CAMAW: A Clustering Algorithm for Multiple Applications in WSAN

Elton A. Costa, Luci Pirmez, Claudio M. de Farias, Flávia C. Delicato
Programa de Pós-Graduação em Informática
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
E-mail: {eltonalvescosta, luci.pirmez, cmicelifarias, fdelicato}@gmail.com

***Abstract*—This paper proposes a clustering algorithm tailored for multiple applications in Wireless Sensor and Actuator Networks (WSANs) called Clustering Algorithm for Multiple Applications in a WSAN (CAMAW). CAMAW is an application aware clustering algorithm, since besides sharing the WSAN infrastructure with multiple applications simultaneously, it clusters the nodes according to each application requirements. The main benefits of using CAMAW are: (i) it is an energy-efficiency algorithm for WSANs since it reduces data traffic, by multiplexing data of a same monitoring type for several applications and (ii) is a dynamic clustering algorithm because it organizes WSAN in groups faces the arrival and the departure of running applications at runtime. CAMAW outperforms the traditional clustering algorithms regarding network lifetime in all considered scenarios.**

Keywords-Clustering; Application-aware; Wireless sensor networks; multiple applications.

I. INTRODUCTION

Recent advances in micro-electromechanical systems and wireless communication technologies have enabled the building of low-cost and small-sized sensors nodes, which are capable of sensing, processing and communicating through wireless links [1]. Wireless Sensor and Actuator Networks (WSANs) are composed of tens, hundreds or even thousands of sensor nodes [1]. Nodes in WSANs commonly rely on non-rechargeable batteries as their energy sources, and the replacement of depleted batteries is not always feasible or desirable. The data gathered by the different sensor nodes is transmitted to one or more sink nodes, which are connected to other networks, such as Internet. These sink nodes have more processing power and are powered by an unlimited energy source. Actuator nodes are able to convert an electrical signal (a virtual command) into a physical phenomenon (an action) as sounding alarms, switched on/off electric appliances or closing gates.

Traditionally, WSANs were designed for a single purpose, a single application. Specifically, each network node was programmed to collect and process data for a single application. This approach is known as fit-for-purpose [2]. In the single-application approach, each new application is bundled with a WSN at the time of deployment. This sensor network design usually incorporates redundancy in the sensor deployment to ensure the successful execution of the target application and to meet the defined quality of service (QoS) requirements. This approach is not concerned with the reuse of software artifacts and the resource sharing. If this same approach is used to support multiple applications belonging to different organizations, this leads to redundant deployments, wasting energy. Independent sensor networks dedicated to a specific applications are not the most cost efficient, or the most practical deployment technique under a wide variety of conditions, for example large-scale networks having thousands of nodes or covering large geographical areas, such as urban areas [4]. An example can be seen in [3], in which a WSAN is used to monitor a smart building. Now consider that there are two users interested in the building. The first is the building conservation board, as it needs to make sure that the building is in conditions to receive employees. The second is a company that has rent the building for its operation. It is quite possible that the conservation board has already deployed its own WSAN to monitor the environment. In this case, the company can reuse the existing sensor nodes during the company work period. The sensors could monitor temperature, luminosity, humidity and several other environmental parameters. Those sensors could be used for different applications. A temperature sensor can be used by air-conditioning and by fire detection application. Without sharing those sensors there would be two WSANs, one for each user. Virtualization [4] is a technology that can aid in tackling this issue, as it enables the sharing of resources/infrastructure by multiple applications/users.

According to the authors in [5], there are two categories of WSAN virtualization: node level and network level. In the network level virtualization, a subset of sensor nodes belonging to a deployed network is assigned to execute the tasks of given application at a given time, while the other sensor nodes remain available for other application tasks. Such subset composes a virtual sensor network (VSN). By considering that each subset is dedicated to an application, a WSAN can be utilized by multiple applications concurrently, thus realizing the (network level) virtualization. In [4], sensor nodes form clusters to support applications that monitor dynamic phenomena. The sensor nodes within each cluster execute application(s) tasks, meaning a sensor node can be part of multiple clusters. Therefore, clustering is a key feature to provide network level virtualization and allow sharing the network resources among multiple applications.

