, J}}, {1} where 1 indicates the family position of its 1-hop downstream neighbour G. F has two 1-hop downstream neighbours H and I, so NT_Data_PM$_F$ ={F, {H, I}}, {1, 2} where 1 and 2 indicate the family positions of its 1-hop downstream neighbours H and I. Since E is a leaf node so NT_Data_PM$_E$ ={E, { }}, { }.

At hop 2, there are two nodes; B and C and both are tree-heads. B has one 1-hop downstream neighbour D, so NT_Data_PM$_B$ ={B, {D, G, J}}, {1} where 1 indicates the family position of its 1-hop downstream neighbour D. C has two 1-hop downstream neighbours E and F, so NT_Data_PM$_C$ = {C, {E, F, H, I}, {1, 2} where 1 and 2 indicate the family positions of its two 1-hop downstream neighbours E and F.

At hop 1, there is only one node, A, which is a tree-head. Node A has two 1-hop downstream neighbours B and C, so NT_Data_PM$_A$ ={A, {B, D, G, J, C, E, F, H, I}}, {1, 5}, where 1 and 5 indicate the family positions of its two 1-hop downstream neighbours B and C.

When the $\mathcal{BS}$ receives NT_Data_PM$_A$, it gets to know that A is its 1-hop downstream neighbour. It then uses the index_numbers {1, 5} and gets to know that B and C are its 2-hop downstream neighbours and the remaining nodes D, G, J, E, F, H and I are its 3-hop downstream neighbours.

It can be seen that the PM method actually provides a certain degree of topology data aggregation: the NT_Data_PM is restructured at each hop and only the index_numbers for the 1-hop downstream neighbours are recorded in NT_Data_PM. As a result, fewer bytes are required to carry NT_Data_PM than NT_Data_FM. For example, let's compare NT_Data_PM$_A$ with NT_Data_FM$_A$. Assuming that each Node_ID is coded using a 1-byte code (allowing the maximum network size of 256 nodes), and each index_number is coded using a 1-byte code, then NT_Data_PM$_A$ generates 96 bits (12 bytes) of topology data, as compared to a 160 bits (20 bytes) of topology data generated by NT_Data_FM$_A$.

The PM method is more efficient, in terms of cutting down the length of a message carrying the topology data,

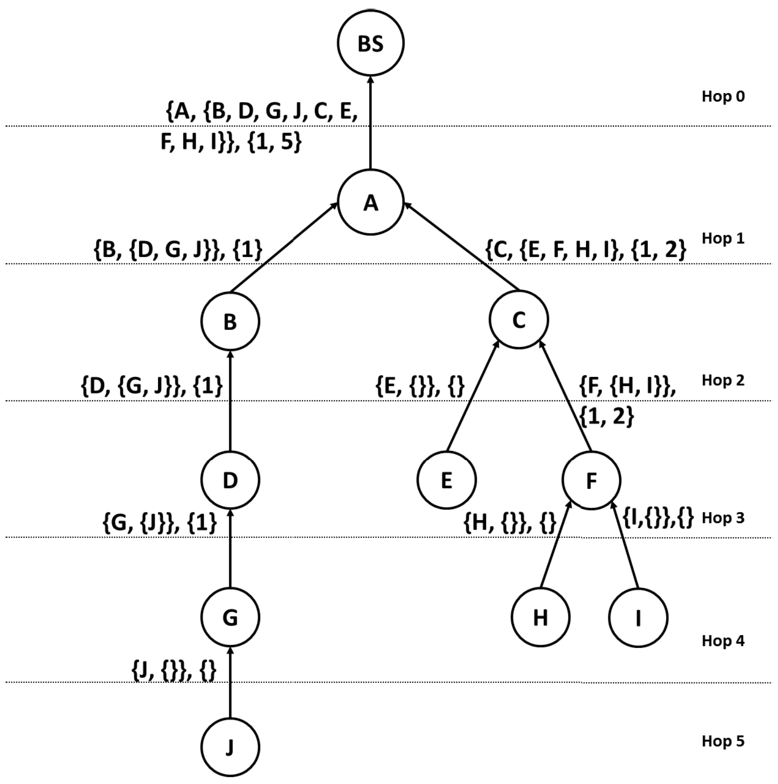

FIGURE 2. **Partial mapping method explained.**

as compared to the FM method. However, the PM method works only when the network is limited to 4 hops (hop 0 - 3). For example, in Fig. 2, the $\mathcal{BS}$ learns that G, H, I and J are its 3-hop downstream neighbours, which is actually not the case. With the PM method, a node can only distinguish its 1-hop and 2-hop downstream neighbours from its downstream neighbours in other hops and all the downstream neighbours in other hops get categorised as a node's 3-hop downstream neighbours. A typical WSN deployment contains a large number of sensor nodes and it is very likely that the network has a hierarchical structure with 5 or more hops. This means that the PM method cannot be applied to learn the routing topology in a real WSN environment.

C. REVISED PARTIAL MAPPING (RPM) METHOD

To address the limitation of the PM method, we propose two modifications to the PM method. The first modification is to divide the tree headed by a tree-head into a set of branches, where a branch refers to a unique path formed by the tree-head's downstream neighbours to connect exactly one leaf node to the tree-head. The second modification is to code the index numbers of the leaf nodes in the second set of the topology data, instead of the index numbers of the 1-hop downstream neighbours of the tree-head. We now apply these modifications to the PM method and explain the Revised Partial Mapping (RPM) method.

The idea used in the RPM method is to express the topology data for the different branches in the tree headed by a tree-head using two sets. The first set lists tree-head's own Node_ID and the Node_IDs in each branch in an orderly fashion, and the second set lists the index numbers of the leaf nodes in these branches.

To express a node's topology data in the RPM method, we replace EF-ID and NP-EF in the topology data with two new fields. The two fields and the corresponding data structure for RPM are defined as follows:

Definition 8 (Branch-Based Extended Family Node_IDs (BEF-ID)): This is a set of Node_IDs representing the nodes on each branch headed by a tree-head. For each branch, the Node_IDs are listed based on their position in the branch, such that the node at the top of the branch occupies the first available position, followed by the next node in the branch and so on till the leaf node at the bottom of the branch.

Definition 9 (Leaf Nodes' Positions in the Extended Family (LNP-EF)): This is a set of integers, called index_numbers, assigned just to the leaf nodes (i.e. the bottom most node) on each branch to indicate their respective positions in BEF-ID.

Definition 10 (Neighbourhood Topology Data With RPM (NT_Data_RPM)): NT_Data_RPM uses a node's BEF-ID and LNP-EF to express aggregated topology data at a node. NT_Data_RPM for a leaf node only contain its own Node_ID in BEF-ID and a 0 value in LNP-EF. For a tree-head X with b branches and n_j nodes (where $1 \leq j \leq b$) along each branch,

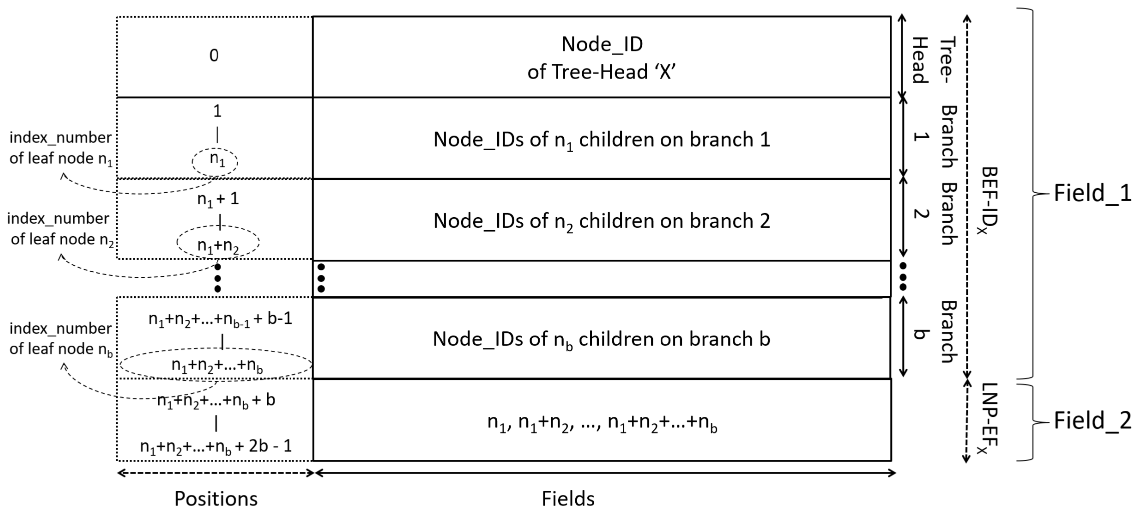
0
Node_ID of Tree-Head 'X'
index_number of leaf node n1
1
n1
Node_IDs of n1 children on branch 1
index_number of leaf node n2
n1 + 1
n1+n2
Node_IDs of n2 children on branch 2
index_number of leaf node nb
n1+n2+...+nb-1 + b-1
n1+n2+...+nb
Node_IDs of nb children on branch b
n1+n2+...+nb + b
n1+n2+...+nb + 2b - 1
n1, n1+n2, ..., n1+n2+...+nb
Tree-Head
Branch 1
Branch 2
Branch b
BEF-IDX
LNP-EFX
Field_1
Field_2
Positions
Fields

FIGURE 3. Data structure used in RPM.

NT_Data_RPM$_X$ is given as:

$$\begin{aligned} &NT\_Data\_RPM_X \\ &\quad = BEF-ID_X, LNP-EF_X \\ &\quad = \{Node\_ID_X, Node\_IDs\ of\ n_1 \\ &\qquad children\ on\ branch\ 1, \ldots, Node\_IDs \\ &\qquad of\ n_b\ children\ on\ branch\ b\}, \{index\_number \\ &\qquad of\ leaf\ node\ n_1, \ldots, index\_number \\ &\qquad of\ leaf\ node\ n_b\} \end{aligned} \tag{3}$$

The data structure for NT_Data_RPM is shown in Fig. 3.

With the use of above definitions, we now describe the RPM method. Each node in the network, starting from the first non-leaf node layer, collects the Node_IDs for all its branches, appends its own Node_ID to the collected Node_IDs and constructs its BEF-ID. It then determines the positions occupied by leaf nodes in BEF-ID for each branch and put these positions in LNP-EF. It then constructs NT_Data_RPM using BEF-ID and LNP-EF, and sends it to a 1-hop upstream neighbour. When a 1-hop upstream neighbour receives this data, it first separates the branches in BEF-ID by using the index_numbers in LNP-EF. Once the branches have been separated, the node determines the hop position for each Node_ID using the agreed order of Node_IDs appearance in each branch, as specified in the data structure 3.

Using Fig. 4, we now illustrate how the $\mathcal{BS}$ learns routing topology using the RPM method. On the left side of the tree headed by node A, there is only one branch connecting leaf node J to A. At hop 5, J produces NT_Data_RPM that contains just its own Node_ID in BEF-ID and a 0 value in LNP-EF, as it is a leaf node. At hop 4, G heads one branch with leaf node J and produces NT_Data_RPM$_G$ = {G, {J}}, {1}, where 1 is the index_number for leaf node J in BEF-ID$_G$. Similarly, at hop 3 and hop 2, nodes D and B generate NT_Data_RPM$_D$ = {D, {G, J}}, {2} and NT_Data_RPM$_B$ = {B, {D, G, J}}, {3}, where 2 and 3 indicate the index_number for their only leaf node J in BEF-ID$_D$ and BEF-ID$_B$ respectively.

