PAFusion: A Power-Aware Framework for Distributed Data Fusion Application in Wireless Sensor Networks

Zongqing Lu*, Su-Lim Tan†
School of Computer Engineering
Nanyang Technological University
Singapore
{luzo0002*,asslTan†}@ntu.edu.sg

Jit Biswas
Institute for Infocomm Research
Agency for Science, Technology and Research
Singapore
biswas@i2r.a-star.edu.sg

***Abstract*—Distributed data fusion application in wireless sensor networks is very promising, because the change from centralized model to distributed model will distribute the computation and consequently power consumption of data fusion from sink node to the entire network nodes. More importantly, placement data fusion functions in networks can also decrease data transmission cost and sink node can easily and efficiently re-task different and multiple applications for the same network structure. In order to achieve these advantages, data fusion functions of application need to be placed at optimal nodes with minimal power consumption and to be dynamically transferred between networks nodes to prevent nodes from being exhausted of power. In this paper, PAFusion a power-aware framework for distributed data fusion applications in wireless sensor networks is proposed. It is designed for mapping distributed data fusion application into networks. We have evaluated its performance compared to DFuse from placement properties, placement delay and control overhead through simulation. The simulation results show PAFusion has better performance than DFuse.**

I. Introduction

To overcome sensor failures, technological limitations, spatial, and temporal coverage problems in WSNs, three properties must be ensured: cooperation, redundancy, and complementarity [1][2]. Usually, a region of interest can only be fully covered by the use of several sensor nodes, each cooperating with a partial view of the scene; data fusion can be used to compose the complete view from the pieces provided by each node. Redundancy makes the WSNs less vulnerable to failure of a single node, and overlapping measurements can be fused to obtain more accurate data. Complementarity can be achieved by using sensors that perceive different properties of the environment; data fusion can be used to combine complementary data so the resultant data allows inferences that might be not possible to be obtained from the individual measurements [3].

Due to cooperation, redundancy and complementarity properties, WSNs are often composed of a large number of sensor nodes posing a scalability challenge caused by potential collisions and transmissions of redundant data. Regarding the energy restrictions, communication should be reduced to increase the lifetime of the sensor nodes. Thus, data fusion is also important to reduce the overall communication load in the network, by avoiding the transmission of redundant messages [3]. Data fusion can be distributed into network and executed on network nodes, to reduce the data from redundant sensor node, to fuse the information from complementary sensor node and to get the complete view from cooperative nodes, consequently only send the inferences demanded to end user. This distributed data fusion can extremely reduce the data transmission cost and with it there is no need for a centralized node to process the collected information.

In WSNs, data fusion is closely related to data communication. The reason is that due to the limited power sources of current sensor nodes, it is usually desirable to take advantage of the limited computation capacity of sensor nodes to perform in-network data fusion to reduce the overall data traffic. Different distributed computing paradigms have been adopted in WSNs, and depending on the chosen paradigm, data fusion occurs in different ways.

In-network aggregation [4][5][6][7] has been put forward as the most popular computing paradigm for wireless sensor networks. The idea is to take advantage of the node computation capacity and perform the desired data aggregation en route - eliminating data redundancy, minimizing transmissions and thus saving energy. This paradigm shifts the focus from the traditional address centric approaches for networking (finding short routes between pairs of addressable end-nodes) to a more data centric approach (finding routes from multiple sources to a single destination that allows in-network consolidation of redundant data). Although In-Network Aggregation paradigm can reduce the transmission cost by performing aggregation of sensed data at selected sensor nodes in WSNs, there are two major limitations in this paradigm. First, the design of in-network aggregation is application-specific from network protocol, network topology to aggregation function. So re-tasking application is the main limitation of in-network aggregation. Second, the aggregation functions are too simple to perform advanced data fusion en route and to make a inference in-network. Aggregation functions in literature are MAX, MIN, AVERAGE and so on, which can only reduce the redundant

data transmission from homogeneous sensors. So In-Network Aggregation can only minimize the transmission cost of data partially.

Mobile Agent paradigm [8][9][10][11] employs migrating codes to facilitate flexible application re-tasking, local processing, and collaborative signal and information processing. In this model, instead of each sensor node sending raw data or pre-processed data to the processing center like Client-Server model, the processing code is moved to the data locations through mobile agents. The benefits of implementation of Mobile Agent paradigm in WSNs are twofold. First, they can potentially reduce the transmissions of raw data and the bandwidth consumption by moving data processing location at sensor nodes which transmit raw data and expend much energy in Client-Server paradigm. In addition, Mobile Agent paradigm provides an effective way to overcome network latency when the number of nodes is large [8]. However, mobile agents' movements also cost much power, if agent code is larger than raw data sensed by sensor nodes or if mobile agent is sent the sensor sources that are far away from the sink node to gather the sensed data. Mobile Agent paradigm is not suitable to implement the applications, which use advanced data fusion or information processing algorithms to make a inference about monitoring environment, because these algorithms always require many data sources be processed simultaneously. If mobile agent intends to implement these algorithms, it must carry data of previously visited sensor node to next visiting node until all the data is collected, however, which will cost a lot of transmission overhead. If mobile agent just brings sensed information back to the sink node, it has no difference with client-server model.

Active Networks allow the injection of customized programs into the network nodes [12]. Accordingly, this paradigm allows high complexity and customizable computations to be performed within the network. In this case, data fusion may travel in the network as active packets, allowing different methods and applications (even unpredicted ones) to be executed at different moments, instead of storing every possible fusion algorithm into the nodes. This paradigm is especially interesting for at least two scenarios: (i) when we cannot predict the application's behavior (e.g., an exploratory WSN deployed in Mars); and (ii) when we need to design long-lived networks whose applications may need to be remotely changed [3]. Particularly, the DFuse [13], the Maté [14] and the SensorWare [15] frameworks were proposed to implement the Active Networks paradigm into sensor networks.

