Packet Merging-driven Tree Construction in Wireless Sensor Networks

Aleksandar Ilić
IHP
Frankfurt (Oder), Germany
ilic@ihp-microelectronics.com

Mario Schölzel
IHP
Frankfurt (Oder), Germany
schoelzel@ihp-microelectronics.com

***Abstract*—Performance of Medium Access Control (MAC) protocols used in convergecast sensor networks such as TreeMAC depends on topology of the data gathering tree used for routing data packets. We show that data rate can be increased if this topology is optimized during the tree construction phase. The algorithm that performs tree construction and optimization in parallel is proposed and then compared with two similar state of the art protocols.**

***Index Terms*—wireless sensor networks, convergecast, data collection tree, minimum path tree, bipartite graph maximum matching**

I. Introduction

One of the most common data collection scenarios in wireless sensor networks (WSN) is convergecast. It is collection of data from many sensors to one node which is called the sink. Depending on network application, the optimization goal for the data collection protocol in such networks is either minimization of the energy consumption, or maximization of the data rate. Data rate depends the most one the MAC (Medium Access Control) protocol. In the case of a sensor network in which traffic is generated by all nodes in the network and the data generation frequency is high, best results are obtained when a TDMA (Time Division Multiple Access) protocol is used. These protocols avoid collisions by allocating conflicted nodes in separate time slots. There are many algorithms that can optimize this allocation in terms of the data rate. However, before a schedule can be calculated, the network must be discovered, and the gathering tree formed, by allocating children groups to parent nodes. Apart from the TDMA schedule, data rate of such a network is also dependent from the data gathering tree structure. This paper proposes tree formation algorithm that helps increase the data rate by allowing the nodes to merge packets and transmit them together in one time slot.

In many-to-many networks, the optimal TDMA schedule is the shortest one i.e. the one that allows to all the nodes to transmit in the lowest number of slots while preventing conflicted nodes from transmiting at the same time. This schedule can be calculated using graph coloring algorithms. Such a schedule will be optimal only when all nodes have equal load (equal channel demand). However this is not the case in a convergecast network. In convergecast networks, nodes which are the closest to the sink have the highest load (since they need to pass all the data from their children) and therefore require more time slots than the nodes at higher levels. The problem of finding the optimal schedule in this case is shown to be NP-complete by Choi *et al.* [1]. Most efficient protocols divide the gathering three into top subtrees and then schedule the individual subtrees. A top subtree is a subtree rooted at one of the sink's children. One such protocol is TreeMAC, it achieves a schedule length equal to $3N$ where N is the number of nodes in the network excluding the sink [2]. Similar schedule length is reported in [3], while Grandham *et al.* [4] reports the same schedule length in general case, with the possible reduction when top subtrees are independent (can transmit at the same time without conflicts). This result has a theoretical value, but it will rarely be applicable in a real world scenario. Since most of the protocols for TDMA scheduling in data gathering trees are similar and produce the schedule of similar length, the proposed optimization is going to be demonstrated using the example of TreeMAC. It can however be applied to other similar protocols without modifications.

Packet merging is a technique used to increase network throughput by merging multiple smaller packets that have the same destination address [5]. It is possible to merge packets and still transmit them in one time slot because due to the technical reasons the time slot size will often be longer than the transmission time of an average packet. This technique is especially useful in convergecast networks because most of the packets have the same destination address, the sink. When applied to TreeMAC, this technique can reduce the schedule up to three times, depending on the tree topology. TreeMAC operates on shortest path trees and uses two-hoop interference model to schedule parallel transmissions. To allocate slots proportionally to the workload, it divides time into frames (one frame equals three time slots). The sink will be active all the time, while different branches of the tree are allocated in different time frames. If a branch contains n nodes, in general case $3n$ slots are required for scheduling this branch, and the sink receives one packet in each frame. When packet merging is used, the sink can receive up to three packets in one frame, providing that the topology allowed for the three packets to be merged.

The schedule formation and the topology dependence is going to be demonstrated using the two data gathering trees constructed on the same network shown in Fig. 1(a). In the

figure, circles represent sensor nodes, and their IDs are written inside the circles. A solid line represents the connection between a parent and its child, while a dashed line connects a parent with its possible child. It is assumed that all packets have length that allows for three packets to be merged and transmitted together in one time slot. The generated time schedule for the unoptimized network is shown in Fig. 1(b). In the first frame TreeMAC schedules nodes 1 and 7 to transmit in parallel in slot 3 of the frame. During this frame, packets from nodes 5, 3 and 1 are merged and transmitted to the sink. Meanwhile, the packet from node 7 is forwarded to node 5 and will reach the sink in the next frame. The rest of the schedule is formed in the similar manner. Note that in the third frame, only the packet from node 6 reaches the sink, because all nodes on the way to the sink have already transmitted a packet. If node 6 was assigned as a child to node 4 its packet would have been merged with those of nodes 2 and 4 and delivered to the sink together, reducing the schedule from 5 to 4 frames as shown in the Fig. 1(c). Therefore we deduce that in order to maximize packet merging, the number of leaf nodes the with height (distance from the sink) smaller than the maximal height of the tree needs to be minimized.