On the right side of the tree headed by node A, there are three branches connecting leaf nodes E, H and I to A. At hop 4, there are two leaf nodes, H and I, so NT_Data_RPM$_H$ = {H, { }}, {0} and NT_Data_RPM$_I$ = {I, { }}, {0}. Similarly, at hop 3, E being a leaf node generates NT_Data_RPM$_E$ = {E, { }}, {0}. The other node F at hop 3 heads two branches with leaf nodes H and I, so NT_Data_RPM$_F$ = {F, {H, I}}, {1, 2} where 1 and 2 are the index_numbers for the leaf nodes H and I in BEF-ID$_F$. At hop 2, C heads three branches with leaf nodes E, H and I and its topology data is described as NT_Data_RPM$_C$ = {C, {E}, {F, H}, {F, I}}, {1, 3, 5}, where 1, 3 and 5 indicate the index_numbers of the three leaf nodes in BEF-ID$_C$.

At hop 1, A heads four branches with leaf nodes J, E, H and I, so NT_Data_RPM$_A$ = {A, {B, D, G, J}, {C, E}, {C, F, H}, {C, F, I}}, {4, 6, 9, 12}, where 4, 6, 9 and 12 indicate the index_numbers of the four leaf nodes in BEF-ID$_A$.

When the $\mathcal{BS}$ receives NT_Data_RPM$_A$, it gets to know that A is its 1-hop downstream neighbour and using the index_numbers present in LNP-EF$_A$, it determines that there are four branches (say b1, b2, b3, b4) in BEF-ID$_A$. Using the index_numbers {4, 6, 9, 12} in LNP-EF$_A$, it separates the four branches as b1: {B, D, G, J}, b2: {C, E}, b3: {C, F, H} and b4: {C, F, I}. Next the $\mathcal{BS}$ identifies the hop positions for the nodes in the four branches. Since A is a 1-hop downstream neighbour, so for b1, B would be a 2-hop downstream neighbour, D would be a 3-hop downstream neighbour, G would be a 4-hop downstream neighbour and J would be a 5-hop downstream neighbour to $\mathcal{BS}$. Using the same logic, C is identified as a 2-hop downstream neighbour, E and F are identified as 3-hop downstream neighbours and H and I are identified as 4-hop downstream neighbours to $\mathcal{BS}$.

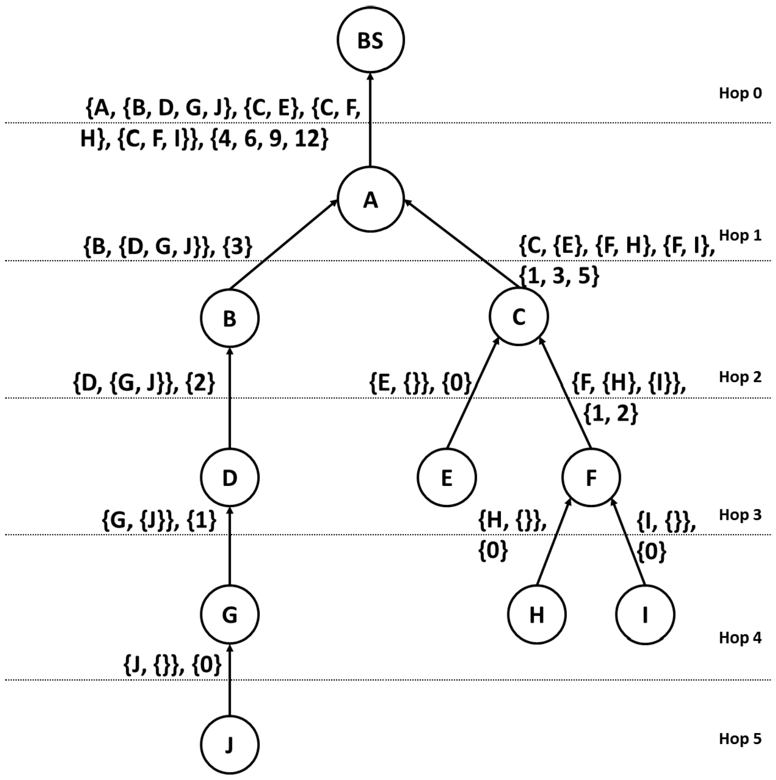

FIGURE 4. Revised partial mapping method explained.

In comparison with the PM method, the RPM method can express the routing topology of a HWSN over a higher number of hops. It also uses fewer bytes to code the topology data than the FM method, which means that the topology data of more nodes could be coded into NT_Data_RPM. For example, let's compare NT_Data_RPM$_A$ with NT_Data_PM$_A$ and NT_Data_FM$_A$. NT_Data_RPM$_A$ correctly describes that the tree headed by A extends over 5 hops, whereas NT_Data_PM$_A$ described that the tree headed by A extends over just 3 hops. Assuming that each Node_ID is coded using a 1-byte code (allowing the maximum network size of 256 nodes), and each index_number is coded using a 1-byte code, then NT_Data_RPM$_A$ generates 136 bits (17 bytes) of topology data, as compared to a 160 bits (20 bytes) topology data generated by NT_Data_FM$_A$.

The RPM method gives a better performance than both the FM and PM methods, however, the RPM method comes with two costs. The first one is the computational cost introduced by the aggregation and message restructuring carried out at each node, which is the price for the benefits offered by the RPM method. The second cost is the communication cost associated with the message carrying NT_Data_RPM. Since the Node_ID of each node along each branch is added in the message, so an upstream node with multiple leaf nodes will be added multiple times in the message. For example, in Fig. 4, C's Node_ID is added three times in BEF-ID$_A$ at hop 1, for the three branches reported by C. The addition of an upstream node multiple times creates redundancy in the message carrying NT_Data_RPM. The communication cost of the RPM method would be low if there is a low redundancy (i.e. fewer number of repeated Node_IDs) in the message and high otherwise. As the redundancy in the message, in turn, depends on how nodes connect to each other to form the branches, so the performance of the RPM method actually depends on the routing topology.

D. THEORETICAL ANALYSIS

This section compares the three topology coding methods. With each of the methods and for a given data payload length, we analyse how much of the routing topology (i.e. the number of nodes and the number of hops in the network) can be learnt by the $\mathcal{BS}$.

As mentioned in Section III-C, the performance of the RPM method depends on the routing topology. However, this is not the case for the other two methods, i.e. their performance is independent of the routing topology. For a data payload length of 'd' bytes, 'm' 1-hop downstream neighbours and 'b' branches:

- A maximum of h_FM hops and a maximum of n_FM nodes can be coded with the FM method, where:

$$\text{h\_FM} = \frac{d}{2} \tag{4}$$

$$\text{n\_FM} = \frac{d}{2} \tag{5}$$

As discussed in Section III-A, the FM method requires the Node_ID and Hop_Count of each node to code the topology data. This means that maximum nodes and maximum hops that can be coded using the FM method each would be one half of d and this is the case, regardless of any values of m or b.

- A maximum of h_PM hops and a maximum of n_PM nodes can be coded with the PM method, where:

$$\text{h\_PM} = 3 \tag{6}$$

$$\text{n\_PM} = d - m \tag{7}$$

As discussed in Section III-B, the PM method only works when the network is limited till hop 3 and this is true, regardless of any values of d, m or b. Since m bytes are required to code the index_numbers of m 1-hop downstream neighbours, so the remaining bytes can be used to code a maximum of $d - m$ nodes.

To evaluate and compare the performance of the RPM method against the other two methods, we consider two routing topologies:

1) Best Routing Topology: Nodes are connected such that NT_Data_RPM contains zero redundancy.
2) Worst Routing Topology: Nodes are connected such that NT_Data_RPM contains the maximum possible redundancy.

For both routing topologies, we compare the maximum number of hops that can be coded using the RPM method against the PM method, and the maximum number of nodes that can be coded using the RPM method against the FM method. This comparison is reasonable as the RPM method aims to code more hops than the PM method and more nodes than the FM method in the topology data.

1) BEST ROUTING TOPOLOGY

Fig. 5 shows the best routing topology for a HWSN that would generate zero redundancy with the RPM method. The network consists of $i + b$ nodes, where $C_1, C_2, ..., C_i$ are the non-leaf nodes and $L_1, L_2, ..., L_b$ are the leaf nodes along b branches. All the non-leaf nodes are along one branch that connects leaf node L_1 at hop $i + 1$ to the $\mathcal{BS}$. All the other branches connect leaf nodes $L_2, ..., L_b$ at hop 1 directly to the $\mathcal{BS}$. To code Node_IDs and index_numbers in this routing topology, data payload length of ‘d’ bytes should be at least:

$$\text{d} = i + 2b \tag{8}$$

where i bytes are required to code Node_IDs for i non-leaf nodes and $2b$ bytes are required to code the Node_IDs and index_numbers of b leaf nodes.

Under the best routing topology and for a data payload length of ‘d’ bytes, ‘m’ 1-hop downstream neighbours and ‘b’ branches:

- A maximum of $\text{h}_0\text{\_RPM}$ hops can be coded with the RPM method, where:

$$\text{h}_0\text{\_RPM} = d - 2b + 1 = i + 1 \tag{9}$$

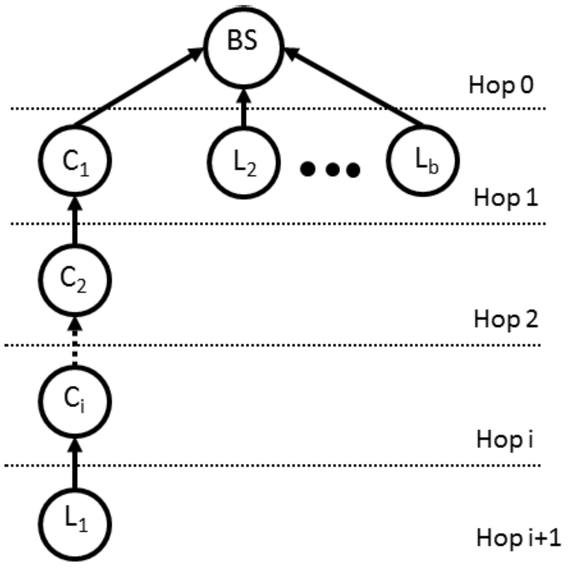

FIGURE 5. Best routing topology for zero redundancy.

As ‘2b’ bytes are reserved to code the Node_IDs and index_numbers for leaf nodes along ‘b’ branches, so the remaining bytes can represent non-leaf nodes along ‘d - 2b’ hops. 1 is added to ‘d - 2b’ hops to include the hop for the leaf node along the longest branch.

- As no redundant nodes are present in NT_Data_RPM, therefore the number of redundant nodes $\text{r}_0\text{\_RPM}$ in NT_Data_RPM is given as:

$$\text{r}_0\text{\_RPM} = 0 \tag{10}$$

- With no redundant nodes in NT_Data_RPM, a maximum of $\text{n}_0\text{\_RPM}$ nodes can be coded with the RPM method, where:

$$\text{n}_0\text{\_RPM} = d - b = i + b \tag{11}$$

As ‘b’ bytes are used to code the index_numbers for leaf nodes along ‘b’ branches, so the remaining ‘d - b’ bytes can be used to code the Node_IDs.

- With no redundant nodes in NT_Data_RPM, redundancy $\text{R}_0\text{\_RPM}$ in NT_Data_RPM is calculated as:

$$\text{R}_0\text{\_RPM} = \frac{\text{r}_0\text{\_RPM}}{\text{n}_0\text{\_RPM}} * 100 = 0 \tag{12}$$

Comparing equations 6 and 9, RPM performs better than the PM method when:

$$\begin{aligned} \text{h}_0\text{\_RPM} &> \text{h\_PM} \\ d - 2b + 1 &> 3 \\ b &> \frac{d-2}{2} \end{aligned} \tag{13}$$

When b is equal to $\frac{d-2}{2}$, both RPM and PM can code the same number of maximum hops (i.e. 3) in the topology data. When b is greater than $\frac{d-2}{2}$, the RPM method can only code the topology data if all the b branches terminate at hop 1.