In Client-Server paradigm, sensor nodes need to transmit sensed information to the sink node individually, where these information will be processed using data fusion algorithm [16][17][18][19] ,so it is not power-efficient computing paradigm for WSNs, where energy constraint is one of the most important issues. As shown in [9], this paradigm is interesting only when the network has few sources and the network is small-scale. In-Network Aggregation is the most popular paradigm in WSNs, which performs data aggregation, while data is routed towards the sink node,to reduce data transmitting cost. The main limitation of In-Network Aggregation is that it cannot re-task application and that data aggregation are too simple to support distributed data fusion algorithm,which can reduce data transmission further and make a inference in-network. In contrast to Client-Server and In-Network Aggregation, Mobile Agent Paradigm saves the network bandwidth and provides an effective way to overcome network latency when the number of nodes is large, but it also does not support distributed data fusion. Because in Active Networks paradigm can easily re-task application and support distributed data fusion algorithm, which can be executed on every network node by transferring binary code in DFuse or by executing scripts in SensorWare or by execute VM language in Maté, Active Networks paradigm can be used to long-lived networks whose applications may need to be remotely changed and networks where distributed data fusion are deployed to minimize energy dissipation and make a inference about monitoring environment in-network.

As we discussed above, data fusion is very important in WSNs and distributed data fusion can extremely reduce the data transmission cost, consequently prolong the longevity of network, and make inference in-network about monitoring environment. Among current four computing paradigms, Client-Server, In-Network Aggregation, Mobile Agent and Active Networks, only Active Networks can support distributed data fusion in WSNs, as discussed. Active Networks paradigm integrated with distributed data fusion has the potential to tremendously reduce energy dissipation in WSNs , where energy conservation is the most challenging issue. The goal of this work is to minimize energy expenditure needed to perform distributed data fusion in Active Networks computing paradigm.

In this paper, we focus on challenges of mapping fusion functions of application task graph to network nodes in WSNs. This mapping has to be designed power efficiently, which means it should not only minimize power consumption of data fusion application, but also optimize the control overhead of itself. In addition, in order to average the power consumption of all the nodes so as to increase network longevity, data fusion functions also need to be transferred between network nodes.

The main contributions of this work can be summarized as following:

- Fusion function placement algorithm: a novel fusion function placement algorithm is designed for mapping all types of single application to network nodes with optimization of power consumption for data fusion application and with good placement properties including distance to optimal location, the delay of placement of application task graph, control overhead and scalability.
- Simulation for scalability: we simulate our PAFusion in large networks with more than one thousand nodes and different sizes of application task graph by QualNet network simulator. It is shown that control overhead and delay of placing application task graph introduced by PAFusion scale linearly with network size and application task graph size.

The rest of the paper is structured as follows. Section 2 gives overview of mapping from application task graph to network nodes, power consumption of distributed data fusion application in WSNs and cost function. Section 3 presents our PAFusion framework. Section 4 shows simulation methodology and Section 5 shows the results of our PAFusion and comparison with DFuse in [20]. Then we present some directions for future research work in Section 6 and conclude in Section 7.

II. Overview of Mapping From Distributed Data Fusion Application to Network Nodes

Mapping from distributed data fusion application to network nodes is to place all fusion functions of data fusion application onto selected network nodes in order to minimize power consumption of application, hence prolong the lifetime of wireless sensor networks.

The inputs of mapping are application task graph consisting of data flow, relationship among fusion functions, data rate of each data source, data ratio (either data contraction or expanding) at each fusion function and each fusion function code, and cost function. The assumptions about networks made in this paper are that all the nodes in networks are reachable from sink node, all of them are homogeneous such as radio transceiver and processor and the routing tables are available in application layer, which include hop count and next hop to a given destination that are needed for making decisions in application task graph mapping process. The mapping process consists two phases, fusion function placement and fusion function transfer maintenance. In the phase of fusion function placement, fusion functions of application are placed on network nodes. And in fusion function transfer maintenance phase, fusion functions are transferred among network nodes to maintain minimized power consumption of data fusion application and average power consumption of all the nodes according to cost function of application. The following gives mathematic definition of mapping from application task graph to network nodes, power consumption of application after application task graph mapping to network nodes and definition of cost function in application.

A. Mapping from Application Task Graph to Network Nodes

Given a sensor network as a graph $N = (V_N, E_N)$, where V_N denotes the node set and E_N denotes the edge set representing the communication links between nodes, and application task graphs $T = (V_M, E_M)$, where V_M denotes fusion function set in data fusion application, E_M denotes data flow set in data fusion application, the goal is to find mappings $a : (V_1, V_2 \cdots V_M) \rightarrow \{v|v \in V_N\}$ and $b : (E_1, E_2 \cdots E_M) \rightarrow \{e|e \in E_N\}$.

A mapping $a : (V_1, V_2 \cdots V_M) \rightarrow \{v|v \in V_N\}$ generates an overlay network of fusion function points to network nodes; this also generates a mapping $b : (E_1, E_2 \cdots E_M) \rightarrow \{e|e \in E_N\}$ of data flow to communication links. The fusion function placement algorithm is to determine mapping from fusion functions to network nodes and routing protocol of network is to determine mapping data flows to routing paths, which will not be discussed in this report.

In network, a network node can play one or more of four roles: sensor node, relay node, fusion function node and sink node. Sensor nodes and sink node are determined by application, fusion function nodes are determined by fusion function placement and relay nodes on the path between those nodes is determined by routing protocol.

