Pushing the Frontiers of Cross-layer Optimization in Wireless Sensor Networks Right up to the Application Layer

S. Chatterjea and P.J.M. Havinga
Pervasive Systems Group, Department of Computer Science, University of Twente
P.O. Box 217, 7500AE, Enschede, The Netherlands, (supriyo, havinga)@cs.utwente.nl

Abstract

Unlike conventional computer networks such as office LANs or the Internet which can be used for a host of applications, wireless sensor networks are typically used for specific applications in mind. It is this fundamental difference combined with the fact that energy efficiency is of paramount importance in wireless sensor networks that a different approach needs to be taken when designing protocols. In this paper we first describe the main differences and benefits of the cross-layered techniques used for sensor networks over the traditional OSI layered approach used for conventional networks. We then give two examples which illustrate how we extend usual sensor network cross-layered optimization by going beyond just MAC-routing optimizations to optimizations between the application and the MAC layers.

1. INTRODUCTION

The Open Systems Interconnection Model or OSI Model has long been used as a basis to develop protocols and services designed to run on various types of computer networks such the Internet or LANs. One of the main advantages of the seven-layer OSI model is that it allows interoperability between different products and services made by different manufacturers. Thus every individual layer acts as a black box hiding the details of how it deals with incoming data from the other layers. Only the interface between the layers is made public to allow for compatible communication between neighboring layers. So as an example, a person designing a web browser need not worry whether the network interface of the computer running the web browser application is a wired or wireless connection. Another important aspect about conventional computer networks is that they are used for a whole range of applications. For example, a person may use the same network for emailing, transferring files from a network drive or even downloading streaming video from a video sharing web site. It is this general purpose nature of the network and support for interoperability that makes it essential to have a layered architecture - e.g. different protocols such as FTP or HTTP will run regardless of a wired or wireless network interface.

Wireless sensor networks (WSNs) however, have a few fundamental differences when compared to the conventional computer networks mentioned above. Rather than interoperability, the main emphasis in WSNs is energy-efficiency. Another fundamental difference is that WSNs are typically application specific networks. Thus the general approach in WSN design is to maximize energy efficiency by taking advantage of the application specific nature of WSNs.

While having a cross-layered architecture for WSNs does have a multitude of advantages, it has several drawbacks both from business and technical perspectives. As WSNs are application-specific networks, one would be inclined to conclude that every sensor network application would required a completely customized solution. This of course is economically impractical as it would entail designing a new architecture for every application. Thus it is essential to design a cross-layered architecture that is able to adapt its operation to a certain extent based on the variations that take place in the environment where the WSN is deployed. This would allow a single architecture to work optimally in a single *class* of applications rather than just a single application thus making it more economically viable. Furthermore, a cross-layered architecture has many interdependencies between different layers and this can make the higher layers particularly vulnerable to perturbations in the lower layers. Thus steps need to be taken to minimize such undesirable effects.

The main focus of this paper is to describe how cross-layer optimization - specifically with the involvement of the application layer - can benefit the operation of a WSN. We illustrate this using two examples based on an adaptive MAC protocol and scheduling for data aggregation. We first give a brief summary of related work in the next section. This is followed by an overview of the MAC protocol that our work is based on. We then describe two separate examples showing how both the MAC layer and application layer can benefit from exchanging information.

2. RELATED WORK

Cross-layer optimization is a key term that is used frequently in sensor network research terminology. However, the term does not have a well-defined definition. Generally, as stated

in [1], cross-layer optimization techniques may be categorized as follows: (i) Physical+MAC , (ii) MAC+Routing, (iii) PHY+Routing and (iv)PHY+MAC+Routing. It is immediately obvious from this categorization that the application layer is usually excluded from the realm of cross-layer optimization. We intend to further broaden the definition of cross-layer optimization by illustrating the benefits that can be obtained when optimizations are carried out between the application and MAC layers. For example, in the two examples described in this paper, we show how the MAC protocol could benefit by using information from the application layer and also how the application layer could benefit from the information provided by the underlying MAC layer.