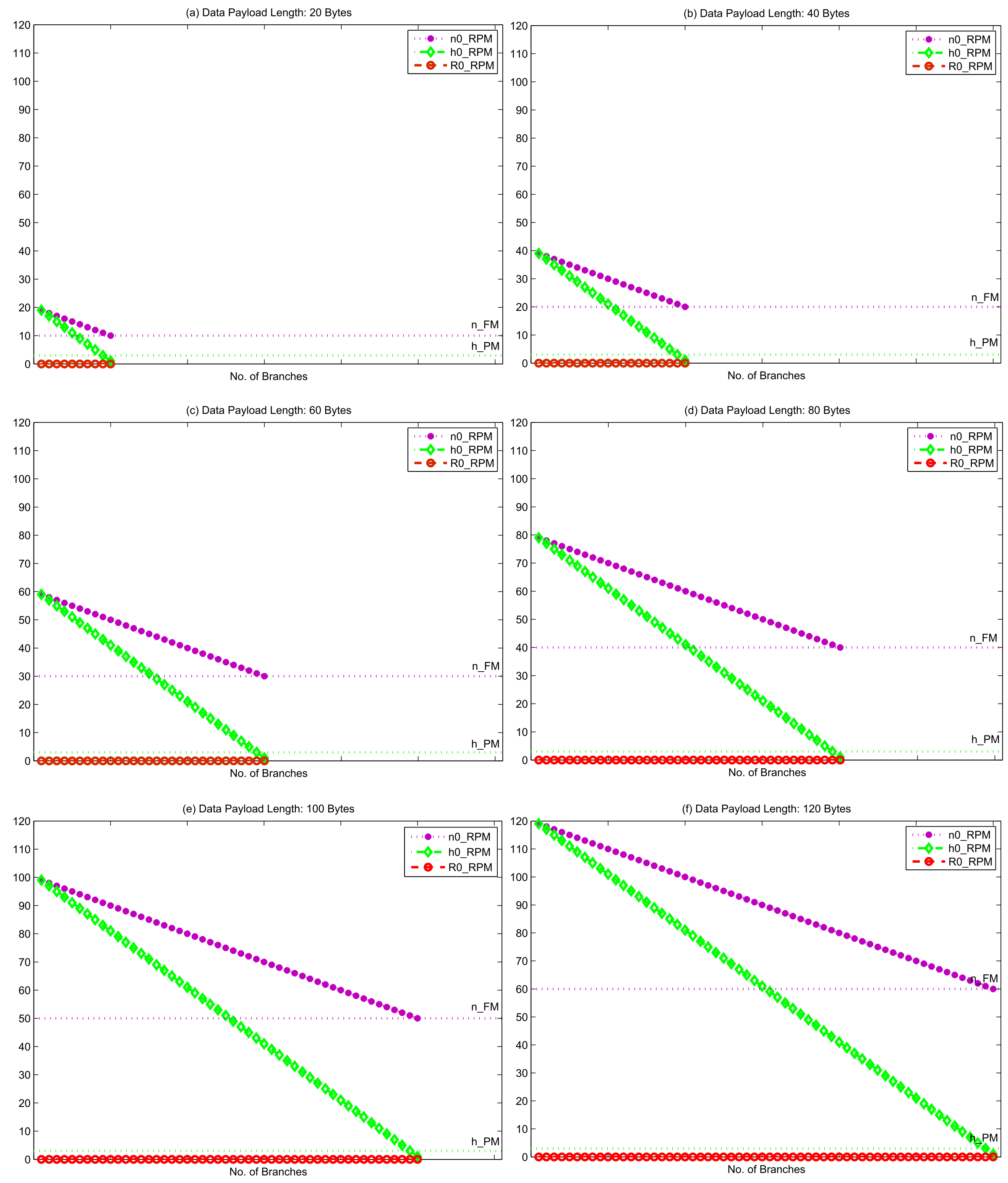

FIGURE 6. Comparing RPM against FM and PM under best routing topology.

Comparing equations 5 and 11, RPM performs better than the FM method when:

$$\begin{aligned} n_0\_RPM &> n\_FM \\ d - b &> \frac{d}{2} \\ b &> \frac{d}{2} \end{aligned} \tag{14}$$

When b is equal to $\frac{d}{2}$, both RPM and FM can code the same number of maximum nodes in the topology data. When b is greater than $\frac{d}{2}$, it is no longer possible to code the topology data using the RPM method with 'd' bytes.

Fig. 6 compares h_0_RPM against h_PM and n_0_RPM against n_FM at zero redundancy (i.e. R_0_RPM) for data payload lengths of 20, 40, 60, 80, 100 and 120 bytes respectively. Based on the results, on average, RPM gives a 92%

better performance than the PM method and a 96% better performance than the FM method under the best routing topology.

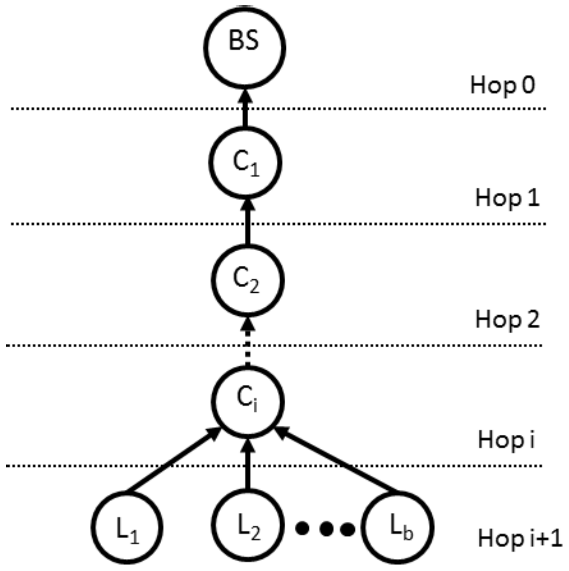

FIGURE 7. Worst routing topology for maximum possible redundancy.

2) WORST ROUTING TOPOLOGY

Fig. 7 shows the worst routing topology for a HWSN that would generate maximum possible redundancy with the RPM method. The network consists of $i + b$ nodes, where C_1, C_2, ..., C_i are the non-leaf nodes and L_1, L_2, ..., L_b are the leaf nodes along b branches. All the non-leaf nodes are along one branch till hop i. This branch connects leaf nodes $L_1, L_2, ..., L_b$ present at hop $i+1$ to the $\mathcal{BS}$. To code Node_IDs and index_numbers in this routing topology, data payload length of 'd' bytes should be at least:

$$d = ib + 2b \tag{15}$$

where ib bytes are required to code Node_IDs for i non-leaf nodes b times (one time for each branch) and $2b$ bytes are required to code the Node_IDs and index_numbers of b leaf nodes.

Under the worst routing topology and for a data payload length of 'd' bytes, 'm' 1-hop downstream neighbours and 'b' branches:

- A maximum of h_{max}_RPM hops can be coded with the RPM method, where:

$$\begin{aligned} h_{max}\_RPM &= \frac{d-2b}{b} + 1 \\ &= \frac{ib+2b-2b}{b} + 1 = i + 1 \end{aligned} \tag{16}$$

  As '2b' bytes are reserved to code the Node_IDs and index_numbers for leaf nodes along 'b' branches and each non-leaf Node_ID is added b times in the topology data, so the remaining bytes can represent non-leaf nodes along $\frac{d-2b}{b}$ hops. 1 is added to $\frac{d-2b}{b}$ hops to include the hop for the leaf nodes.
- A maximum of r_{max}_RPM redundant nodes can be present in NT_Data_RPM, where:

$$\begin{aligned} r_{max}\_RPM &= (h_{max}\_RPM - 1) * (b - 1) \\ &= i * (b-1) \end{aligned} \tag{17}$$

  Here 1 is subtracted from h_{max}_RPM to exclude the last hop $i + 1$ that contains leaf nodes. For the remaining i hops, a redundancy of $b - 1$ is generated at each hop as each non-leaf Node_ID is added b times in NT_Data_RPM.
- With r_{max}_RPM redundant nodes in NT_Data_RPM, a maximum of n_{max}_RPM nodes can be coded with the RPM method, where:

$$\begin{aligned} n_{max}\_RPM &= n_0\_RPM - r_{max}\_RPM \\ &= (d-b) - i*(b-1) = i + b \end{aligned} \tag{18}$$

  Here r_{max}_RPM is subtracted from n_0_RPM to get the number of unique Node_IDs present in NT_Data_RPM.
- With r_{max}_RPM redundant nodes in NT_Data_RPM, maximum possible redundancy R_{max}_RPM in NT_Data_RPM is calculated as:

$$R_{max}\_RPM = \frac{r_{max}\_RPM}{n_0\_RPM} * 100 \tag{19}$$

Comparing equations 6 and 16, RPM performs better than the PM method when:

$$\begin{aligned} h_{max}\_RPM &> h\_PM \\ \frac{d-2b}{b} + 1 &> 3 \\ \frac{d}{b} - 1 &> 3 \\ \frac{d}{b} &> 4 \\ b &> \frac{d}{4} \end{aligned} \tag{20}$$

When b is equal to $\frac{d}{4}$, both RPM and PM can code the same number of maximum hops (i.e. 3) in the topology data. When b is greater than $\frac{d}{4}$, the RPM method can only code the topology data if all the b branches terminate either at hop 1 or hop 2.

Comparing equations 5 and 18, RPM performs better than the FM method when:

$$\begin{aligned} n_{max}\_RPM &> n\_FM \\ (d-b) - i*(b-1) &> \frac{d}{2} \end{aligned} \tag{21}$$

which is possible, when:

$$\begin{aligned} i &< \frac{d-2b}{b} \\ b &< \frac{d}{i+2} \end{aligned} \tag{22}$$