B. Power Consumption in Networks

Data need to be transferred from sensor node to fusion node, from fusion nodes to next fusion node (or sink node) in distributed data fusion application. For a distributed data fusion application, the total power consumption per unit time of data transmission and data fusion after mapping to network nodes is shown as (1).

$$P_{total} = \sum_{i=1}^{N} \sum_{j=1}^{M} \Big\{ e_{radio}(j) \times \sum_{k=1}^{L} \{output_k \times hop(j, next)\} + e_{comp}(j) \times f_i \Big\} \quad (1)$$

In (1), N is the number of levels of application task graph, M is the number of fusion functions in level i and L is the number of data output(s) of fusion j. $output_k$ represents data output (in bits per unit time) of fusion function, $hop(j, next)$ is the distance (in number of hops) between node j and next fusion function (its parent), $e_{radio}(j)$ is energy consumed per bit by the radio at node j, f_j is total number of instructions of performing fusion function at node j with total input data per unit time and $e_{comp}(j)$ is the energy consumed per instruction by the processor at node j. According to our assumption that network nodes are homogeneous and to that computation cost cannot be reduced, equation (1) can be simplified as (2), which we use to estimate the total power consumption per unit time and try to minimize in mapping process.

$$P_{total} = \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{L} output_k \times hop(j, next) \quad (2)$$

C. Cost Function

Our cost function is different from those in [20][21]. In this paper, cost function of application mainly defines three parameters which are search radius in hop count for optimal node during fusion function placement process (introduced in next section), periodic update interval in fusion function transfer process (introduced in next section) and minimal remaining power level for fusion function node to host fusion function. To average the power consumption of all the network nodes and avoid being exhausted of network nodes, this minimal remaining power level can be more than one. After the remaining power levels of all the network nodes are below current minimal remaining power level, this minimal level can be changed to a lower one so as to make power of all the network nodes be consumed gradually.

III. PAFusion Framework

As we discussed in last section, the procedure of mapping application task graph can be divided into two parts, fusion function placement where each fusion function will be placed on network node with optimal energy cost and fusion function transfer where fusion function will be dynamically transferred between network nodes according to cost function defined by application.

The general problem of mapping an arbitrary application task graph to network nodes is NP-complete [22][23]. Given a specific application task graph, an optimal solution can be found deterministically. However, as reasoned in [20] optimality is not always guaranteed in minimum Steiner Tree (MST) and existing approximate solutions to Steiner Tree and graph mapping problems, which assume network topology is known and network is static, are impractical to be applied to sensor networks. So it is more efficient to design a heuristic approach for fusion function placement.

A. Fusion Function Placement

Fusion function placement phase is designed to place each fusion function on an optimized node in networks. There are two major steps in this phase. First sink node choose potential nodes to host fusion functions in networks, second these nodes will communicate with each neighbors to find an optimal node with least power consumption for hosting current fusion function. The first step called initial placement has a huge impact on total control overhead in this phase. For example, if sink node firstly chooses a potential fusion function node n hops away from an optimal node, at least n role transfers are needed before reaching the optimal node. These n role transfers will introduce lots of control overhead including current nodes broadcast transfer request and neighbors compute power consumption for hosting that fusion function.

In [20], the authors proposed top-down and bottom-up approaches for applications, all fusion functions of which are data expanding, and applications, all fusion function of which are data contraction. However, this property is application specific, all fusion functions of an application are not always all data expanding or all data contraction. So these two proposed approaches cannot map hybrid (means some fusion functions are data contraction and others are data expanding) application task graph to network nodes optimally as shown in Fig. 1 (c). The rule of thumb for application fusion functions placement is to place fusion functions with data contraction near their data sources and fusion functions with data expanding near sink node in order to reduce the transmission costs. However, application task graph is always a tree with several levels. Placing fusion functions according to this rule individually may actually increase power consumption of data transmission in other level as shown in Fig. 1 (b). So we propose to divide the application task graph into two parts-data contraction part and data expanding part according to data ratio of each level of application task graph. Suppose the application task graph is a tree with N levels and the leaf nodes are sensor data sources. Their parents are the fusion functions that are $N-1$ level distant from the root. The data ratio β_M for each level can be computed according to (3).

$$\beta_M = \begin{cases} \frac{\sum_{i=1}^{O} \alpha_i \times (\sum_{j=1}^{P} input(M)_{ij})}{(\sum_{k=1}^{Q} output(M+1)_k)/\beta_{M+1}} , & M+1 \neq N \\ \frac{\sum_{i=1}^{O} \alpha_i \times (\sum_{j=1}^{P} input(M)_{ij})}{\sum_{k=1}^{Q} output(N)_k} , & M+1 = N \end{cases} \quad (3)$$

Here, i is the number of fusion functions in level M, j is the number of data inputs of each fusion function, k is the number of data outputs of previous level and α is data ratio of each fusion function. The level with minimal data ratio among all levels is used as a partition line to divide application task graph. Then the placement of fusion functions is based on the following rules:

- If the minimal data ratio is less than 1, place all the fusion functions from N to the partition line near data sources and place other fusion functions near sink node according to the approaches introduced in the next sub-section.
- Else place all fusion functions of application task graph near sink node.