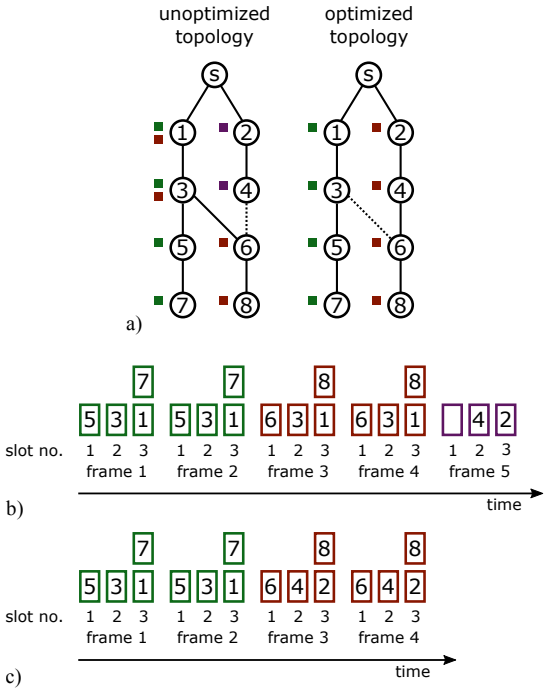

Fig. 1. TreeMAC schedule example

Section II presents an algorithm that performs optimization according to those constraints. The main difference between the proposed algorithm and the existing topology control techniques is in the definition of the optimal topology. The existing work focuses predominantly on load-balanced shortest path trees. Thereby was load-balanced defined in different ways. Hsiao *et al.* introduces the definitions of a fully load-balanced tree and a top load-balanced tree [6]. To evaluate the result, weight of a tree is introduced as the total number of nodes in the tree. Top load-balanced tree is a tree in which the weight of each top subtree is equal, while the second one is the one in which all the branches have the same weight. Another definition, given by Andreou *et al.* defines a load-balanced tree as a tree in which the branching factor of each node does not excede a certain value [7]. Load-Balanced trees were so far used to achieve different optimizations, such as increasing network lifetime [8], reducing number of collisions [7], or increasing network throughput [9]. Another approach is network optimization based on link quality [10]. However this was so far done for protocols that do not utilize parallel scheduling. Since networks with parallel scheduling are our focus, and implementation of such methods in these networks is connected with many difficulties, we have chosen to deal with link quality issue by simply avoiding usage of bad links when possible. To achieve this link quality is ascertained in the tree construction phase, whenever a new node is discovered. Another similar topic in topology control is minimum energy routing [11]. Those algorithms are however not applicable here because they would route the packets between nodes on the same level, which would introduce the need for a different parallel scheduling algorithm. In Section III we show that for the intended application, the proposed method performs better than the existing ones.

II. The Proposed Tree Construction Protocol

The goal of the algorithm is to construct shortest path spanning tree while minimizing the number of leaf nodes on the levels lower than the tree height. Since the tree must be a shortest path tree, children parent assignment is done between the nodes on two consecutive levels (level is the set of nodes at the same distance from the sink). The lower level is called the parent level, and the upper the children level. Nodes from the first level are called parents and nodes from the second children. A node on the parent level that does not get a child assigned becomes leaf node. Therefore, the optimal assignment is the one that minimizes the number of nodes on the parent level without children. Fig. 2 illustrates that this problem is not trivial even when the number of nodes on the both levels is the same. In the figure, if node $C2$ is assigned to node $P1$, node $P3$ will be left without children, even though it has two possible children Fig. 2 (b).

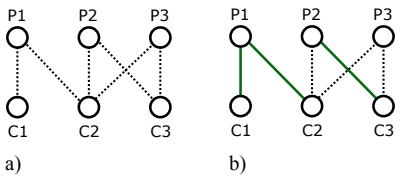

Fig. 2. Network example that demonstrates the assignement problem

A graph comprised of a parent level, a children level and the possible connections between the two levels is a bipartite graph (Fig. 2). The assignment problem can be divided into two steps. The first step is connecting maximum number of

children parent pairs in such a way that each parent can have maximum of one child, and each child maximum of one parent. If there are more children than parent nodes, some of the children will remain unconnected. Such children will have the possibility to connect only with parents which already have a child assigned. Thus, their assignment will have no effect on the number of nodes without children on the parent level, and they can be assigned to a parent in the second step using either random or link-quality based criteria. The first step represents problem of finding a maximum matching of a bipartite graph. Therefore in order to calculate children allocation, we first find the maximum matching using Ford-Fulkerson algorithm. Then we simply connect the remaining unconnected children to a possible parent with the best link quality or the smallest ID in case that link quality was not ascertained.