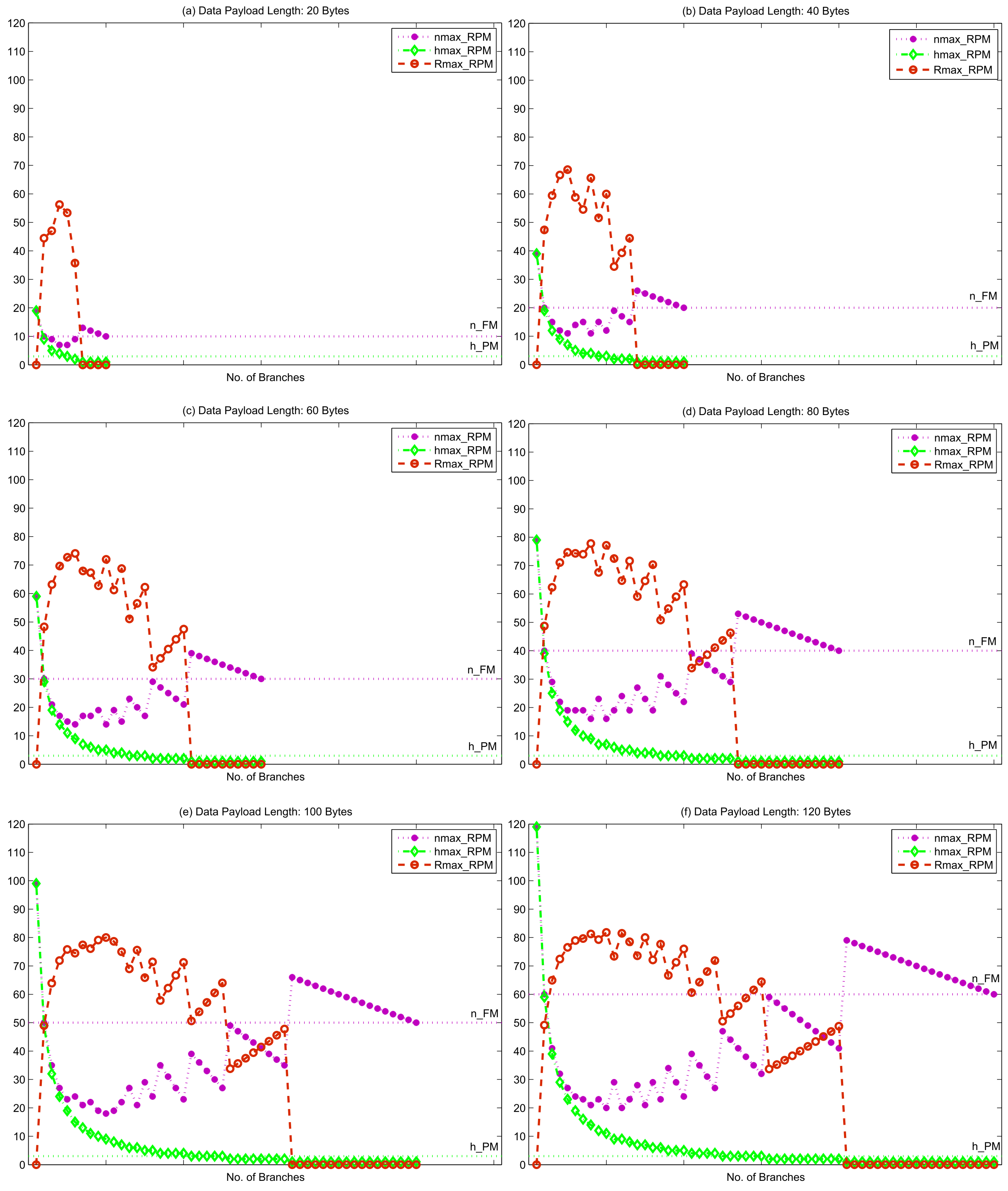

FIGURE 8. **Comparing RPM against FM and PM under worst routing topology.**

When b is equal to $\frac{d}{i+2}$, both RPM and FM can code the same number of maximum nodes in the topology data. When b is greater than $\frac{d}{i+2}$, it is no longer possible to code the topology data using the RPM method with 'd' bytes.

Fig. 8 compares h$_{max}$_RPM against h_PM and n$_{max}$_RPM against n_FM for maximum possible redundancy (i.e. R$_{max}$_RPM) for data payload lengths of 20, 40, 60, 80, 100 and 120 bytes respectively. Based on the results, on average, RPM gives a 40% better performance than the PM method and a 35% better performance than the FM method under the worst routing topology.

To evaluate the RPM method, we have designed an Aggregation based Topology Learning (ATL) protocol using this method. In the remaining part of this paper, we discuss the design, implementation and evaluation of the ATL protocol.

IV. ATL DESIGN PRELIMINARIES

This section describes the design preliminaries for the ATL protocol, i.e. the notation, assumptions and design requirements used for the protocol design.

A. NOTATIONS

The notations used in the description of the ATL protocol are summarised in Table 1.

TABLE 1. Notations used in protocol.

Symbol	Meaning
$\mathcal{BS}$	Base Station
$\mathcal{A}$	Aggregator node
$\mathcal{L}$	Leaf node
Q_{BS}	Query message sent by $\mathcal{BS}$
TD_{Hop}	Time taken to deliver a message over one hop
h	Estimated number of hops in the network
$TD_{Network}$	Estimated time to deliver a message over h hops
h_X	Hop count of a node X
TO_X	Timeout value set by X on receiving Q_{BS}
QR_X	Query response message sent by X
TL Algorithm	Topology Learning Algorithm
TDA Algorithm	Topology Data Aggregation Algorithm

B. ASSUMPTIONS

The following assumptions are used in the design of the ATL protocol:

(A1) There is only one $\mathcal{BS}$ in the network and there are no energy and memory constraints on the $\mathcal{BS}$.
(A2) The nodes in the network are organized in a tree structure where each node is connected to exactly one 1-hop upstream neighbour (i.e. its parent) at any given time. However each node may have any number of 1-hop downstream neighbours (i.e. its child nodes).
(A3) For simplicity, trees or subtrees are collectively called trees, and the head of a tree is referred to as tree-head.
(A4) The parent and child relationships are already established prior to the execution of the ATL protocol.
(A5) Each node knows its hop count from the $\mathcal{BS}$.
(A6) The clocks of all nodes are synchronised with the $\mathcal{BS}$.
(A7) The ATL protocol collects topology data in uniform length z intervals.
(A8) A tree-head collects and processes NT_Data_RPM from each of its child nodes independent of the other child nodes during the z interval, i.e. a message carrying NT_Data_RPM is processed at the tree-head as soon as it is received.
(A9) A tree-head forms its own NT_Data_RPM at the completion of the z interval.
(A10) The ATL protocol does not deal with data integrity issues in the network.

C. DESIGN REQUIREMENTS

Three requirements have been specified for the design of the ATL protocol, and these are:

(R1) Completeness: The routing topology learnt should contain all the active nodes in the network.
(R2) Correctness: The routing topology learnt should correctly capture the topological connections among the nodes in the network. The routing topology learnt should also correctly identify the high-energy consuming nodes, that are likely to create energy holes, in the network.
(R3) Energy Costs: The energy cost incurred in learning the routing topology should be as low as possible.

V. THE ATL PROTOCOL

This section gives an overview of the ATL protocol and describes the detailed design of the protocol including the messages and the algorithms that it uses.

A. PROTOCOL OVERVIEW

The ATL protocol uses the pull-based data collection method, whereby topology data collections are periodically initiated by the $\mathcal{BS}$ through issuing a query message once every z interval [28].

The working of the ATL protocol, based on the four functional blocks (discussed later in Section V-B), can be described in two phases: (i) query construction and transmission, and (ii) topology learning, topology data aggregation and query response transmission.

In the first phase, the $\mathcal{BS}$ constructs and broadcasts a query message Q_{BS} to its 1-hop downstream neighbours (e.g. $\mathcal{A}$). $\mathcal{A}$, in turn, forwards Q_{BS} further in the network to its 1-hop downstream neighbours. This process continues in the same manner till Q_{BS} reaches the $\mathcal{L}$ nodes.

In the second phase, starting at the $\mathcal{L}$ nodes, each node packs the topology data into a query response message QR_L and sends it to its 1-hop upstream neighbour (e.g. $\mathcal{A}$). A learns the topology of the tree headed by itself based on the topology data carried in the received QR_L messages, generates new topology data for this tree (which involves the aggregation of its own topology data with those received), packs the new topology data into QR_A and sends it to a 1-hop upstream neighbour. The process continues at each hop until the $\mathcal{BS}$ receives QR messages from its 1-hop downstream neighbours and learns the routing topology.

Based on the topology data carried in the QR messages and the data structure depicted in Fig. 3, the $\mathcal{BS}$ learns the downstream routing topology. Fig. 9 illustrates the two phases and the operations involved in these phases.

B. PROTOCOL IN DETAIL

The ATL protocol consists of four functional blocks, i.e. Query Construction and Transmission, Query Response Construction and Transmission, Topology Learning and Topology Data Aggregation. The messages and algorithms corresponding to the four building blocks are explained as follows:

1) QUERY CONSTRUCTION AND TRANSMISSION (QUERY MESSAGE FORMAT)

To learn the routing topology of a HWSN, the $\mathcal{BS}$ constructs a query message Q_{BS} at the start of a z interval, to solicit topology data (i.e. NT_Data_RPM) from each sensor node

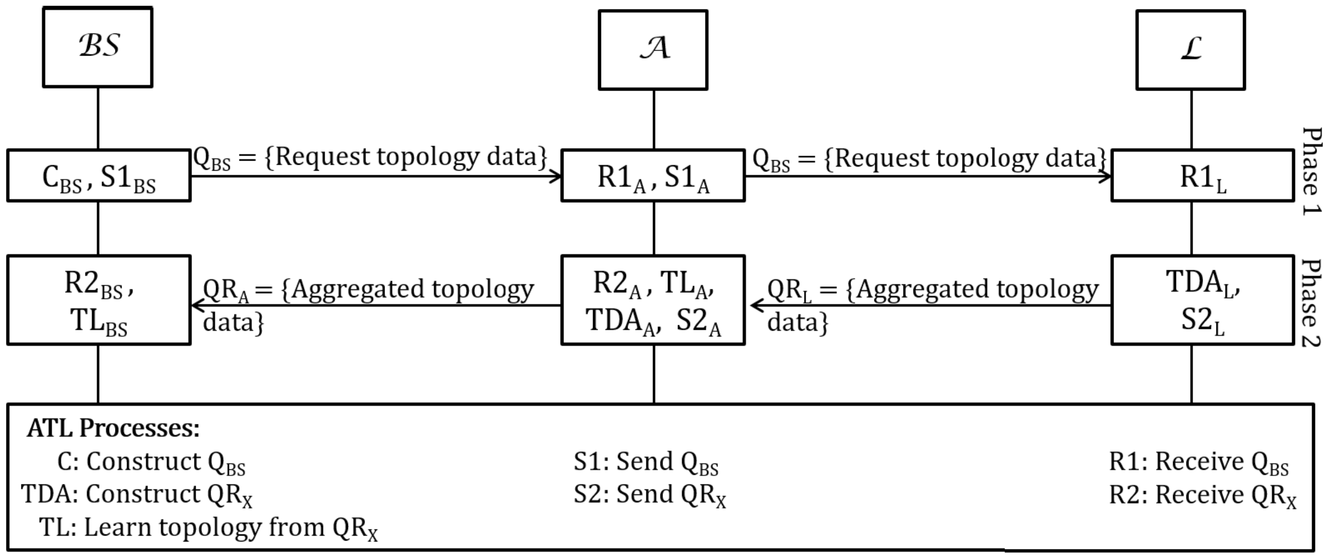

FIGURE 9. Sketch of ATL protocol.

in the network. Q_{BS} has the following structure:

$$Q_{BS} = Node\_ID_{BS}, TD_{Hop}, TD_{Network}$$
$$where\ TD_{Network} = h * TD_{Hop} \quad (23)$$

$TD_{Network}$ is the time by which the $\mathcal{BS}$ expects responses, in the form of topology data, from its 1-hop downstream neighbours. This marks the completion of the z interval.

Once constructed, Q_{BS} is broadcasted by the $\mathcal{BS}$ to its 1-hop downstream neighbours. When a 1-hop downstream neighbour Y receives Q_{BS}, it immediately forwards the query to its 1-hop downstream neighbours. Y also starts a timeout upon the transmission of a Q_{BS}. Timeout at Y, denoted as TO_Y, is given as:

$$TO_Y = TD_{Network} - \{h_Y * TD_{Hop}\} \quad (24)$$

TO_Y sets the time at which Y sends its response, in the form of its topology data, to the $\mathcal{BS}$.

2) QUERY RESPONSE CONSTRUCTION AND TRANSMISSION (QUERY RESPONSE MESSAGE FORMAT)

To respond to the query message Q_{BS}, each node in the network constructs a query response message 'QR'. For a node Y, QR is denoted as QR_Y and contains its topology data, i.e. $NT\_Data\_RPM_Y$. $NT\_Data\_RPM_Y$ contains Y's own topology data aggregated with the topology data from its 1-hop downstream neighbours, received before the expiry of TO_Y. When TO_Y expires, Y sends QR_Y to its 1-hop upstream neighbour.

The structure of the QR message is defined in Fig. 3 and the respective fields are already explained in Section III-C.

3) TOPOLOGY LEARNING (TOPOLOGY LEARNING (TL) ALGORITHM)

The topology learning process at a tree-head is the process of identifying the topological positions of Node_IDs, contained in the QR messages received from its 1-hop downstream neighbours. To do so, each tree-head (including the $\mathcal{BS}$) uses the Topology Learning (TL) algorithm. For a tree-head Y, the input to the TL algorithm is $NT\_Data\_RPM_X$, contained in QR_X, received from a 1-hop downstream neighbour X. The output of the TL algorithm is a routing topology produced for the tree headed by Y.