Algorithm 1: Algorithm for determination of placement of fusion functions

```
/* for partitioning and placing application task graph with N levels */
/* DataRatio is used to calculate β using (3 */)
CurrentLevel ← N − 1
RecordLevel ← 0
RecordRatio ← 1
while CurrentLevel > 0 do
    if DataRatio (CurrentLevel)< 1 then
        if DataRatio (CurrentLevel)< RecordRate then
            RecordRatio ← DataRatio (CurrentLevel)
            RecordLevel ← CurrentLevel
        end
    end
    CurrentLevel ← CurrentLevel − 1
end
if RecordLevel = 0 then    /* Did not find any data ratio less than 1 */
    PlacementNearSinkNode (N − 1 → 1)
else
    PlacementNearDataSources (N − 1 → RecordLevel)
    PlacementNearSinkNode (RecordLevel − 1 → 1)
end
```

Algorithm 1 shows the complete algorithm for determining the partition of application task graph and fusion function placement. These two approaches are used to place fusion functions as shown below:

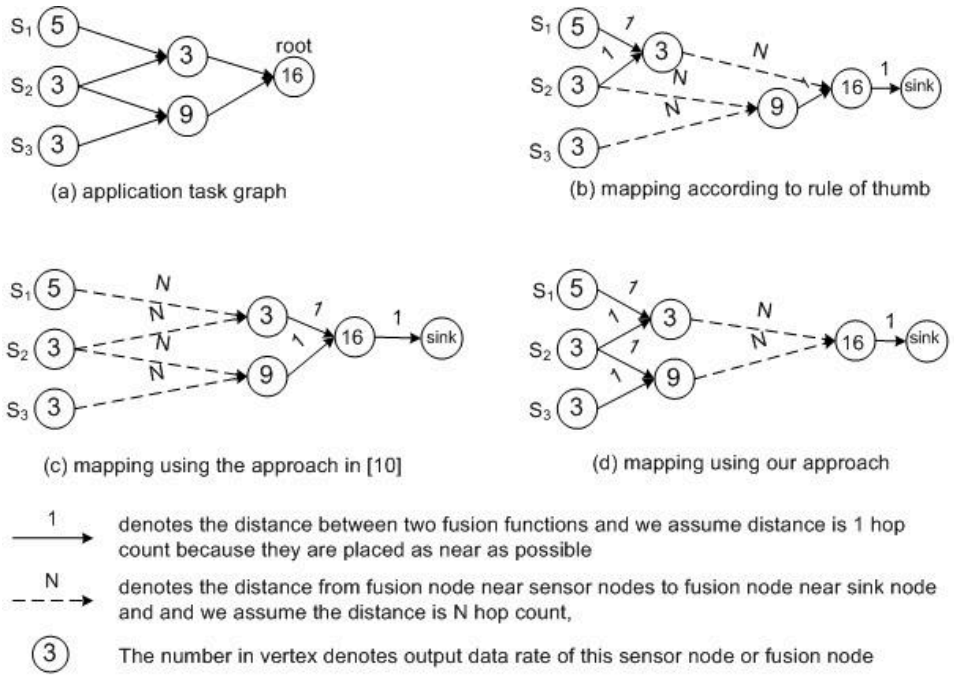

Fig. 1. Mapping of application task graph using various approaches

Approach of placement fusion function near data sources: for example fusion function θ at level M, sink node transmits placement requirement message (PEM) which includes the addresses and data rates of all the data input nodes of θ to next hops of routing paths from sink node to these data inputs nodes of θ. When next hop node receives the packet, it finds the route paths to these data input nodes. Then it computes the transmission costs that are the sum of each product of data rate and hop count to these data input nodes if it hosted this fusion function. Then the node adds its transmission cost to PEM then sends it to next hop of routing path to that data input node. Next node repeats these actions and only replaces transmission cost if its transmission cost is smaller for hosting this fusion function.

During the transmission process, if one node could not find the path to destination or other data input node due to the failure of other node in networks, it would send a failure message back to sink node. After receiving failure message, sink node will update routing tables and check out whether these data input nodes are reachable from sink node. If routing paths to these node cannot be found, sink node will inform application, otherwise it will send out PEMs again. Sink node also need to set a timeout timer after every time sending out PEMs to avoid the failures like one node failed after receiving PEM and to retransmit PEMs.

When data input nodes of fusion function θ receive this packet, they will directly send it back to sink node. Leaf nodes are not supposed to host any fusion function, because they need to sense information and send it out constantly, which costs lots of energy. If it hosted any fusion function, it would be exhausted very quickly. When sink node receives PEMs sent back by each data input nodes, sink node chooses the node associated least transmission cost for hosting this fusion function as a potential candidate. Then sink node informs chosen fusion function node and all the data input nodes of this fusion function.

When all the fusion functions are chosen in M level, sink node will recursively map fusion functions of next higher level to nodes in networks in the same way till the last one-boundary level is mapped to the networks. Because the paths to data input nodes found in route table are always the shortest paths to these destinations according to route protocol used in networks, the node on the path with least transmission cost has shorter hop distance than data input nodes themselves. Moreover, the chosen node for hosting fusion function is always on the path from data input nodes with largest data rate of that fusion function to sink node, so it is near the optimal position or it is exactly that optimal position when the data input number of fusion function increases.

Approach of placement fusion function near sink node: suppose placing fusion function θ at level M, sink node also transmits the addresses and data rates of inputs of fusion function θ to next hops of routing path from sink node to these data input nodes of θ. However, this time only nodes with M hop count to sink node will compute cost of hosting this fusion function and send it back to sink node. If M is more than hop count from sink node to data input node, the node, which is the next from data input node to sink node, will do that. Otherwise sink node chooses the one with less power consumption for hosting fusion function θ. When all the fusion functions are placed in M level, sink node will recursively map fusion functions of next higher level. For the

highest level or root fusion function, sink node can decide to host it or not according to its resource constraints. After choosing placements of all the fusion functions of application task graph, sink node will let these potential candidate nodes for hosting fusion functions figure out whether there is optimal node with less power consumption and satisfying cost function of application among all neighbors according to cost function. Current fusion function node will broadcast a message including its power remaining, power consumption of hosting this fusion function, its data input nodes addresses, and cost function. After receiving the message, network nodes will compute power consumption for hosting this fusion function and check whether it satisfies the cost function. There are two cases for finding optimal nodes:

- If current node satisfies cost function, which means current node satisfies the remaining power requirement of cost function, the nodes that satisfy cost function and have less power consumption than current fusion are required to reply the message, sink node will search the optimal node in search radius defined in cost function. Then the node with least power consumption will be chosen as final placement for that fusion function.
- If current node does not satisfy cost function, the nodes that satisfy cost function are required to reply the message. If fusion node does not get any reply for neighbors in search radius defined in cost function within the timeout interval, it will increase the search radius for finding optimal node by one more hop till it gets a suitable node to host this fusion function. If it still does not get any network node, it will change the minimal remaining power level to a lower one defined in the cost function.