Clustering algorithms are responsible for organizing the network in groups, called clusters. Clusters generally have a cluster leader, called Cluster Head (CH), and a set of member sensor and actuator nodes, called cluster members (CM). The main role of a CH is to receive the data collected by the sensors of its cluster and route it towards the sink node using either one hop or multihop communication. Since data communication is an energy-demanding operation and

 ISBN: 978-1-61208-439-8

the overall distance among cluster members and their respective cluster-head is generally smaller than the distance among these nodes and the sink, cluster members save transmission energy thus contributing to increase the network lifetime [6]. Cluster members can collaborate about recent data measurements and determine how much information should be transmitted to the sink node [1]. A CM usually chooses which CH to associate itself through a mechanism that uses some distance-based criteria [2][7], such as received signal strength indicator (RSSI) and shortest communication distance, among others. A drawback of most existing clustering algorithms for WSAN is that they are typically designed to meet the requirements of a single target application. Usually, the traditional clustering algorithms form the clusters based on the geographical position of the nodes (defined by both GPS positioning or RSSI). Therefore, these algorithms may include nodes in the clusters that do not attend to the requirements of an application, since they are unaware of them. Also, the nodes resources would not be shared among the different applications simultaneously running on the network, representing a waste of energy.

Several challenges arise for designing clustering algorithms for multiple applications. Different applications may have different target areas, different monitoring interests (in terms of type of sensing data), and different data sensing and data transmission rates. In the multiple applications approach there will be several clusters, each one carrying out the monitoring tasks of a given application. Nodes can belong to more than one cluster simultaneously and change among clusters over time. Those clusters must share among them common data, avoiding repeating common tasks.

Considering the aforementioned characteristics, this paper proposes an application aware clustering algorithm, called Clustering Algorithm for Multiple Applications in a WSAN (CAMAW), since it clusters nodes based on the application's area of interest and requirements. CAMAW enables the resource sharing among multiple applications, hence realizing the network-level virtualization. It allows the sensor nodes to attend several applications from groups that were created keeping in mind the matching of the nodes sensing resources and the applications requirements. In other words, CAMAW promotes a rational use of the network resources because it first clusters the nodes strictly according to the applications requirements, which restricts both the interest area, as the apt set of nodes to be clustered. Second, it enables the sharing of the network resources between applications, given the ability of CAMAW to identify commonalities between sensing requirements of the different running applications as an opportunity to reduce sensing and communication efforts. CAMAW uses both features as a way to save energy and to prolong the network lifetime.

This paper is divided as follows: Section II reviews the related works. Section III presents CAMAW, our proposed clustering algorithm for multiple applications in WSNs. In Section IV, we describe the experiments to evaluate the proposal. Section V concludes this paper and outlines future work.

II. RELATED WORKS

Several works have proposed WSN virtualization approaches. The work of Khalid et al. [8] proposes a middleware framework for network virtualization for Smart Home and Ambient Assisted Living (SHAAL). SHAAL is based upon the virtualization of sensor network that enables multiple applications to run on a network with heterogeneous nodes. In SHAAL, a single application can be distributed over a number of clusters, where a node is capable of participating of several clusters. Moreover, the sharing of the infrastructure is made possible by an abstraction layer that resides at each sensor node. The virtual manager, i.e., the core of the middleware, has to sure that the clusters are made dynamically according to the application requirements. SHAAL like CAMAW organizes the WSAN dynamically considering the arrival and departure of applications. However, CAMAW intends to share the monitoring data between applications with common interests on data, in order to minimize monitoring efforts and therefore saving energy.

Another work, SenShare [2] attempts to address the technical challenges arise from the network level by constructing overlay sensor networks which are not only responsible for providing the most suitable members to perform tasks from applications, but also isolating the network traffic of a target application from the network traffic generated by other applications or the supportive mechanisms used to maintain the network overlay. For achieving the goal of traffic isolation, SenShare extends each application packet at the runtime with a 6 bytes long application routing header, but the entire network message is still formatted under the IEEE 802.15.4 standard. Since the sensor nodes of a cluster can be located in anywhere within the network, the nodes with allocated tasks and physical neighbors that can communicate with single hop messages are then formed in a cluster. This generally results in a number of clusters that are isolated from each other. For constructing a WSN from these clusters as a single connected application-specific network, virtual links between the clusters need to be established with the help of nodes that are not performing tasks from the target application. Virtual links between clusters are incrementally generated by three consecutive steps, where 1) identify the nodes that are on the edges of a connected node cluster, 2) discover optimum paths from the nodes selected in the previous step that connect the local cluster to other clusters, and 3) ensure all the clusters are connected together and can access the network's sink. In SenShare, several instances of the same RSSF could run in isolated, one per application, while in CAMAW all applications run in a single instance, which enables to find and eliminate redundancies in sensing and communication, according to the common applications requirements.