Upon the receipt of QR_X, the tree-head Y first identifies the Node_ID at the first position in $BEF\text{-}ID_X$ (i.e. $Node\_ID_X$) as a 1-hop downstream neighbour. It then examines the index_numbers in $LNP\text{-}EF_X$. This determines the number of branches present in $BEF\text{-}ID_X$. Y then separates the branches in $BEF\text{-}ID_X$ using the index_numbers in $LNP\text{-}EF_X$. For each such branch, Y determines the hop positions for each node, where the first Node_ID in the branch is a 2-hop downstream neighbour, the second Node_ID is a 3-hop downstream neighbour and so on till the last Node_ID in the branch.

The pseudo code for the TL algorithm at a tree-head Y is provided in Algorithm 1.

Under assumption (A8), a tree-head Y processes the NT_Data_RPM from each of its 1-hop downstream neighbours independently. This means that as soon as Y gets QR_X from a 1-hop downstream neighbour X, it processes $NT\_Data\_RPM_X$ to learn the routing topology conveyed by X.

It is important to mention that there may be message corruptions or losses in the network. As queries are sent periodically at every z interval, the message corruptions or losses are not expected to affect the topology learning in the network.

4) TOPOLOGY DATA AGGREGATION (TOPOLOGY DATA AGGREGATION (TDA) ALGORITHM)

The topology data aggregation process is used by each node to aggregate its own topology data with the topology data

Algorithm 1 Topology Learning (TL) Algorithm

```
%Topology Learning at a Tree-head Y%

Input: NT_Data_RPM$_X$ = {BEF-ID$_X$, LNP-EF$_X$}
Output: Identification of Topological Positions of Nodes in BEF-ID$_X$

1. Identify 1-hop downstream neighbour.
   BEF-ID$_X$[0] is a 1-hop downstream neighbour.

2. Identify the number of branches b in BEF-ID$_X$.
   b = Total index_numbers in LNP-EF$_X$

3. i ← 1

4. For each b:
   4.1. j ← 2
   while i ≤ LNP-EF$_X$[b] do
      4.2.1. BEF-ID$_X$[i] is a j-hop downstream neighbour.
      4.2.2. i = i + 1
      4.2.3. j = j + 1
   end while
```

Algorithm 2 Topology Data Aggregation (TDA) Algorithm

```
%Topology Data Aggregation at a Node Y%

Input: NT_Data_RPM$_X$ = {BEF-ID$_X$, LNP-EF$_X$}
Output: NT_Data_RPM$_Y$ = {BEF-ID$_Y$, LNP-EF$_Y$}

1. Identify the number of branches b in BEF-ID$_X$.
   b = Total index_numbers in LNP-EF$_X$

2. i ← 1

3. For each b:
   3.1. BEF-ID$_Y$ = {BEF-ID$_Y$} || {BEF-ID$_X$[0]}
   while i ≤ LNP-EF$_X$[b] do
      3.2.1. BEF-ID$_Y$ = {BEF-ID$_Y$} || {BEF-ID$_X$[i]}
      3.2.2. i = i + 1
   end while
   3.3. Add index_number i-1 to LNP-EF$_Y$
```

received from its downstream neighbours while it constructs an outgoing QR. To do so, each node uses the Topology Data Aggregation (TDA) algorithm. For a node Y, the input to the TDA algorithm is NT_Data_RPM$_X$, contained in QR$_X$, received from a 1-hop downstream neighbour X. The output of the TDA algorithm is the aggregated topology data produced by Y, i.e. NT_Data_RPM$_Y$.

Upon the receipt of QR$_X$, the node Y first identifies the number of branches in BEF-ID$_X$. For each such branch, it adds BEF-ID$_X$[0] as the first Node_ID and then adds the Node_IDs from the branch into BEF-ID$_Y$. Y also determines the index_number for the last Node_ID for each branch in BEF-ID$_Y$ and puts this index_number in LNP-EF$_Y$.

The pseudo code for the TDA algorithm at a node Y is provided in Algorithm 2.

It is important to understand the working of the TDA algorithm if a node Y has no downstream neighbours. If Y receives no QR messages by the time when TO$_Y$ expires, it produces NT_Data_RPM that contains just Node_ID$_Y$ in BEF-ID$_Y$ and a 0 value in LNP-EF$_Y$, i.e. NT_Data_RPM$_Y$ = {Node_ID$_Y$, {}}, {0}.

At the expiry of TO$_Y$, the node Y sends QR$_Y$, containing NT_Data_RPM$_Y$, to its 1-hop upstream neighbour. This contains aggregated topology data for Y and all its 1-hop downstream neighbours, from which topology data was received before the expiry of TO$_Y$.

VI. EXPERIMENTAL EVALUATION

This section reports the experiments carried out to assess the completeness, correctness and energy costs of the ATL protocol. It first describes the environment under which the experiments were carried out and then discusses the experimental results and findings.

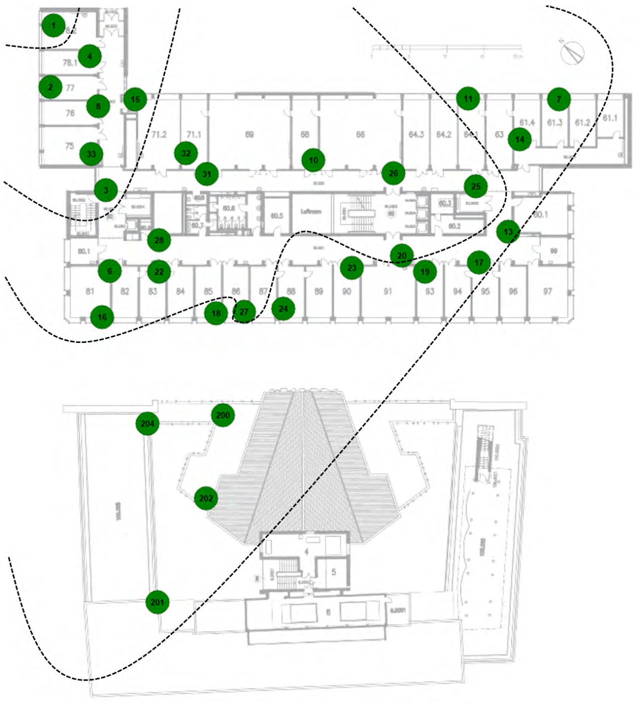

FIGURE 10. Network layout on flocklab test bed.

A. EXPERIMENT ENVIRONMENT AND SETUP

We have used the Flocklab wireless sensor network test bed [29] to evaluate the ATL protocol. The test bed consists of 31 nodes, out of which 27 nodes are deployed indoors and 4 nodes are deployed outdoors. Fig. 10 shows the layout of the test bed. This layout is generated by using the connectivity map provided on Flocklab website [30]. Based on this layout and with node 1 as the head of the tree (i.e. $\mathcal{BS}$), the test bed has a topology of a hierarchical tree with 4 hops.

The ATL protocol is implemented on TinyOS [26] using NesC [31], a programming language used to build WSN applications on TinyOS platform. Both Q_{BS} and QR messages are routed using CTP (Collection Tree Protocol) [32]. The data payload length of Q_{BS} is fixed at 3 bytes. For the data payload length of QR, two lengths, i.e. 27 bytes and 42 bytes, have been selected to investigate the impact of data payload length on completeness, correctness and energy costs of ATL.

The time for each experiment run is fixed at 1200 seconds. Initially, each experiment was run for a time ranging between 600 to 3600 seconds, where 600 seconds is the default duration set in FlockLab XML test configuration file and 3600 seconds is the maximum duration supported by the Flocklab test bed. Fig. 11 shows the percentage of participating nodes versus different run time for the experiments. Since the percentage of participating nodes remains the same after 1200 seconds, so 1200 seconds is chosen as the time for each experiment run. These 1200 seconds are divided into 240 intervals of 5 seconds each and each node has the tendency to generate 240 QR messages during an experiment run. 16 experiments are carried out so each result plotted in the graphs is the average of 3840 results.

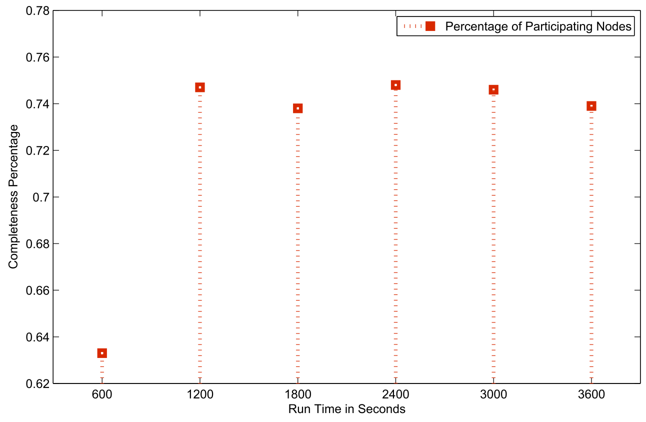

FIGURE 11. Percentage of participating nodes.

The parameter values used in the experiments are summarised in Table 2.

TABLE 2. Experiment Settings

Parameter	Value
Test bed	Flocklab
Number of Nodes	30 sensor nodes and a BS
Field Size	Across one level of ETZ-building and surrounding rooftops at ETH Zurich
BS Location	Top left
Data Payload Length for Q_{BS}	3 bytes
Data Payload Length for QR message	27 bytes and 42 bytes
Simulation Time	1200 s
Number of Repeated Runs	16

B. RESULTS AND DISCUSSIONS

As the energy consumed by a node is largely dependent on its communication load, we have considered two factors in the evaluation of the ATL protocol. These two factors are: data payload length and number of messages received and transmitted by a node. To investigate the impact of these two factors on the ATL protocol, we have considered four scenarios in our experiments:

- Scenario 1 (S1): data payload length is 27 bytes and all the nodes in the network receive and transmit uniform number of messages.
- Scenario 2 (S2): data payload length is 27 bytes and different nodes in the network receive and transmit non-uniform number of messages.
- Scenario 3 (S3): data payload length is 42 bytes and all the nodes in the network receive and transmit uniform number of messages.
- Scenario 4 (S4): data payload length is 42 bytes and different nodes in the network receive and transmit non-uniform number of messages.

For S1 and S3, we excluded node 3 while running the experiments. This is because, from the connectivity map provided on the Flocklab website [30], it was observed that 20 of the 30 nodes are within the communication radius of node 3. If node 3 is included in the network, it will bear much higher communication load than other nodes in the network, violating our assumption used in carrying out the experiments. For S3 and S4, we increased the data payload length for QR messages. The reason for this was that, when working with S1 and S2, it was noticed that topology data for some nodes was not reported to the $\mathcal{BS}$, even when it was present in the QR messages of the nodes in the lower hops. Increasing the data payload length for a QR message allows $\mathcal{BS}$ to receive topology data from all the nodes in the network.

In the following sections, we evaluate the ATL protocol against the three design requirements mentioned in Section IV-C. We first give the detailed results for S1 from this evaluation. We then compare the results for all the four scenarios and present our findings.

1) S1: SMALL DATA PAYLOAD LENGTH AND UNIFORM NUMBER OF MESSAGE EXCHANGES ACROSS THE NETWORK

The results of the protocol run for S1 are as follows:

(i) **Completeness:** The $\mathcal{BS}$ gets data from 26 nodes, which represents 89.66% of the network for a total of 29 nodes in the network.