Thereafter, the chosen node will claim as a fusion function node and contact all the data input nodes and parent(s) to confirm all these nodes are alive and then send sink node a ready for receiving fusion function code message, if not, a failure message including failed node ID will be sent to sink node. If sink node receives a ready for receiving message, it will transfer fusion function code to this node. If sink node receives a failure message, sink node will reinitialize a placement process for failed node. If failed node is in level $M+1$ and it is sensor node, sink node will report a sensor node failure directly to application. The process must be executed level-by-level, because positions of fusion functions of $M+1$ directly impact power consumption of fusion node of level M. After all the fusion functions are placed on network nodes, application is ready to begin. In order to reduce transmitted messages, only nodes have better health than current node will send message back to current node. Then current node eventually sends fusion function code to the node that has best health for hosting this fusion function among the nodes it received messages from and informs its data input nodes, parent nodes and sink node that fusion function has been transferred to that node. When the chosen node receives fusion code, it will claim as fusion node so that any failure during this transfer can be found. The process must be executed level-by-level, because positions of fusion functions of $M+1$ directly impact power consumption of fusion node of level M.

B. Fusion Function Transfer Maintenance

Fusion function transfer maintenance is designed to transfer fusion function between network nodes to keep fusion nodes from being exhausted according to cost function of application. During maintenance phase, function transfer can be triggered by periodic update of power consumption or by that current fusion node does not satisfy the cost function any more. If fusion function transfer triggered by periodic update, current fusion node will do exactly like in case one (introduced in last subsection) for finding optimal node to check whether there is a network node consumes less power than current fusion node for hosting this fusion function, because of the dynamic property of networks. If fusion function transfer is triggered by that current fusion node does not satisfy cost function any more, fusion node will do exactly like in case two to transfer fusion function. The only difference between fusion function transfer and finding optimal node in previous phase is that fusion function code is transferred from current fusion function node not sink node.

Besides, maintenance phase needs to deal with node failures and recovery from failures. When sink node failed, it can be directly detected by application. If relay node fails, its impact will be deal with by routing protocol, which will choose other efficient path to go through. These will not be discussed here. In the last section we discussed how to detect for failures and recover from them in fusion function placement phase. In this section, only fusion function nodes' failure and sensor nodes' failure will be discussed during data transmission process.

Fusion function node will send out an 'alive message' to its data input node when fusion node does not receive data from in a timeout interval. Let the fusion function be f and the data input node be k. If k does not reply message, f will send a failure message including ID of k to sink node. Sink node will contact k again. If k is also not reachable from sink node, it turns out k is either failed or not reachable in networks any more due to network partitioning. If k is sensor node, sink node will report a sensor node failure message node to the application; otherwise, sink will initialize a fusion function placement to replace f. If k is reachable from sink node, k will be required to contact with f to avoid false alarm. If it is not a false alarm, this indicates some other node failed and partitioned the connection between k and f. Sink node will initialize a fusion function placement to replace f. After fusion function placement process, test data will be sent from data input node to fusion function and recovery procedure is finished.

IV. Simulation Methodology

In order to evaluate the performance of PAFusion, we choose DFuse in [20] to compare with PAFusion. However because DFuse is only suitable for applications all the fusion functions of which are either all data contraction or all data

Network Parameters	
Maximum Transmission Range	50m
Network Coverage	1050m×1050m
Total Network Node	Up to 1024
Node Configuration Mode	Grid
Data Rate at Physical Layer	2Mbps
Application Task Graph Specification	
Data Input Number of One Fusion Function	Up to 6
Application Task Graph Size (the number of fusion functions of application)	Up to 32
Application Task Graph Connectivity	Tree based

TABLE I
BASIC PARAMETER CONFIGURATION FOR SIMULATION

expanding, on the contrary PAFusion does not have such limitation, in simulation we assume all fusion functions of application task graph are data contraction and all network nodes satisfy cost function before mapping application task graph so as to compare PAFusion with DFuse. We simulated our PAFusion and placement module of DFuse in QualNet a discrete event simulator for network simulation. We assume all the network nodes are stationary. The parameter values used in the simulation are presented in I.

There are four performance metrics evaluated in simulation:

1) Distance to optimal node after initial placement: In both PAFusion and DFuse, fusion functions are initially placed on selected network nodes, and then these nodes will find optimal nodes for hosting these fusion functions. Because distance from initial placement to optimal location not only has huge impact on the quality of steady-state behavior of application [20], distance to optimal node can be a benchmark for evaluating the performance of placement algorithms
2) Delay of fusion function placement: The total delay for mapping distributed data fusion application task graph into networks nodes includes the time for finding optimal positions for all the fusion functions in application task graph and the time spent on fusion function code transfers. However, the latter one heavily depends on specific application. So, in order to evaluate the time delay caused by PAFusion, in simulation we only consider the time delay spending on finding optimal nodes for hosting fusion functions.
3) Control overhead: Control overhead shows total messages transmitted during fusion function placement process, which gives a view of network traffic and power consumption introduced by fusion function placement procedure. In this paper, we define a PEM transmitted in one hop as one message of control overhead.
4) Scalability: We simulate PAFusion framework on the networks of more than one thousand nodes and for different sizes of application task graph to evaluate whether performance metrics of PAFusion scale well with large networks and application task graph.