As both our examples deal extensively with the MAC layer, we first give an overview of the MAC protocol we base our work on and explain the advantages of using it.

3. LMAC: A LIGHTWEIGHT MEDIUM ACCESS CONTROL PROTOCOL

LMAC [2] is a TDMA-based medium access control protocol designed for WSNs. Time in LMAC is divided into frames, each of which is further divided into a fixed number of time slots. Every node chooses its own slot using a distributed algorithm that uses only locally available information. A node is allowed to pick any slot as long as it is not owned by any other node within its two-hop neighborhood.

A time slot consists of two sections, the Control Message (CM) and the Data Message (DM). The CM, which contains control information and has a fixed length, is broadcast by a node to its neighbors during its own time slot once *every* frame irrespective of whether the node has any data to send. The CM contains a table which indicates the slots that are occupied by itself and its one-hop neighbors and other control information. Every node maintains a *Neighbor Table* that stores the information about its one-hop neighbors, e.g. ID, occupied slot, number of hops to sink node, etc. Thus a node can automatically work out its degree from its Neighbor Table. Occupied slots are marked with a 1 where as unoccupied ones are marked with a 0. A node joining the network first listens out for the CMs of all its neighbors and then picks one of the slots that is marked as unoccupied by performing an OR-operation. The DM contains higher layer protocol messages. The length of the DM can vary depending on the amount of data that a node needs to send. It does however, have a maximum length.

The reason for choosing LMAC is that it provides a lot of ”free” information that can have a wide range of uses. For example, all nodes running LMAC have *continuous* information about their first order neighbors. Thus any changes in network topology can be immediately detected without any additional message transmissions. LMAC also has an inherent synchronization system that not only ensures its own reliable operation but also ensures that other protocols running above it do not suffer from timing related problems. Additionally, LMAC's scheduling scheme can be re-used by other higher layer scheduling protocols in order to save energy. We illustrate how this may be done in the second example.

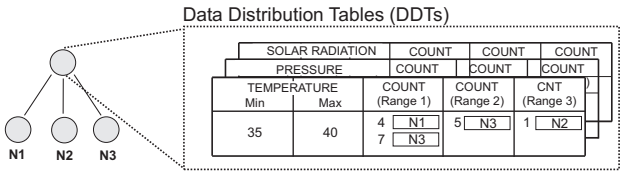

Fig. 1: Format of data distribution table

4. AI-LMAC: AN ADAPTIVE, INFORMATION-CENTRIC AND LIGHTWEIGHT MAC PROTOCOL FOR WSNS

AI-LMAC [3] is a TDMA-based protocl that is an adaptive and information-aware version of the LMAC protocol [XX]. The main idea of the protocol is to assign multiple slots to certain nodes in the network that are expecting more data to flow through them than other nodes in response to a query that has been injected into the network. Such a scheme helps allocate resources to the areas of the network that require more attention thus reducing both latency and buffer overflows.

Every node running AI-LMAC maintains Data Distribution Tables (DDTs) shown in Figure 1. When a node receives a query it looks up its DDTs to deduce how many of its children are going to respond to the particular query. Every node maintains its own set of DDTs each of which is a table representing a particular type of sensor that is present within its set of child nodes. So if for instance a node and its children possess temperature and air pressure sensors, then the node would have two separate DDTs. A DDT is built over time by monitoring the data that flows through it. In other words, statistics about the data flowing through the network is collected without incurring any extra overhead. It is simply based on the data that already needs to flow from the leaf nodes to the root. Thus statistics are collected using only locally available information. Upon receiving a reading from a child node, the parent node updates the entry in the appropriate DDT depending on a number of variables: the type of sensor that originally generated the reading, the region where the reading originated from and the value of the reading itself. Additionally, the DDT also keeps track of the maximum and minimum readings obtained and also which particular immediate neighbor sent it the reading.