(ii) **Correctness:** Fig. 12 shows the learnt routing topology at the $\mathcal{BS}$ on the Flocklab test bed. From Fig. 12, we see that the $\mathcal{BS}$ has five 1-hop downstream neighbours (nodes 2, 4, 8, 15, 33), four 2-hop downstream neighbours (nodes 6, 16, 31, 32), ten 3-hop downstream neighbours (nodes 10, 18, 20, 22, 25, 26, 27, 28, 200, 202) and seven 4-hop downstream neighbours

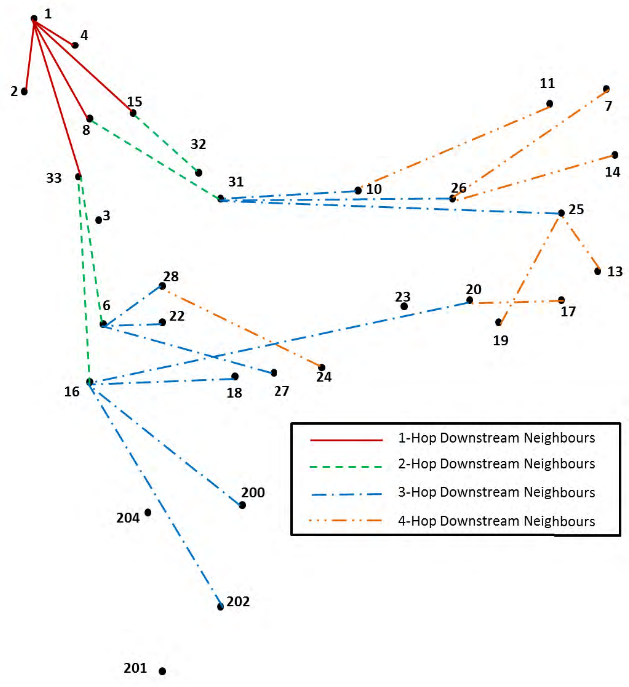
1-Hop Downstream Neighbours
2-Hop Downstream Neighbours
3-Hop Downstream Neighbours
4-Hop Downstream Neighbours

FIGURE 12. Learnt routing topology in S1.

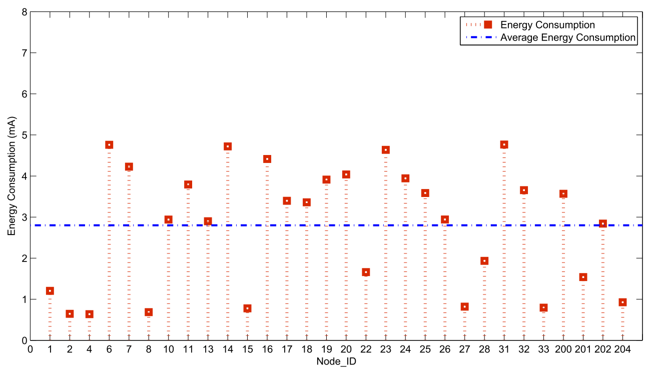

FIGURE 13. Node-wise energy consumption in S1.

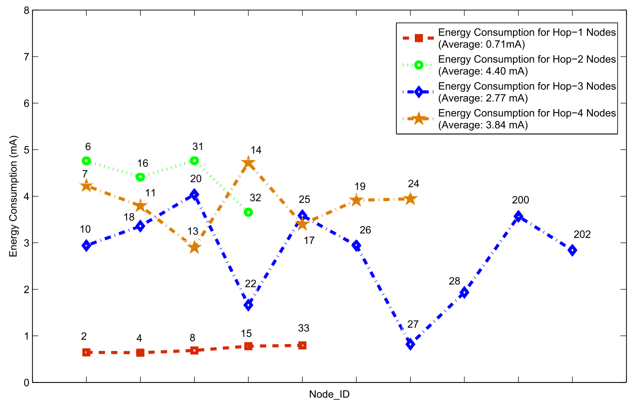

FIGURE 14. Hop-wise energy consumption in S1.

(nodes 7, 11, 13, 14, 17, 19, 24). Comparing this with the real network layout shown in Fig. 10, we can see that the test bed actually forms a 5-hop network instead of a 4-hop network. 12 nodes are identified as being present in the same hop as originally perceived. This represents 41.38% of the topological connections among the nodes. 14 nodes are identified as being present in a lower hop than originally perceived. That is, nodes 10, 20, 22, 25, 26, 27, 28 are identified as being in hop 3 instead of hop 2, and nodes 7, 11, 13, 14, 17, 19, 24 are identified as being in hop 4 instead of hop 3. This represents 48.28% of the topological connections among the nodes. The remaining 3 nodes (nodes 23, 201, 204) are not identified at all, which accounts for 10.34% of the network.

Based on the routing topology learnt in Fig. 12, we have identified two classes of high energy-consuming nodes, likely to create energy holes, in the network. The first class contains nodes {6, 16, 31} and these are the nodes with a large number of 1-hop downstream neighbours. The second class contains nodes {7, 11, 13, 14, 17, 19} and these are the nodes that connect to different 1-hop upstream neighbours (i.e. nodes {10, 20, 23, 24, 25, 26}) at different times. In the following section, we verify if these nodes are actually the high energy-consuming nodes in the network.

(iii) **Energy Costs:** To investigate the energy costs imposed on each node, for computation and communication of NT_Data_RPM, we have used the power profiling service in FlockLab. Fig. 13 shows the energy consumption by each node in the test bed. The average energy consumption is measured to be 2.80 mA and the energy consumption for different nodes varies between 0.64 mA to 4.77 mA.

To further investigate the variations in energy consumption per node, we put the nodes into four groups based on their hop counts (as determined in Fig. 12) and plot their energy consumption in Fig. 14. From the results shown in the figure, it can be seen that the nodes that consume the lowest level of energy are the nodes in the hop-1 group, i.e. nodes 2, 4, 8, 15, 33. As these nodes receive QR messages from just one or two nodes in the hop-2 group, these nodes receive, process and transmit the fewest QR messages in the network. On contrary, the energy consumed by the nodes in the hop-2 group, i.e. nodes 6, 16, 31, 32 is the highest compared to the nodes in the other groups. As these nodes receive QR messages from the nodes in the hop-3 group, which contains the highest number of nodes in the network, these nodes receive, process and transmit the most QR messages in the network. The energy consumed by nodes in the hop-3 group, i.e. nodes 10, 18, 20, 22, 25, 26, 27, 28, 200, 202 is lower than that by the nodes in the hop-2 group. This result is also within our expectation as there are fewer nodes in the hop-4

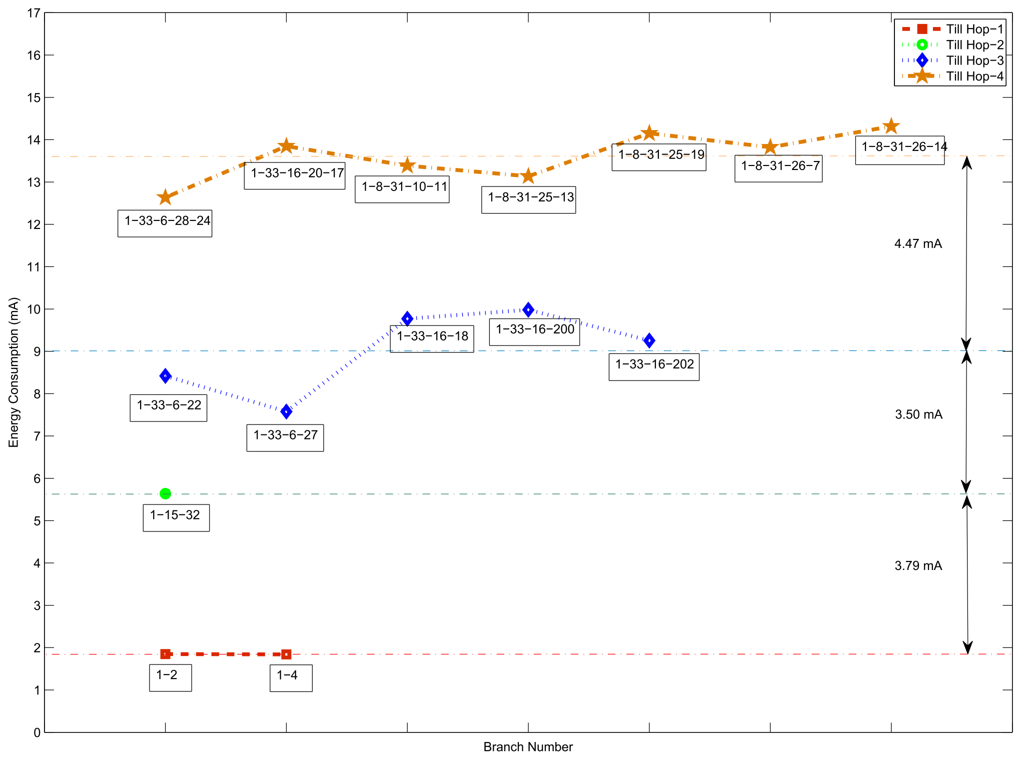

FIGURE 15. **Branch-wise energy consumption in S1.**

group than there were in the hop-3 group. The energy consumed by nodes in the hop-4 group, i.e. nodes 7, 11, 13, 14, 17, 19, 24 is also lower than that by the nodes in the hop-2 group but higher than the nodes in the hop-3 group. We further investigated the high energy consumption by these nodes and examined the topology data communicated for these nodes. From the topology data, we found that these nodes connect to different nodes in hop-3 group at different times, e.g. node 14 connected to node 25 at one time, while at another time it connected to node 26 instead. This behaviour indicates a routing inconsistency and CTP allows a node to resolve routing inconsistencies by broadcasting routing beacons [32]. A similar behaviour is observed for all the nodes in hop-4 group and the broadcast of routing beacons explains the high energy consumption by the nodes in the hop-4 group. It is interesting to note that all of these nodes are confined to the top-right region and are farthest from the $\mathcal{BS}$, as shown in Fig. 10. For future reference, we call the group of nodes in the top-right region (i.e. nodes 7, 11, 13, 14, 17, 19) as the Energy_Drain_1 group and the nodes within their broadcast range (i.e. nodes 10, 20, 23, 24, 25, 26) as the Energy_Drain_2 group.

To verify the observations made above, we have plotted nodes' energy consumptions based on tree branches terminating at different hops in Fig. 15. We have identified 15 branches in the network and classified them into four different groups based on the hops at which these branches terminate. There are 2 branches terminating at hop-1, 1 branch terminating at hop-2, 5 branches terminating at hop-3 and 7 branches terminating at hop-4. From the figure, we can make two observations. Firstly, branches in the same group (i.e. terminating at the same hop) have a similar level of energy consumption. Secondly, branches in different groups consume different levels of energy, and the differences are consistent and obvious. On average, each hop increases energy consumption by 3.92 mA in a branch.

To verify if the high energy-consuming nodes have been correctly identified by the ATL protocol, we plot the nodes' energy consumptions on a normal distribution in Fig. 16. There are 14 nodes (excluding the outdoor node 200) with energy consumptions greater than the average energy consumption and out of these 14 nodes, 12 nodes have been identified by the ATL protocol, thus providing an accuracy of 85.71%.

To further assess the cost incurred by topology learning using the ATL protocol, we conducted another set of experiments to assess the energy consumption by each node when the topology learning feature is enabled versus when the topology learning feature is disabled. The results are shown in Fig. 17. From the results, it can be seen that the energy consumption in the two cases are at a similar level, averaging at 2.80 mA and 2.99 mA respectively. The results indicate that when the

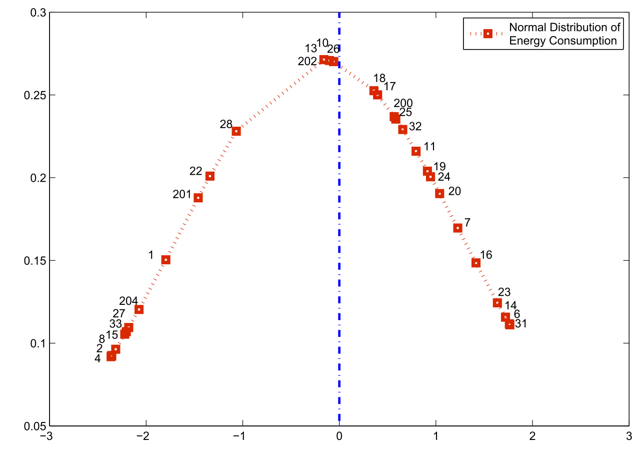

FIGURE 16. Normal distribution of node-wise energy consumption in S1.