The work of Caldas *et al.* [9] proposes S-LEACH, an application aware cluster-based routing algorithm for shared sensor networks because is designed to deal with several applications simultaneously sharing the same infrastructure of wireless sensor network. Therefore, in S-LEACH the clusters formation is created in order to route the data for

 ISBN: 978-1-61208-439-8

multiple applications by transmitting these data once. Besides, by considering a context of shared applications in a common sensors infrastructure, the CH nodes of S-LEACH use data fusion algorithms designed for Shared Sensor Networks [10] instead of traditional fusion techniques. In S-LEACH, the clustering process is unaware of the applications, which means that it first organizes the whole WSAN in clusters and, only then, takes notice of the applications in order to promote the sharing of the collected data. While CAMAW only organizes clusters according to the applications requirements as a way to restrict the application interest area and also minimize the clustering, monitoring and transmission efforts. It also helps to avoid the clustering of nodes that are unnecessary.

Finally, the authors in [11] present a clustering algorithm called self-configurable clustering (SCCH). SCCH firstly clusters the sensor nodes and selects the CHs (cluster heads). To define CHs a fuzzy system is used and local information of each sensor node is considered. The output of the fuzzy system is a value representing the eligibility of sensor nodes to be CHs. Then, nodes in the network compare their eligibilities against others'. A node with the maximum eligibility value will introduce itself as a CH and the rest of the nodes as backup CHs (BCHs). As a result, the CMs (cluster members) can ensure that there is always a BCH for their CHs. Therefore; in case of CH failure the CMs can replace the BCH with the permanent CH failure. CAMAW is different from SCCH because: (i) it is designed for clustering multiple applications while SCCH is for WSANs; (ii) CAMAW is an application-aware while SCCH is concerned about the nodes location in the monitored area.

III. CAMAW

CAMAW is a clustering algorithm executed periodically in all nodes of a WSAN. There is one cluster (and its respective CH) for each application. Each period of execution is a cycle. The cycle begins by synchronizing all nodes in the WSAN, for this procedure we may use a well-know synchronization algorithm such as the one presented in [12]. Then, the nodes wait for messages coming from the Sink Node. If the message type is for creating a new application, CAMAW is responsible for clustering the nodes for such application according to node capacities and application requirements. Otherwise, if the message is for terminating an application, two cases are possible: first, if the application is the only application in the cluster, the node should maintain the cluster formation but stop all monitoring activities; second, if there are other applications in the network, the nodes shall free the resources used by this application while maintaining the nodes working.

CAMAW is only concerned about the clusters formation. Other procedures such as data collection and data fusion are out of scope of our work.

A. Data Structures

The network is composed of a set V of sensor nodes $v_i \in V$, where $V = \{v_1, v_2, ..., v_n\}$ and of a set of applications $a_j \in A$, where $A = \{a_1, a_2, ..., a_m\}$. A node may perform monitoring tasks for 0 to m applications simultaneously. During the algorithm execution there are two possible states for the applications in the network: *Active* or *Inactive*. An application is *active* if there are sensor nodes monitoring for this application. An application is *inactive* if there is no cluster in the networking performing monitoring tasks in its behalf.

The data structures used by CAMAW (stored in every node) are *NodeCapabilities* and *AppRequirements*. *NodeCapabilities* stores the *NodeID* (a unique node identifier, such as the node MAC address), node's capabilities regarding types of monitoring interfaces (*TpMnt*) and rate in use (*TxUse*), a list of all physical neighbors and the node's residual energy. *AppRequirements* stores the Application identifiers (*AppID*) of the applications in *active* state supported by the sensor node. Besides, for each application *AppRequirements* also stores the monitoring interests expressed in terms of: time that the application can remain running on the network, i.e., the duration of the application (*TDur*); the monitoring requirements (sensing unit (*TpMnt*) and Rate (*TxApp*)), the node's role (CM or CH) for this application and the *ID* of the CH. It also stores a list of all *NodeIDs* neighboring nodes able to monitor for this application (*NeighborSet*). Additionally, for each neighboring nodes this structure stores an utility value that informs how promising a node is in order to become CH for a given application. This value is calculated by the function W described in D.2.*a*. This structure also stores the geographical location (POS), which indicates the position of the center of the area of interest and its radius (x, y, r). Finally, the data structure also stores Aptitude, the information if the node is apt to monitor for a given application (0 = not apt and 1 = apt). A node is considered apt if this node (i) has one or more sensing units that are of interest for the application and (ii) is located at the application area of interest. We introduce an availability function that indicates whether a sensor can provide the required service at the specified area. The function is shown below.

$$A(t,x,y,i) = \begin{cases} 1, & \text{if a sensor is available} \\ 0, & \text{if a sensor is unavailable} \end{cases} \quad (1)$$

Where t is the sensing unit that an application requires, x and y are the geographical location for the monitoring event and I is the Sensor ID.