The complete network discovery and tree construction protocol is coordinated by the sink. Since parent assignment is calculated based on two tree levels and connectivity between them, the discovery and assignment is performed level by level. Algorithm executed by the sink to preform tree construction is shown in Fig. 3. In line 2 the sink discovers the first level. This is done by sending a *Query* message with the MAC address set to broadcast. Nodes that are not assigned yet will respond to this message and be discovered. After a node was discovered, a link quality estimation can be performed using an adequate method. If the link quality is not satisfactory the node will be removed from the list of discovered nodes. In line 6 nodes on the first level are assigned to the sink. This is done by sending a message to each of the nodes. Then the second level is discovered using the function *DiscNext()* in line 7. This function takes the list of nodes on the last assigned level as an argument, and instructs every node in this list to perform discovery of the next level. The function collects the responses, forms a list of nodes on the next level and their connectivity with the previous level, and returns this list. In line 9 the parent assignment for this newly discovered level is calculated using the previously described algorithm. Nodes on this level are then assigned to parents and subsequently asked to perform discovery of the next level (lines 10-12). This is repeated until no nodes are discovered on a certain level (while loop starting at line 8). Because in i^{th} step, the nodes at the height i are discovered and then assigned to a node on the previous level, it is obvious that the formed tree will be a shortest path tree.

```
1:  procedure ConstructTree
2:      parentLevel ← DiscLevel1()
3:      if parentLevel.count = 0 then
4:          return
5:      end if
6:      AssignLevel1(parentLevel)
7:      childrenLevel ← DiscNext(parentLevel)
8:      while childrenLevel.count > 0 do
9:          asg ← Assing(parentLevel, childrenLevel)
10:         Assign(childrenLevel, asg)
11:         parentLevel ← childrenLevel
12:         childrenLevel ← DiscNext(parentLevel)
13:     end while
14: end procedure
```

Fig. 3. Tree construction algorithm

III. Evaluation

To evaluate the proposed algorithm, we are going to compare it with the state of the art top-load balancing algorithm proposed by Incel *et al.* [12] and the balancing algorithm proposed by Andreou *et al.* [7], called ETC (Energy Driven Tree Construction). The first one utilizes search sets to determine the optimal parent for a child. A search set identifies nodes on up to two levels above the child that will be left with only one possible branch to connect in the future if the child is allocated to a certain parent. A child chooses the parent for which the sum of its subtree weight plus the search set size is minimal. ETC algorithm defines maximal branching factor as $\sqrt[h]{N}$, where h is the tree height and N is the number of nodes in the network. If this factor is exceeded at a certain node, it will ask some of its children to try to find another parent. This will not always be successful, and therefore the tree produced is called near-balanced tree.

The comparison is performed on the example network that is shown in Fig. 4(a). We are going to demonstrate the drawbacks of the existing methods using this network and show how the proposed method overcomes them. The more extensive comparison is going to be presented at the presentation in the form of simulation results. The data gathering tree produced by the top load balancing algorithm is shown in Fig. 4(b). We can see that in this case, no children will be assigned to node 2, even though node 4 was available as its possible child. This happened because when child 4 was assigned to a parent, both possible parents, node 1 and node 2, were in branches with the same weight (equal to one). Second criteria used, search set, was also the same for both parents. Both search sets in this case contained only node 7, because this is the only node that will have to join the same branch as node 4 (node 8 will be able to choose between subtrees rotted at node 1 and at node 3). Therefore the tie was broken on the smaller index, and node 1 became the parent of node 4. We also observe that for node 5, the algorithm worked as expected. In that case, the search set for node 1 contains nodes 8 and 9, and for node 3 it contains only node 9. Therefore node 5 was assigned to node 3, allowing for the nodes 8 and 9 to be shared between two subtrees in the final step.