In AI-LMAC, we assume that a parent-child relationship exists between all the nodes in the network, such that the root of the network can be considered to be the highest parent in the hierarchy. Using the DDT, every node would know how much ”importance” to give every one of its immediate children.

Using the DDTs a node cannot decide by itself, how much importance it should give itself to transmit. This is because the DDTs only contain information about a node's active child information. A node is not aware of the data generated by the other children of its own parent node as they may not be in range. Thus, the parent is the only node that has knowledge of the proportion of data that will be contributed by each of its immediate children. The idea here is that if a node realizes that

a subset of its immediate children is going to transmit large quantities of data, then more attention needs to be paid to this particular subset of child nodes. In this case, when we say more attention, we actually refer to assigning multiple slots to a particular child.

However, even though a parent node knows which child node deserves more slots to be assigned to it, it cannot send such a rigid instruction to its children as in LMAC. This is because in LMAC, when a node performs slot assignment, it has knowledge of the slot ownership of its first and second order nodes. In this case, the parent node would not know slot ownership information about the slot assignments of its child node's second order nodes since they are three hops away.

Thus, the responsibility of the parent node is simply to "advise" the child, i.e. the parent node sends a message to every one of its children indicating the ideal number of slots that a particular node should take up under the current conditions. It is then up to the child node to follow the advice as closely as possible. This naturally depends on the number of empty slots available.

The process of giving advice starts at the root node of the tree when a query is first injected into the network. This process then percolates down the branches of the tree towards the leaf nodes. If however, the process of giving advice started at an intermediate node, this would increase the chance of performing unfair slot allocations. This is because a node assigning slots would not be aware of the bandwidth requirements of all its sibling nodes which are not within its direct range. From this argument, it is obvious that if we apply this rule repeatedly, the root node is the only node which can assign slots fairly at the beginning. We term this as *horizontal fairness* as the mechanism ensures that all sibling nodes (i.e. a the same level) under a certain parent are allocated slots fairly.

Apart from establishing a horizontal relationship between nodes, we also introduce a mechanism to include *vertical fairness*. In order to prevent buffer overflow problems, our mechanism ensures that that the total number of slots assigned to the immediate children of a certain parent node, does not exceed the number of slots owned by the parent. This reduces the likelihood of data packets being dropped due to lack of bandwidth. Furthermore, leaf nodes are prevented from being allocated excessive bandwidth using this mechanism.

Thus introducing *two dimensional fairness* ensures that the number of slots taken up by a node does not only depend on its siblings but on its parent as well. Once a node has received the ideal number of slots it should take up, it checks to see which slots are free within its 2nd order neighborhood. Just like in LMAC, once a node decides to take up a certain slot, it "marks" the slot using a "1" to indicate that the slot has been taken up.

A. Experimental Analysis

Our framework provides a mechanism to assign more bandwidth to those parts in the network that encounter more data traffic than others. In fact, the assigned bandwidth is proportional to the expected traffic. Hence our framework