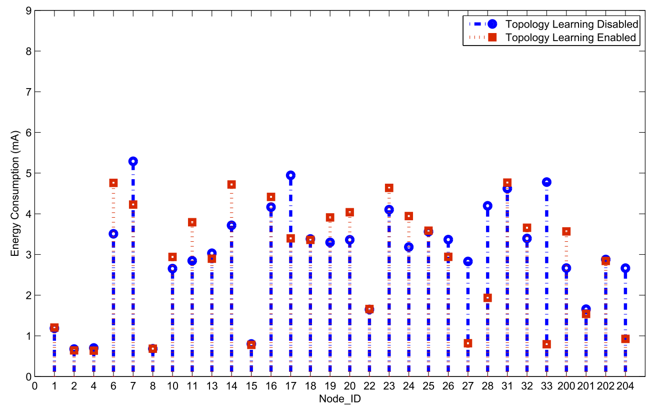

FIGURE 17. Energy consumption per node when topology learning is enabled vs when it is disabled.

topology learning feature is enabled, the extra energy cost incurred is negligible, so we can say that the topology learning method implemented in the ATL protocol is energy-efficient.

As demonstrated in Sections III-C, the performance of the RPM method greatly depends on the routing topology and the redundancy introduced by it in the topology data. We now measure the redundancy present in the QR messages, carrying the topology data, that are received at the $\mathcal{BS}$. We call a byte carrying a redundant Node_ID as a redundant byte. Table 3 shows the number of redundant bytes present in the QR messages received at the $\mathcal{BS}$. From the table, it can be seen that for a total of 52 used bytes, there are 12 redundant bytes, representing 23.1% of the used bytes. However, the data payload length of a QR message is 27 bytes, which means that there are 12 redundant bytes for a total of 135 bytes, representing 8.9% of the total bytes.

2) COMPARISON BETWEEN THE FOUR SCENARIOS

Table 4 summaries the results collected for all the four scenarios. From these results, we can make the following observations:

TABLE 3. Redundant Node_IDs in QR Messages

Node_ID	Used Bytes	Redundant Bytes
2	2	0
4	2	0
8	3	0
15	21	7
33	24	5

(i) **Completeness:** The routing topology learnt in S3 and S4 contains all the active nodes in the network, while the routing topology learnt in S1 contains 26 of the 29 active nodes and the routing topology learnt in S2 only contains 20 of the 30 active nodes. Due to the small data payload length of the QR message, topology data from some nodes can not be added in the QR message of the upstream neighbours. This affects a larger number of nodes in S2 as compared to S1, as a lot of nodes are dependent on node 3 in S2.

(ii) **Correctness:** The closest match to the real network layout, shown in Fig. 10, is observed only in S4 where 86.67% nodes are identified as being present in the same hop as originally perceived. With respect to the identification of high energy-consuming nodes, S1 performs the best where 85.71% of the high energy-consuming nodes are correctly identified by ATL. It is important to state that in both S3 and S4, due to increased data payload length, 27 nodes have energy consumption higher than the average energy consumption. To calculate correctness in both these scenarios, we have considered very high energy-consuming nodes, which constitute 5 nodes in S3 and 13 nodes in S4 respectively.

(iii) **Energy Costs:** Fig. 18 shows the energy consumption by each node in the test bed in the four scenarios. The two dotted straight lines in each figure indicate the average energy consumption in the respective scenarios.

In all four scenarios, the hop-1 group contains almost the same number of nodes. However, the energy consumption in the hop-1 group is much higher in S3 and S4 as compared to S1 and S2, due to large data payload length of QR message in S3 and S4. A large number of nodes are present in the hop-2 group in S2 and S4 as compared to S1 and S3. As 20 of the 30 nodes in the test bed are within the communication radius of node 3, more nodes communicate directly with node 3 and appear in the hop-2 group. The energy consumption in the hop-2 group in S1, S3 and S4 is high as compared to the other groups. This is because this group receives and processes QR messages from the hop-3 group, which has the largest number of nodes. The energy consumption in the hop-2 group in S2 is comparatively low as there are only seven nodes in its hop-3 group. The energy consumption in the hop-3 group in S1 is a bit lower as compared to S2, S3 and S4. This is because no nodes from the Energy_Drain_1

TABLE 4. Comparing the Four Scenarios

Scenario	**%Completeness**	**%Correctness**			
		Same Hop Nodes	Different Hop Nodes	Unreported Nodes	High Energy-Consuming Nodes
1	89.66	41.38	48.28	10.34	85.71
2	66.67	53.33	13.33	33.33	64.28
3	100	58.62	41.38	0	80.00
4	100	86.67	13.33	0	76.92
Scenario		**Average Energy Consumption (mA)**		**Energy Consumption Variation(mA)**	-
1		2.80		0.64 - 4.77	-
2		2.67		0.62 - 7.38	-
3		3.81		2.71 - 5.49	-
4		4.00		2.64 - 6.14	-
Scenario	**Number of Nodes**				
	Hop-1 Group	Hop-2 Group	Hop-3 Group	Hop-4 Group	Unreported
1	5	4	10	7	3
2	5	7	7	1	10
3	4	5	15	5	-
4	5	10	14	1	-
Scenario	**Average Hop-wise Energy Consumption (mA)**				
	Hop-1 Group	Hop-2 Group	Hop-3 Group	Hop-4 Group	-
1	0.71	4.40	2.77	3.84	-
2	0.82	2.84	3.31	3.63	-
3	3.58	3.87	3.90	4.10	-
4	3.64	4.51	4.02	3.13	-
Scenario	**Number of Branches**				
	Hop-1 Group	Hop-2 Group	Hop-3 Group	Hop-4 Group	Total
1	2	1	5	7	15
2	2	4	6	1	13
3	1	-	11	5	17
4	3	5	13	1	22
Scenario	**Average Branch-wise Energy Consumption (mA)**				
	Hop-1 Group	Hop-2 Group	Hop-3 Group	Hop-4 Group	Average Increase Per Hop
1	1.84	5.64	9.14	13.61	3.92
2	1.78	5.40	7.61	14.05	4.09
3	4.79	-	13.21	17.27	3.84
4	4.99	9.85	13.58	15.90	3.63
Scenario		**% Redundant Bytes vs. Used Bytes**		**% Redundant Bytes vs. Total Bytes**	-
1		23.1		8.9	-
2		10.8		3.0	-
3		20.7		7.1	-
4		14.8		4.3	-

group are present in S1's hop-3 group, whereas two, two and five nodes from the Energy_Drain_1 group are present in the hop-3 groups in S2, S3 and S4 respectively. A small number of nodes are present in the hop-4 group in S2 and S4 as compared to S1 and S3, and their energy consumption is also lower as compared to S1 and S3. This is because only one node from the Energy_Drain_1 group is present in their respective hop-4 groups, whereas six and four nodes from the Energy_Drain_1 group are present in the hop-4 groups in S1 and S3 respectively.

For all the four scenarios, branches in different groups consume different levels of energy, and the differences are consistent and obvious. On average, each hop increases energy consumption by 3.87 mA in a branch. In all four groups, the energy consumption levels in S3 and S4 are noticeably higher than the energy consumption levels observed in S1 and S2. The energy consumption in S1 and S2 is almost at the same level in the first two groups. This is because S2 has no branches containing node 3 in the first two groups. In the third group, energy consumption level in S2

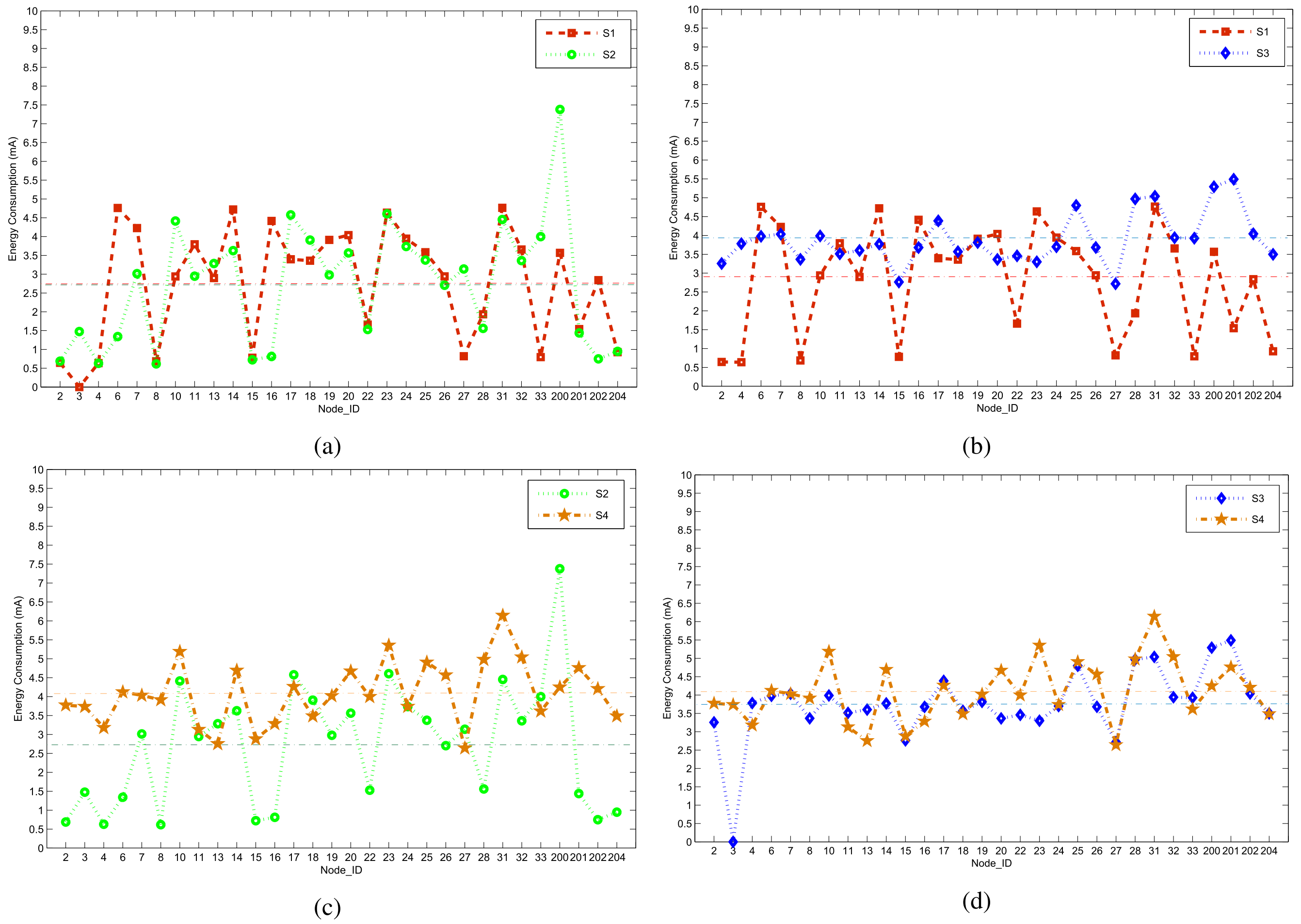

FIGURE 18. Comparing node-wise energy consumption in S1 - S4.

is still lower than that in S1, despite the inclusion of node 3 in one of its branches. This is because all the branches in S1 include either node 6 or node 16, and as discussed in Section VI-B.1, these nodes have high energy consumption levels. In the last group, energy consumption level in S2 is higher than that in S1, as the only branch in S2 includes node 3 and only two of the seven branches in S1 include node 6 and node 16. The energy consumption level in S4 is higher than S3 in the first three groups but lower in the last group. This is because in S4, more nodes communicate directly with node 3 and generate more branches in the first three groups. In the last group, energy consumption level in S4 is lower than that of S3, as there is only one branch in S4 as compared to five branches in S3.