V. PERFORMANCE EVALUATION

In this section, we evaluate PAFusion and compare PAFusion and DFuse with respect to the above performance metrics by simulation. Although these performance metrics are affected by many parameters, only a set of important parameters is chosen such as typical data input number of one fusion function, network size and application task graph size.

A. Evaluation of Distance to Optimal Node

In simulation, the sink node randomly chooses data input nodes with random data rate for each fusion function from network nodes, and then places each fusion function on network node. Fig. 2 shows the average distance (in hop count) to optimal node after initial placement of PAFusion according to different data input number and network size (including 9×9, 16×16, 2×20, 25×25 and 32×32 grid networks). Fig. 2 show the average distance to optimal node is relatively steady in different sizes of networks and it is decreased according to the increase in the data input numbers. It is because of more data inputs more network nodes on the paths from sink node to data input nodes, which compute their transmission cost for hosting fusion function after sink node transmits PEM to each data input node. It is more accurate for sink node to choose fusion function node by getting more information about hosting cost from more network nodes.

Fig. 3 gives the comparison of average distance to optimal node after initial placement in PAFusion and DFuse. In DFuse, according to [20] sink node just lets an appropriate data input node choose the node with shorter distance to sink node among their neighbors as initial fusion function node. This approach that does not consider data rate of each data input will cause giant gap between initial placement and optimal node, so in the simulation of DFuse, we let data input node with highest data rate among all the data input nodes choose initial fusion function. Although this modification can reduce the distance between initial placement and optimal node, as shown in Fig. 3 distance using DFuse increases quickly with the increase of data input number. Because choosing a neighbor with shorter distance to sink node becomes more and more inefficient when data input number increases. When data input number increases, optimal node is always further away from any of its data input node. When network size grows, this problem will become more severe as shown in Fig. 3. It causes more transfers from initial placement to optimal node, which will lead to more control overhead, power consumption and delay of application task graph.

B. Evaluation of Delay for Fusion Function Placement

According to our approach for fusion function placement in PAFusion, placements of all the fusion functions of one level of an application task graph can be initialized simultaneously

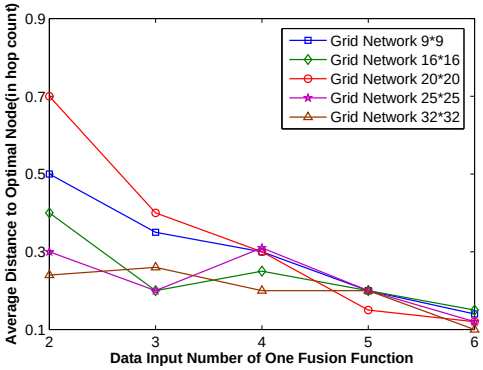

Fig. 2. Average distance (hop count) to optimal position after initial placement in PAFusion

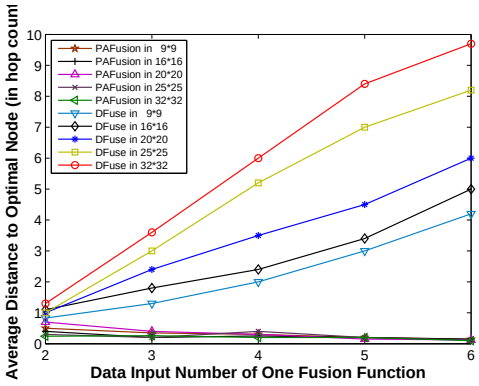

Fig. 3. Comparison of average distance to optimal node after initial placement in PAFusion and DFuse

and application task graph will be mapped to networks level by level. After all the potential nodes for hosting fusion functions in each level of application task graph are chosen, potential nodes in each level will simultaneously try to find optimal node and this process is also executed level by level. As discussed in last subsection, after initial placement the average distance between potential node and optimal node is always less than 1 hop count under various network sizes and data input numbers. So the delay for finding all the optimal nodes for application task graph is shown in (4).

$$D = \sum_{i=1}^{N} \left(\max_{1 \le j \le M} (d_{ij}) + d_t \right) \tag{4}$$

In (4), N is the number of levels of application task graph, M is the number of fusion functions in level i, d_{ij} is the delay for finding potential node and d_t is the time spending on find optimal node in one hop. In order to get rid of specific application for simulation results, we use the delay of placing one fusion function $d_{ij} + a \times d_t$ to represent the delay in simulation, a is the distance from potential fusion function node to optimal node after initial placement. Although the delay is related with a lot of parameters such as physical model and MAC protocol used in wireless networks, in simulation only data input number of one fusion function, network size are considered.

Fig. 4 shows the delay for placing one fusion function node has linear scale with data input number in different network sizes. As shown in Fig. 4, the delay increases with the increase in the data input number, because more data inputs of one fusion function cause more time for the sink node to communicate with these data input nodes. Fig. 5 shows the change of the delay according to the change of network size with different data input number. As shown in Fig. 5 , the delay also has a linear relation with network size. The delay increases with the increase of network size, because large network size causes longer path from the sink node to data input node. Fig. 6 gives the delay comparison between DFuse and PAFusion. The delay of DFuse is always longer than PAFusion and the delay ratio between DFuse and PAFusion increases rapidly with the increase of data input number.