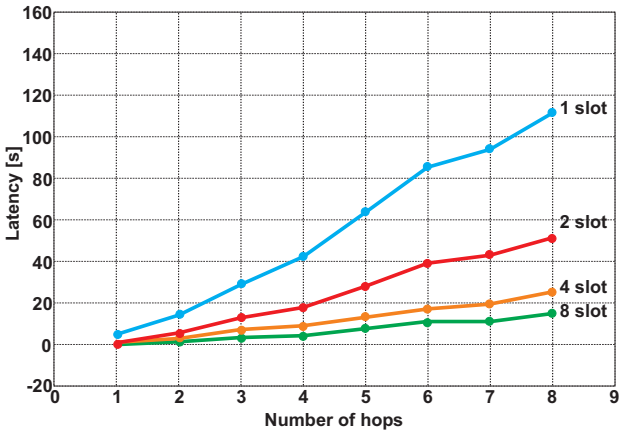

Fig. 2: Average latencies for different number of maximum allowable slots

is able to minimize the overall latency in the network and also the number of messages which need to be buffered in the nodes. Figure 2 illustrates how latency is reduced as the maximum number of allowable slots that can be owned by a node is increased from 1 to 16. These results are obtained by simulation using the discrete event simulator OMNeT++. Results are averaged over five different network topologies consisting of 49 nodes and one root. Ten different runs were carried out per topology. The results clearly indicate that latency is proportionally reduced with the maximum number of controlled slots. However, this holds true only until eight slots. For the sixteen slot scenario, the number of free slots in the network rapidly decreases with every hop from the root and thus the nodes are not able to comply with the advice. Consequently, a bottleneck is created at a few hops (6 to 8) from the root, resulting in higher latency for messages created in those areas.

5. Data Aggregation using a Distributed and Self-Organizing Scheduling Algorithm for Wireless Sensor Networks

Densely deployed WSNs allow environmental monitoring at extremely high spatial and temporal resolutions. However, extracting the raw data from such networks can have problems, e.g. batteries may get drained rapidly due to excessive operation of the transceiver or data quality may deteriorate due to dropped packets caused by network congestion. To solve these problems, we exploit the high degree of spatial correlation that exists between the sensor readings of adjacent nodes in a densely deployed network. Thus, instead of every node transmitting individual readings, we appoint a subset of nodes, referred to as *correlating nodes* that transmit the messages representative of all the remaining nodes at any given point in time. Every correlating node initially transmits information to the sink, indicating the correlation of its readings with its adjacent neighbors. Subsequently, it continues transmitting its own readings until a change in correlation is detected, in which case, it transmits an updated correlation message. The sink then *estimates* the readings of the adjacent neighbors of the correlating node by combining the current readings of the

correlating node with the previously transmitted correlation information. We present a completely distributed and self-organizing scheduling algorithm that (i) prevents two adjacent nodes acting as correlating nodes simultaneously, (ii) increases the robustness and accuracy of the readings by giving every node a chance at some point to act as a correlating node. This ensures that no node is always represented only by estimated readings.

The primary objective of $\mathcal{DOSA}$ [4] is to help decide *when* a particular node should act as a correlating node and thus represent the sensor readings of its 1^{st} order neighbors. During the correlating node's schedule, the node initially transmits correlation information to the sink node followed by its own sensor readings. All the 1^{st} order neighbors *do not* transmit their sensor readings to the sink during this period. Since $\mathcal{DOSA}$ is intended to solve a scheduling problem, we make use of a distributed graph coloring algorithm to assign schedules to individual nodes. From a graph theoretic point of view, since no two adjacent nodes can act as correlating nodes simultaneously, all the nodes chosen by $\mathcal{DOSA}$ to be correlating nodes need to form an *independent set*. Additionally, the correlating nodes for a particular instant of time need to form a *dominating set* since every non-correlating node must be joined to at least one correlating node by some edge.

To hasten the rate of assigning schedules to the nodes, $\mathcal{DOSA}$ utilizes the information provided by LMAC. Instead of coloring all the nodes from scratch, $\mathcal{DOSA}$ meets its requirements by building up on the colors already assigned by LMAC. An added advantage of this form of cross-layer optimization is that a lesser number of messages need to be transmitted for all the schedules to be assigned properly as we make use of information that already exists. Furthermore, $\mathcal{DOSA}$'s dependence on LMAC makes it more reactive to changes in topology as any changes in neighborhood detected by LMAC are immediately filtered to $\mathcal{DOSA}$.

$\mathcal{DOSA}$ uses a distributed graph coloring approach to decide when a particular node should be a correlating node. Every color owned by a node represents a particular frame of time during which a node is required to act as a correlating node. In conventional graph coloring approaches, colors are assigned to vertices such that adjacent vertices are assigned different colors and the number of colors used is minimized. While $\mathcal{DOSA}$'s graph coloring approach also ensures that adjacent nodes in the network do not own the same colors it differs in the sense that each node is allowed to own *multiple* colors, i.e. a node can have multiple schedules. Moreover, the number of colors used in $\mathcal{DOSA}$ is fixed and is equal to the number of slots that are assigned to an LMAC frame.