The network constructed by the ETC algorithm is shown in Fig. 4(c). Since the First-Heard-From principle is used to create the tree and node 1 was the first to perform assignment, both node 4 and node 5 were assigned to it, leaving node 2 without children. Because branching factor for this network is $\beta = \sqrt[3]{10} = 2.15$, node 1 will not try to re-assign any of its children. Finally, Fig. 3(d) shows the network obtained using the proposed algorithm. As it can be seen, the maximum matching for the bipartite graph constructed from levels 1 and

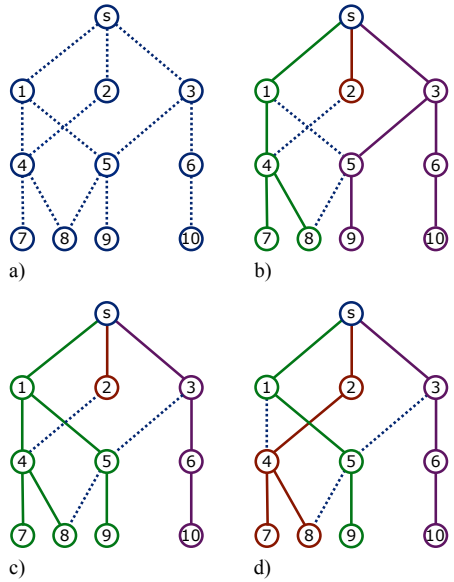

Fig. 4. Comparison of different algortihms (a) the network, (b) top load balancing, (c) ETC, (d) the proposed algorithm

2 contains three edges. The algorithm has found this matching and used it to perform child assignment. The constructed tree is optimal for packet merging and allows scheduling the network in 4 time frames using TreeMAC. For scheduling the other two networks, 5 time frames would be required. We also note that the resulting tree in this case is top load-balanced since the maximum difference between weights of the subtrees is equal to one, even though this was not the intention of the algorithm. This leads to a conclusion that the algorithm could be modified to construct top load-balanced trees, which is left as a topic for future work.

IV. Conclusion

In this paper, the data gathering tree construction protocol for creating shortest path trees is proposed. The protocol is aimed to be used with TreeMAC or similar TDMA MAC protocols that utilize parallel node scheduling and packet merging to increase throughput. We show that in these protocols schedule reduction using packet merging is dependent on the network topology. We then introduce new criterion for network optimization, the number of leaf nodes. The tree construction algorithm that constructs a shortest path tree is then introduced. To optimize the tree topology, the parent assignment needs to be calculated to qualify the defined criterion. The parent assignment algorithm that finds the optimal assignment in polynomial time if it exists is therefore proposed. The proposed algorithm is compared with two state of the art algorithms. The comparison shows that in the considered application scenario, the proposed algorithm performs better.

References

[1] H. Choi, J. Wang and E. Hughes, “Scheduling for information gathering on sensor network,” Wireless Netw., 2007.
[2] W. Song, H. Renjie, B. Shirazi, R. LaHusen, “TreeMAC: Localized TDMA MAC protocol for real-time high-data-rate sensor networks,” Pervasive Mob. Comput., vol. 5, nr. 6, 2009.
[3] S. Ergen, P. Varaiya, “TDMA scheduling algorithms for wireless sensor networks,” Wireless Netw., vol. 16, pp. 985-997, 2010.
[4] S. Gandham, Y. Zhang and Q. Huang, “Distributed time-optimal scheduling for convergecast in wireless sensor networks,” Computer Networks, vol. 52, nr. 3, pp. 610-629, 2008.
[5] V. Akila, T. Sheela and G. A. Macriga, “Efficient Packet Scheduling Technique for Data Merging in Wireless Sensor Networks,” Chinacom, vol. 14, nr. 4, pp. 35-46, April 2017
[6] P. Hsiao, A. Hwang, H. T. Kung, and D. Vlah, “Load balanced routing for wireless access networks,” Proc. IEEE Infocom, pp. 986-995, 2001
[7] P. Andreou, A. Pamboris, D. Zeinalipour-Yazti, P.K. Crysanthis and G. Samaras “ETC: Energy-drive Tree Scheduling in Wireless sensor networks, Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, 2009.
[8] J. He, S. Ji, Y. Pan and Y. Li,“Constructing load-balanced data aggregation trees in probabilistic wireless sensor neworks,” IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 7, pp. 1681-1690, July 2014.
[9] H. Zhang, F. Österlind, P. Soldati, T. Voigt and M. Johansson,“Rapid convergecast on comodity hardware,” SECON, 2010.
[10] W.B. Pöttner, H. Seidel, J. Brown, U. Roedig and L. Wolf, “Constructing schedules for time-critical data delivery in wireless sensor networks,” ACM Transactions on Sensor Networks (TOSN), vol. 10, no. 44, April 2014.
[11] P. Santi, “Topology control in wireless ad hoc and sensor networks,” Wiley, 2005.
[12] Ö. D. Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapaudi, “Fast data collection in tree-based wireless sensor networks,” IEEE Trans. Mobile Comput., vol. 11, nr. 1 pp. 86-91, January 2012.