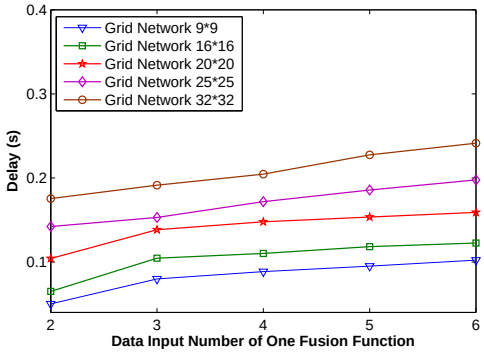

Fig. 4. Delay for placing one fusion function with different data input number with PAFusion

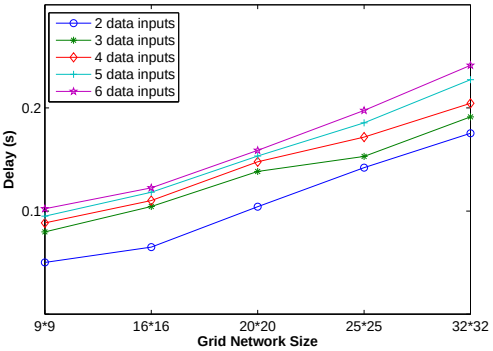

Fig. 5. Delay for placing one fusion function with different network sizes with PAFusion

C. Evaluation of Control Overhead for Fusion Function Placement and Scalability

Power consumption of fusion function placement can be divided into two parts, one is the power consumption introduced by fusion function placement algorithm and another is the power consumption of transferring fusion function codes to selected fusion nodes. The latter one is heavily related to

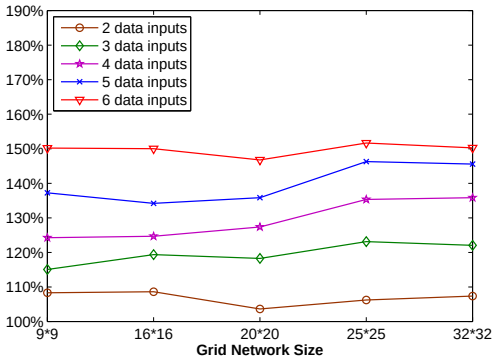

Fig. 6. The delay of DFuse compared with PAFusion

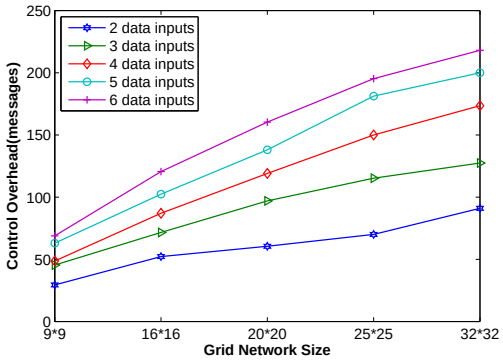

Fig. 7. Effect of network size and data input number upon control overhead of fusion function placement in PAFusion

specific application, so in this paper we only consider the former one. As we discussed in last section, we define a PEM transmitted in one hop as one message of control overhead. So the power consumption introduced by placement algorithm is the sum of power consumption of transmitting PEM and computation. However, computation during placement process is negligible compared with transmitting cost, so the total power consumption can be denoted by (5).

$$p = n_t \times d_{PEM} \times e_{radio} \tag{5}$$

Here n_t is total number of control overhead during placement process, d_{PEM} is data size of PEM and constant. So the total control overhead can be directly used to estimate the power consumption during placement process.

Fig. 7 shows the control overhead for placing one fusion function with different data input number in different network size. Control overhead increases linearly with the increase of network nodes and data input number. The comparison of control overhead for placing one fusion function with 2, 4 and 6 data inputs between PAFusion and DFuse is shown in Fig. 8. Compare to PAFusion, DFuse leads to more control overhead for fusion function placement than PAFusion. Moreover, this difference increases when in larger networks or when fusion function has more data inputs. In order to evaluate effect of application task graph size upon control overhead both in PAFusion and DFuse, we assume all fusion functions of application task graph have 6 data inputs. As shown in Fig. 9, control overhead of PAFusion and DFuse both increases linearly with the increase of application task graph size and PAFusion has lower overhead messages.

VI. Future Work

In the paper, PAFusion determines the mapping of application task graph to network nodes power efficiently and maintains fusion function transfer to average power consumption according to cost function and routing protocol determines data flow path between application task graph nodes. Because not only data flow paths chosen by routing protocol are directly related to data transmission cost in the entire networks but also PAFusion uses routing information to determine the placement

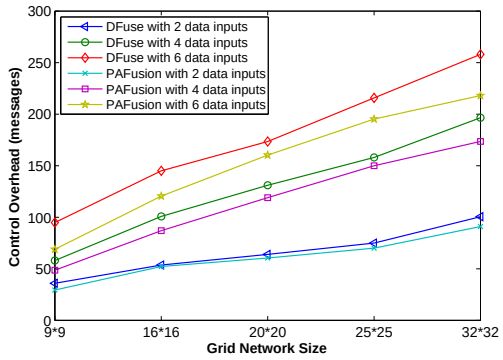

Fig. 8. Comparison of control overhead between PAFusion and DFuse

of each fusion function, routing protocol is very important to PAFusion. In future we plan to design a power and delay aware routing protocol integrated with PAFusion, which can choose different routing path according to application requirements of delay and power consumption. The integration of PAFusion and power and delay aware routing protocol can not only optimize power consumption and extend network longevity further, but also support data fusion application with latency requirement. For example, within the latency requirement