Before describing the details of the operation of $\mathcal{DOSA}$, we first state the constraints derived from the requirements stated above which define its behavior: (i) *Two adjacent nodes cannot own the same colors. This is because two adjacent nodes should not be assigned as correlating nodes in the same time instant*, (ii) *All colors should be present within the one-hop neighborhood of node v, i.e. if node v does not own a particular color itself, the color* ***must*** *be present in one of its neighboring nodes that is one hop away. This ensures that every node's readings will be represented at the sink node for every time instant either directly or through a correlated reading.*

As mentioned in Section 3, LMAC assigns a slot to every node in the network. $\mathcal{DOSA}$ begins its distributed coloring scheme by considering the initial slot assignment phase in LMAC as an input. Slot assignments in LMAC correspond to partial color assignments in $\mathcal{DOSA}$. Thus while LMAC assigns every node with a single color, $\mathcal{DOSA}$ assigns the remaining colors that ensure the adherence to the constraints (i) and (ii) stated above.

The dependency of $\mathcal{DOSA}$ on LMAC allows nodes to adapt autonomously and immediately to changes in network topology. For example, the addition or removal of a node results in the change being reflected in the LMAC Neighbor Tables of all other neighboring nodes within range. $\mathcal{DOSA}$ detects changes in LMAC's Neighbor Table and performs a re-assignment of schedules if any of the neighboring nodes do not meet the constraints mentioned above. Utilizing such cross-layer information from LMAC ensures that $\mathcal{DOSA}$ does not spend additional resources trying to detect topology changes itself.

A. General Operation of $\mathcal{DOSA}$

$\mathcal{DOSA}$ uses a *greedy* approach to assign colors to nodes. Coloring is performed using two types of colors: *LMAC Colors* and $\mathcal{DOSA}$ *Colors*. LMAC Colors refer to the colors that have been assigned by LMAC - due to the slot assignment. $\mathcal{DOSA}$ Colors refer to the additional colors that are assigned by $\mathcal{DOSA}$ to ensure that constraints (i) and (ii) are met. This occurs *after* the LMAC colors have been assigned. $\mathcal{DOSA}$ does not have any control over the LMAC Color of a node as it depends purely on the slot assignment performed by LMAC. In fact, such control is also not required. Therefore, in the following, we refer to $\mathcal{DOSA}$ Colors simply as colors unless otherwise indicated.

Colors are acquired based on a calculated priority. A node computes its priority within its one-hop neighborhood based on its degree and node ID. The higher the degree of a node, the higher its priority. If two neighboring nodes have the same degree, priority is calculated based on the unique node ID; the node with the larger node ID will have the higher priority.

Once all nodes have acquired their LMAC slots, a *BeginSecondPhase* message is injected into the network through the sink node requesting the nodes to begin the $\mathcal{DOSA}$ coloring phase. At this stage, every node receiving the *BeginSecondPhase* message only has an LMAC Color and does not satisfy the constraints mentioned earlier. Thus these nodes mark themselves as `Unsatisfied`. A node only attains the `Satisfied` status when it satisfies the two constraints mentioned in Section 5. Upon receiving the *BeginSecondPhase* message, a node broadcasts the *NodeStatus* message. This message contains information about the node's status (i.e. Satisfied/Unsatisfied) and the list of colors owned. The *ColorsOwned* field is a string of $|\mathbf{K}|$ bits where every color