Finally, S2 and S4 have a lower number of redundant bytes as compared to S1 and S3. This is because in S2 and S4, more nodes communicate directly with node 3 and appear in lower hops, thus reducing the number of redundant Node_IDs carried in the QR messages.

3) FINDINGS

The findings from our experiments are summarised as follows:

- By using the ATL protocol, high energy-consuming nodes, i.e. nodes that more likely create energy holes in a HWSN, can be identified.
- The energy consumed by a node is largely dependent on its communication load (i.e. the length and number of messages received and transmitted by the node). In other words, the energy consumed by a node is low when it receives and transmits only a few messages with a small data payload length and high when it receives and transmits many messages with a large data payload length.
- The energy consumed by a node is also dependent on the routing conditions in the network. In other words, the energy consumed by a node is low when it consistently connects to the same node throughout the network operation and is high when it connects to different nodes at different times.
- The routing topology learnt using the ATL protocol is dependent on the communication load imposed on each

node in the network. With a small data payload length, the routing topology learnt may be incomplete. This incompleteness is more noticeable if different nodes in the network also receive and transmit non-uniform number of messages. With a large data payload length, the routing topology learnt is likely to be complete for both uniform and non-uniform number of messages received and transmitted by different nodes in the network.

- The number of nodes appearing in each hop is also dependent on the communication load imposed on each node in the network. With a small data payload length and uniform number of message exchanges throughout the network, more nodes are likely to appear in higher hops, i.e. towards leaf end of the network. On the other hand, with a large data payload length and non-uniform number of message exchanges throughout the network, more nodes are likely to appear in lower hops, i.e. towards the $\mathcal{BS}$.
- The energy cost introduced at each node by the topology learning capability, facilitated by the ATL protocol, is negligible when this capability is implemented as an integral part of a sensed data collection and aggregation process.

VII. CONCLUSION

This paper has presented the design and evaluation of a novel ATL protocol for randomly deployed HWSNs. The protocol is designed for tree-structured WSNs and aims to identify energy holes in such a network by collecting and aggregating topology data from the nodes in the network, while keeping the energy costs to a minimum. This is achieved by proposing and investigating three topology coding methods, i.e. FM, PM and RPM methods. By carrying out a theoretical analysis of the three topology coding methods, the RPM method has been found to carry maximum amount of topology data among the three methods and is therefore used in the ATL protocol. Performance evaluations of the ATL protocol have been conducted on a real-world WSN test bed to evaluate its completeness, correctness, and energy costs under four different scenarios. The results of the four scenarios have also been compared for node-wise, hop-wise and branch-wise energy consumption. Based on the evaluation results, we have identified two classes of high energy-consuming nodes in the test bed: (1) nodes that have a large number of 1-hop downstream neighbours, and (2) nodes that connect to different 1-hop upstream neighbours at different times. Using the received topology data, the $\mathcal{BS}$ can identify nodes falling in these two classes. Furthermore, the energy cost of the ATL protocol is at a negligible level. This means that using the ATL protocol, the high energy-consuming nodes that are likely to create energy holes in a randomly deployed HWSN can be identified in a cost-efficient manner. An important consideration with the ATL protocol, however, is setting the optimum data payload length for the messages carrying the topology data; it should be small enough to minimize the energy costs and large enough to carry the complete routing topology. In our future work, we will study how to set this optimum data payload length.

REFERENCES

[1] X. Liu, "A survey on clustering routing protocols in wireless sensor networks," *Sensors*, vol. 12, no. 8, pp. 11113–11153, 2012.

[2] B. Deb, S. Bhatnagar, and B. Nath, "A topology discovery algorithm for sensor networks with applications to network management," Rutgers Univ., DCS Tech. Rep. DCS-TR-441, May 2001.

[3] J. Staddon, D. Balfanz, and G. Durfee, "Efficient tracing of failed nodes in sensor networks," in *Proc. 1st ACM Int. Workshop Wireless Sensor Netw. Appl.*, 2002, pp. 122–130.

[4] D. Marinakis, G. Dudek, and D. J. Fleet, "Learning sensor network topology through monte carlo expectation maximization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Apr. 2005, pp. 4581–4587.

[5] L. R. Peralta, L. M. Brito, and E. I. Hernández, "A formal graph-based model applied to cluster communication in wireless sensor networks," in *Proc. 8th Int. Conf. Sensor Technol. Appl. (SENSORCOMM)*, 2014, pp. 137–146.

[6] Q. Chen, S. S. Kanhere, and M. Hassan, "Analysis of per-node traffic load in multi-hop wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 2, pp. 958–967, Feb. 2009.

[7] Z. Cheng, M. Perillo, and W. B. Heinzelman, "General network lifetime and cost models for evaluating sensor network deployment strategies," *IEEE Trans. Mobile Comput.*, vol. 7, no. 4, pp. 484–497, Apr. 2008.

[8] Y. J. Zhao, R. Govindan, and D. Estrin, "Residual energy scan for monitoring sensor networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, vol. 1. Mar. 2002, pp. 356–362.

[9] J. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring wireless sensor networks," in *Proc. 1st IEEE Int. Workshop Sensor Netw. Protocols Appl.*, Mar. 2003, pp. 139–148.

[10] L. Zhang, S. Chen, Y. Jian, Y. Fang, and Z. Mo, "Maximizing lifetime vector in wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 4, pp. 1187–1200, Aug. 2013.

[11] R. Kacimi, R. Dhaou, and A.-L. Beylot, "Load balancing techniques for lifetime maximizing in wireless sensor networks," *Ad hoc Netw.*, vol. 11, no. 8, pp. 2172–2186, 2013.

[12] R. R. Rout and S. K. Ghosh, "Enhancement of lifetime using duty cycle and network coding in wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 12, no. 2, pp. 656–667, Feb. 2013.

[13] A. Liu, D. Zhang, P. Zhang, G. Cui, and Z. Chen, "On mitigating hotspots to maximize network lifetime in multi-hop wireless sensor network with guaranteed transport delay and reliability," *Peer-to-Peer Netw. Appl.*, vol. 7, no. 3, pp. 255–273, 2014.

[14] J. Ren, Y. Zhang, K. Zhang, A. Liu, J. Chen, and X. S. Shen, "Lifetime and energy hole evolution analysis in data-gathering wireless sensor networks," *IEEE Trans. Ind. Informat.*, vol. 12, no. 2, pp. 788–800, Apr. 2016.

[15] M. A. Khan, A. Sher, A. R. Hameed, N. Jan, J. S. Abassi, and N. Javaid, "Network lifetime maximization via energy hole alleviation in wireless sensor networks," in *Proc. Int. Conf. Broadband Wireless Comput., Commun. Appl.*, 2016, pp. 279–290.

[16] N. Jan *et al.*, "A balanced energy-consuming and hole-alleviating algorithm for wireless sensor networks," *IEEE Access*, vol. 5, pp. 6134–6150, 2017.

[17] K.-H. Lee, J.-H. Kim, and S. Cho, "Power saving mechanism with network coding in the bottleneck zone of multimedia sensor networks," *Comput. Netw.*, vol. 96, pp. 58–68, Feb. 2016.

[18] S. Gehlaut, J. Koti, and K. Sakhardande, "To improve the lifetime of wireless sensor network," in *Proc. Int. Conf. Invent. Comput. Technol. (ICICT)*, vol. 2. Aug. 2016, pp. 1–5.

[19] R. E. Mohemed, A. I. Saleh, M. Abdelrazzak, and A. S. Samra, "Energy-efficient routing protocols for solving energy hole problem in wireless sensor networks," *Comput. Netw.*, vol. 114, pp. 51–66, Feb. 2017.

[20] J. Roselin, P. Latha, and S. Benitta, "Maximizing the wireless sensor networks lifetime through energy efficient connected coverage," *Ad Hoc Netw.*, vol. 62, pp. 1–10, Jul. 2017.

[21] S. Halder and A. Ghosal, "Lifetime enhancement of wireless sensor networks by avoiding energy-holes with Gaussian distribution," *Telecommun. Syst.*, vol. 64, no. 1, pp. 113–133, 2017.

[22] H. S. Ramos, A. Boukerche, A. L. Oliveira, A. C. Frery, E. M. Oliveira, and A. A. Loureiro, "On the deployment of large-scale wireless sensor networks considering the energy hole problem," *Comput. Netw.*, vol. 110, pp. 154–167, Dec. 2016.

[23] Y. Xue, X. Chang, S. Zhong, and Y. Zhuang, "An efficient energy hole alleviating algorithm for wireless sensor networks," *IEEE Trans. Consum. Electron.*, vol. 60, no. 3, pp. 347–355, Aug. 2014.

[24] L. Fei, Y. Chen, Q. Gao, X.-H. Peng, and Q. Li, "Energy hole mitigation through cooperative transmission in wireless sensor networks," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 2, p. 757481, Feb. 2015.

[25] P. Chanak, I. Banerjee, and H. Rahaman, "Load management scheme for energy holes reduction in wireless sensor networks," *Comput. Electr. Eng.*, vol. 48, pp. 343–357, Nov. 2015.

[26] P. Levis *et al.*, "TinyOS: An operating system for sensor networks," in *Ambient Intelligence*, vol. 35. Berlin, Germany: Springer, 2005, pp. 115–148.

[27] *Cc2420 Datasheet Reference SWRS041B*, document CC2420, Texas Instruments, Dallas, TX, USA, 2007. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf

[28] F. Hongping and F. Kangling, "Overview of data dissemination strategy in wireless sensor networks," in *Proc. Int. Conf. E-Health Netw., Digit. Ecosyst. Technol. (EDT)*, vol. 1. Apr. 2010, pp. 260–263.

[29] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2013, pp. 153–165.

[30] *FlockLab—Connectivity Map*. Accessed: May 22, 2017. [Online]. Available: https://www.flocklab.ethz.ch/user/topology.php

[31] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," *ACM Sigplan Notices*, vol. 38, no. 5, pp. 1–11, 2003.

[32] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo, "The collection tree protocol (CTP)," *TinyOS TEP*, vol. 123, no. 2, 2006. [Online]. Available: http://www.academia.edu/download/45924902/tep123.pdf

AYESHA NAUREEN received the B.Eng. degree in software engineering and the M.S. degree in information security from the National University of Sciences and Technology, Pakistan, in 2005 and 2008, respectively. She is currently pursuing the Ph.D. degree with the School of Computer Science, The University of Manchester, U.K. Her research interests include security, privacy, protocol design, and performance analysis in wireless sensor networks.

NING ZHANG received the B.Sc. degree (Hons.) in electronic engineering from Dalian Maritime University, China, and the Ph.D. degree in electronic engineering from the University of Kent, Canterbury. Since 2000, she has been with the School of Computer Science, The University of Manchester, U.K., where she is currently a Senior Lecturer. Her research interests are in security and privacy in networked and distributed systems, such as ubiquitous computing, electronic commerce, wireless sensor networks, and cloud computing with a focus on security protocol designs, risk-based authentication and access control, and trust management.

STEVE FURBER CBE FRS FREng received the B.A. degree in mathematics and the Ph.D. degree in aerodynamics from the University of Cambridge, U.K. He is an ICL Professor of computer engineering with the School of Computer Science, The University of Manchester, U.K. He spent the 1980s at Acorn Computers, where he was a Principal Designer of the BBC Microcomputer and the ARM 32-b RISC Microprocessor. Over 100 billion variants of the ARM processor have since been manufactured, powering much of the world's mobile and embedded computing. He moved to the ICL Chair at Manchester in 1990 where he leads research into asynchronous and low-power systems and, more recently, neural systems engineering, where the SpiNNaker project is delivering a computer incorporating a million ARM processors optimized for brain modeling applications.