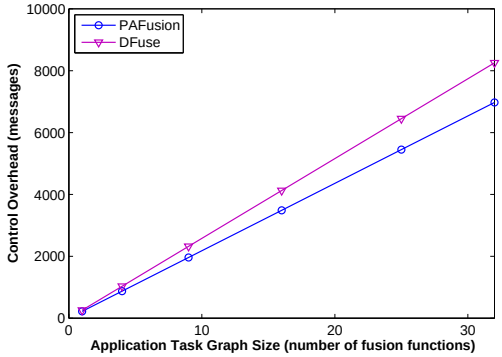

Fig. 9. Effect of application task graph upon control overhead both in PAFusion and DFuse (in 32×32 grid networks)

of application routing protocol may choose a longer path to destination to avoid using nodes in short path but with little power remaining. We also plan to extend PAFusion to support mobile sink node, because in some cases, sink node is not stationary, for example a mobile car can be used as a sink node in forest to monitor environment dynamically. So mobility is another important criterion for PAFusion to support all kinds of data fusion applications. One possible solution is to let the sink node dynamically send out queries to interested sensor nodes, choose and place fusion functions on network nodes during transmission process. Each query has certain valid time interval, after that query expires and all fusion functions are canceled. Another possible solution is to let fusion functions be transferred dynamically according to updated routing path from sensor nodes to the sink node. However, the most promising approach is to place all fusion functions of application task graph near sensor nodes. So the results of data fusion application can be transmitted to sink node directly just like sensed data and there is no need to consider the effect of placement of fusion functions upon delay and power consumption.

VII. Conclusion

In the paper, PAFusion a power-aware framework is proposed to map distributed data fusion application into WSNs for Active Networks computing paradigm and minimize power consumption of application and to dynamically transfer fusion functions between network nodes according to cost function of application. Nodes hosting fusion functions can avoid running out of power by the transfer of fusion function between network nodes. We simulated PAFusion with QualNet in the scenarios with different network size and different data input number of fusion function and compared the simulation results with DFuse, although DFuse has the limitation that it can only map the application task graph with most fusion functions are either data contraction or data expanding. The simulation results show that not only PAFusion can map all kinds of applications, but also it has better placement properties in distance to optimal node after initial placement, less delay for mapping application task graph and less control overhead than DFuse.

References

[1] H. Durrant-Whyte, "Sensor models and multisensor integration," *The International Journal of Robotics Research*, vol. 7, no. 6, p. 97, 1988.

[2] R. Luo, C. Yih, and K. Su, "Multisensor fusion and integration: approaches, applications, and future research directions," *IEEE Sensors Journal*, vol. 2, no. 2, pp. 107–119, 2002.

[3] E. Nakamura, A. Loureiro, and A. Frery, "Information fusion for wireless sensor networks: Methods, models, and classifications," *ACM Computing Surveys (CSUR)*, vol. 39, no. 3, p. 9, 2007.

[4] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 56–67, ACM, 2000.

[5] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pp. 457–458, IEEE, 2002.

[6] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 146–159, 2001.

[7] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 1, pp. 2–16, 2003.

[8] H. Qi, Y. Xu, and X. Wang, "Mobile-agent-based collaborative signal and information processing in sensor networks," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1172–1183, 2003.

[9] Y. Xu and H. Qi, "Distributed computing paradigms for collaborative signal and information processing in sensor networks," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 945–959, 2004.

[10] M. Chen, T. Kwon, Y. Yuan, and V. Leung, "Mobile agent based wireless sensor networks," *Journal of Computers*, vol. 1, no. 1, pp. 14–21, 2006.

[11] M. Chen, T. Kwon, Y. Yuan, Y. Choi, and V. Leung, "Mobile agent-based directed diffusion in wireless sensor networks," *EURASIP Journal on Applied Signal Processing*, vol. 2007, no. 1, p. 219, 2007.

[12] K. Psounis, "Active networks: Applications, security, safety, and architectures," *Communications Surveys & Tutorials, IEEE*, vol. 2, no. 1, pp. 2–16, 2009.

[13] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran, "DFuse: a framework for distributed data fusion," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 114–125, ACM, 2003.

[14] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," *ACM SIGARCH Computer Architecture News*, vol. 30, no. 5, pp. 85–95, 2002.

[15] A. Boulis, C. Han, and M. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, pp. 187–200, ACM, 2003.

[16] L. Yu, N. Wang, and X. Meng, "Real-time forest fire detection with wireless sensor networks," in *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, vol. 2, pp. 1214–1217, IEEE, 2005.

[17] W. Su and T. Bougiouklis, "Data fusion algorithms in cluster-based wireless sensor networks using fuzzy logic theory," in *Proceedings of the 11th Conference on 11th WSEAS International Conference on Communications-Volume 11*, pp. 291–299, World Scientific and Engineering Academy and Society (WSEAS), 2007.

[18] W. Sung, "Multi-sensors data fusion system for wireless sensors networks of factory monitoring via BPN technology," *Expert Systems with Applications*, vol. 37, no. 3, pp. 2124–2131, 2010.

[19] P. Manjunatha, A. Verma, and A. Srividya, "Multi-sensor data fusion in cluster based wireless sensor networks using fuzzy logic method," in *Industrial and Information Systems, 2008. ICIIS 2008. IEEE Region 10 and the Third international Conference on*, pp. 1–6, IEEE, 2009.

[20] U. Ramachandran, R. Kumar, M. Wolenetz, B. Cooper, B. Agarwalla, J. Shin, P. Hutto, and A. Paul, "Dynamic data fusion for future sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 3, pp. 404–443, 2006.

[21] J. Chang and L. Tassiulas, "Energy conserving routing in wireless ad-hoc networks," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 22–31, IEEE, 2002.

[22] M. Garey and D. Johnson, "A Guide to the Theory of NP-Completeness," *Computers and Intractability*, vol. 3, no. 5, pp. 23–26, 1979.

[23] C. Papadimitriou and M. Yannakakis, "Towards an architecture-independent analysis of parallel algorithms," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 510–513, ACM, 1988.