Algorithm 1 $\mathcal{DOSA}$ - Normal Initialization

Input: NodeStatusMSG(SatisfiedStatus(TRUE/FALSE), ColoursOwned)
Output: NodeStatusMSG(SatisfiedStatus(TRUE), ColoursOwned)/ NIL

```
1:  UPDATE(LocalInfoTable, v)
2:  if LocalInfoTable contains entries from ALL adjacent nodes then
3:      if SatisfiedStatus(v)=FALSE then
4:          Compute PRIORITY(v)
5:          if PRIORITY(v)=Highest then
6:              C_v ← K \ C_Γ'(v)
7:              ColorsOwned← C_v
8:              SatisfiedStatus← TRUE
9:              UPDATE(LocalInfoTable, v)
10:             BROADCAST NodeStatusMSG(Degree, SatisfiedStatus, ColoursOwned)
11:         end if
12:     end if
13: end if
```

owned by a node is marked with a 1. Note that $|\mathbf{K}|$ is the number of slots per frame in LMAC. The rest of the bits are marked with a 0. Initially, a node only marks its own LMAC Color as 1 due to the initial LMAC slot assignment. A neighboring node that receives the *NodeStatus* message then performs coloring using $\mathcal{DOSA}$ as outlined in Algorithm 1. Note that the *NodeStatus* message is the only message that is used for the operation of $\mathcal{DOSA}$.

We now briefly describe the operation of $\mathcal{DOSA}$ outlined in Algorithm 1. Upon receiving a *NodeStatus* message, a node first updates its *LocalInfoTable* (Line 1). This table stores all the information contained in the *NodeStatus* messages that are received from all the adjacent nodes. Once a node receives *NodeStatus* messages from *all* its immediate neighbors (Line 2), and if its status is `Unsatisfied`(Line 3), the node proceeds to compute its priority. PRIORITY computes the priority of a node *only* among its unsatisfied neighbors (Line 4), i.e. as time progresses and more nodes attain the `Satisfied` status, PRIORITY needs to consider a smaller number of neighboring nodes. The highest priority is given to the node with the largest degree among its adjacent `Unsatisfied` neighbors. If more than one node has the same degree, then the highest priority is given to the `Unsatisfied` node with the largest NodeID.

The node that has the highest priority among all its immediate unsatisfied neighbors, acquires *all* the colors that are not owned by any of its adjacent neighbors (Line 7). As the node has then satisfied both constraints of $\mathcal{DOSA}$, it switches to the `Satisfied` state, updates its own *LocalInfoTable* and informs all its neighbors through a broadcast operation (Lines 8-10). Note that this technique corresponds to a highest degree greedy approach.

Figure 3 provides a step-by-step example of how the $\mathcal{DOSA}$ algorithm assigns colors to the nodes in a network. We make the assumption in the example that LMAC uses 16 slots.

B. Experimental Analysis

In this subsection, we have carried out simulations to illustrate the benefits of $\mathcal{DOSA}$ in terms of network lifetime and data quality. Note that we define network lifetime as the total time taken before the death of the first node in the network.

Figure 4(a) shows the total number of sensor readings that are generated during a 10 minute interval using both data

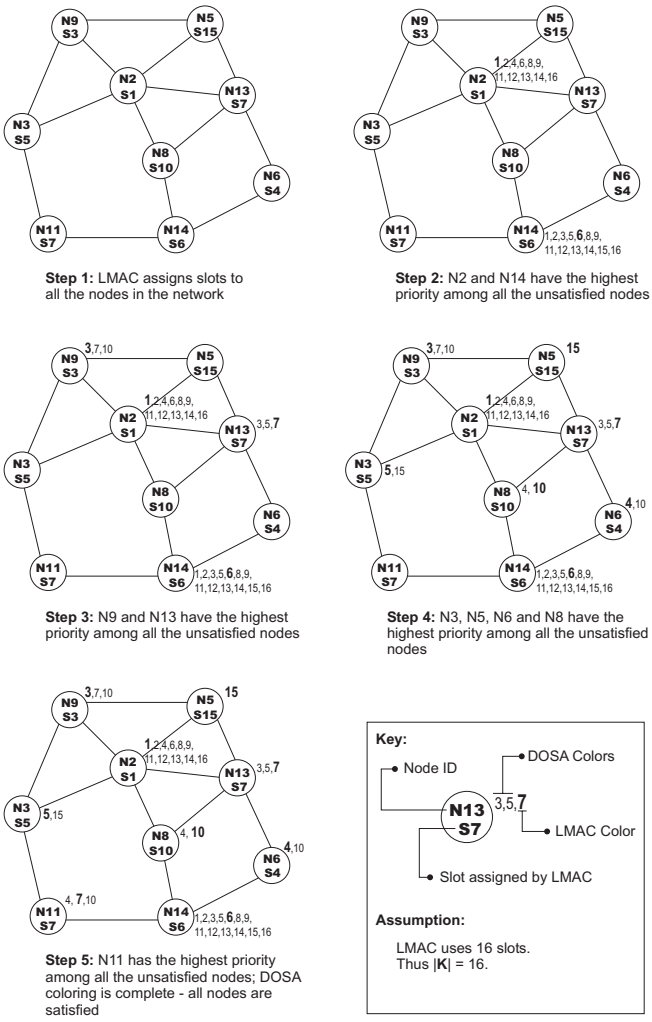

Fig. 3: A step-by-step example of how $\mathcal{DOSA}$ colors are assigned

collection techniques. Figure 4(b) shows the total number of transmit operations performed by all the nodes in the network for the entire duration of the simulation. One can clearly see that Figures 4(a) and 4(b) do not have similar shapes. This is primarily because both raw data collection and $\mathcal{DOSA}$ experience heavy message losses for high sampling rates. The left hand side of the graphs in Figure 4(b) tend towards each other as the limit of the maximum throughput of LMAC is nearly reached. It is this same characteristic that gives the shape of the network lifetime graph in Figure 4(c). Since the total number of message transmissions are nearly the same for both data collection methods at high sampling rates, the network lifetime is also quite similar. It can be seen from Figure 4(c) that $\mathcal{DOSA}$ can help network lifetime improve by up to 83.5% (Epoch = 120s) as compared to raw data collection.

Apart from helping to improve network lifetime, $\mathcal{DOSA}$ also has a significant positive impact on the quality of data collected. When analyzing dropped messages for both data collection scenarios, it is important to realize that every message generated under the $\mathcal{DOSA}$ scheme carries a lot

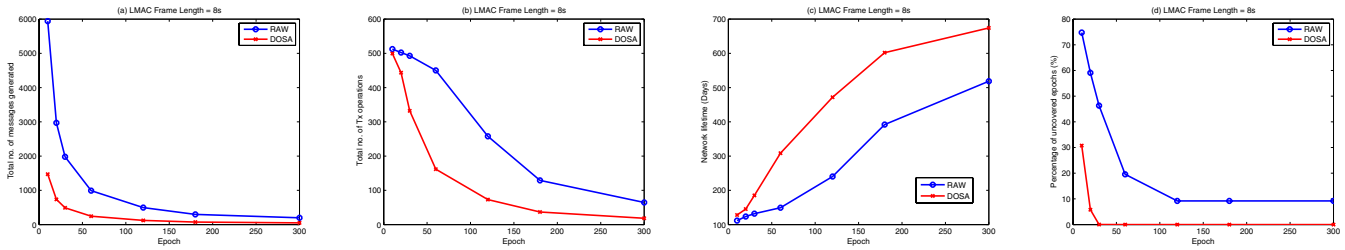

Fig. 4: (a) Total number of messages generated, (b) Total number of transmit operations, (c) Network lifetime, (d) Percentage of uncovered epochs

more weight than a single message in the raw data collection process. This is because a single sensor reading transmitted by a node n under the $\mathcal{DOSA}$ scheme represents not only the reading of n but also those of its adjacent neighbors. Due to this reason, we analyze data quality by observing the number of epochs which are *not represented* at the sink instead of simply counting the number of dropped messages. As an example, suppose a message generated by node n representing its own reading and its neighbors, q, r, and s for the epoch E is lost on the way to the sink due to a buffer overflow event. This would mean that during epoch E, the sink would not have any readings for nodes n,q, r, and s. Based on this example, we present the results of data quality in Figure 4(d). At high sampling rates e.g. when the Epoch is 10s, raw data collection results in around 75% of uncovered epochs while $\mathcal{DOSA}$ results in only 30% uncovered epochs. Uncovered epochs under $\mathcal{DOSA}$ quickly reduces to 0 and remains there as the sampling frequency is reduced. For raw data collection however, the percentage of uncovered epochs levels off at around 10%. We now explain this leveling off characteristic.

Usually a node drops messages when its buffers get filled up. Thus the higher the sampling rate, (i.e. the smaller the value of the Epoch) the larger the proportion of nodes in the network that experience buffer overflows. This naturally also increases the number of lost messages and in turn the percentage of uncovered epochs. However, as the sampling rate is reduced, the number of nodes experiencing buffer overflows may not keep on decreasing to zero. In most topologies, due to the simultaneous generation of messages by all nodes in the network, there will be a certain set of nodes that will *always* experience buffer overflows and will only allow a *fixed* number of messages to successfully traverse towards the root. Thus for low sampling rates, in every epoch, only a fixed number of messages will reach the root regardless of the chosen epoch. It is this characteristic that causes the percentage of uncovered epochs to level off for low sampling rates.

Additionally, Figure 5 shows the benefit of having $\mathcal{DOSA}$ use underlying cross-layer information from LMAC when a node is removed. The total number of messages transmitted by all the nodes was compared over 9900 node deletions with and without cross-layer information being used. When it is not used, every neighbor of the dead node has to transmit a *NodeStatus* message regardless of its status. The results indicate a savings of up to 42% when cross layer information is used.

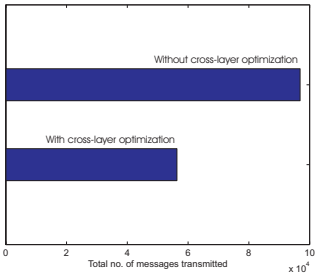

Fig. 5: Number of messages transmitted over 9900 runs with and without cross-layer information.

6. Conclusions and Future Work

In this paper we have presented two examples which show how cross-layer optimizations can be carried out in WSN architectures between the application layer and MAC layer. However, when using cross-layer optimizations, any disturbance in the lower layers could have a detrimental impact on the upper layers. For example, link quality may deteriorate due to adverse weather conditions. This could cause frequent topology changes. This in turn would cause $\mathcal{DOSA}$ to continuously rearrange its scheduling scheme thus reducing the efficiency of the algorithm. Thus further research is required to ensure that the higher layers are well insulated from the instability of the lower layers.

References

[1] T. Melodia, M. C. Vuran and D. Pompili, "The state of the art in cross-layer design for wireless sensor networks", In: *Proc. EuroNGI Workshops on Wireless and Mobility*, Como, Italy, July 2005.

[2] L.F.W. van Hoesel and P. Havinga, "A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing Preamble Transmissions and Transceiver State Switches", In: *First International Conference on Networked Sensing Systems*, Tokyo, 2004.

[3] S. Chatterjea, L.v. Hoesel and P. Havinga, "AI-LMAC: An Adaptive, Information-centric and Lightweight MAC Protocol for Wireless Sensor Networks", In: *Proc. ISSNIP*, Melbourne, Australia, December 2004.

[4] S. Chatterjea, T.Nieberg, Y.Zhang and P. Havinga, "Energy-Efficient Data Acquisition using a Distributed and Self-organizing Scheduling Algorithm for Wireless Sensor Net", In: *Proc. DCOSS*, Santa Fe, NM, USA, June 2007.