B. CAMAW Procedure

In the following subsections we will provide a detailed explanation of our algorithm. It encompasses three phases: (i) *Setup* (Section C) is responsible for configuring the algorithm initial parameters. (ii) *Application Arrival* (Section D) is responsible for clustering the nodes according to node capacities and application monitoring requirements and (iii) *Application departure* (Section E) is responsible for reorganizing the network in the event of an application end. The Pseudo-code of CAMAW can be seen in Figure 1:

 ISBN: 978-1-61208-439-8

Input: Applications that are deployed on the Network *(AppRequirements)*, *NodeCapabilities*
Output: Clusters by application

```
1.  # SETUP PHASE
2.  Fill NodeCapabilities
3.  For each new Round
4.      Execute a synchronization algorithm
5.      if it is not the first round
6.          for each application j
7.              ROLE_SELECTION_PROCEDURE( )
8.              ASSOCIATION_PROCEDURE( )
9.  Wait for messages
10. If message = BS_NEW_APP
11.     For each Application A_i in A
12.         APPLICATION ARRIVAL PHASE ()
13. Else if message = BS_END_APP_ or If (node role = CH and tDur expired)
14.         APPLICATION DEPARTURE PHASE ()
```

Figure 1. CAMAW cluster formation procedure

C. Setup Phase

This phase is responsible for configuring the nodes and inserting values to data structures that will be necessary in other phases. During the *Setup* phase it is also executed a synchronization procedure [12]. The synchronization is important to guarantee spatial and temporal correlation of the data collected by the WSAN. Synchronization makes possible to the algorithm to start data acquisition by several nodes simultaneously. Also, in this phase, for each new round after the first, for each application *j* in the WSAN, the node will execute a Role Selection procedure (described in Section III.D.*b*) for rotating the nodes role. This is used to avoid the energy depletion of the CHs.

D. Application Arrival

This phase is responsible for grouping the nodes into clusters in accordance with *the capabilities of sensor nodes* and *the monitoring requirements of the new application*. This phase is subdivided in the following three procedures: (i) ***Verify*** the ***Aptitude***, (ii) ***Role Selection***, (iii) ***Association.*** In the ***Verify the Aptitude*** procedure the node checks if it is apt to monitor for the new application. In the ***Role Selection procedure,*** each apt node decides its role for the new Application: (i) *Cluster Head* (CH) or (ii) *Cluster Member* (CM). In the ***Association procedure,*** each node is responsible for associating with its respective CH (if the node role is CM) or to wait for the CM to send association requests (if the node role is CH).

The Pseudo-code of this phase can be seen in Figure 2:

Input: Applications that are deployed on the WSAN *(AppRequirements)*, *NodeCapabilities*
Output: nodes with CH Role CH_ID = NodeID).

```
1.  #VERIFY APTITUDE PROCEDURE
2.    Verify if node is apt using (1)
3.   # ROLE SELECTION PROCEDURE
4.    If node is apt AND with no role
5.      Set node rating through (2)
6.      Send CAPABILITIES_EXCHANGE msgs to neighborhood
7.      Wait for CAPABILITIES_EXCHANGE msgs from neighborhood during a fraction of the setup phase slot time of a round
8.      Stores neighbor capabilities from incoming msgs on node's AppRequirements.NeighborSet data structure
9.      For each neighbor node <i> on AppRequirements. NeighborSet
10.       If betterRating <= i rating
11.         Set betterRating to i rating
12.     If nodeRating in AppRequirements.NeighborSet > betterRating
13.       Send NEW_COLLECTOR for all neighboring nodes
14.     Else
15.       Wait for all NEW_COLLECTOR during a fraction of the setup phase slot time of a round
16.       Update CH's candidate capabilities from incoming msgs on AppRequirements.NeighborSet
17.     If node already has a CH role
18.       For each monitor node i in AppRequirements.NeighborSet
19.       For each TpMnt of i in AppRequirements
20.       For each TpMnt of each new AppID in AppRequirements
21.       If TpMnt of new AppID in AppRequirements matches i's TpMnt in AppRequirements.NeighborSet
22.       Add i on newClusterStructure structure
23.       If TpMnt of AppID in AppRequirements do not exists in AppRequirements.AppID
24.       Store TpMnt of AppID in AppRequirements.AppID
25.       Else
26.       If AppRequirements .TxApp of AppID > i's AppRequiremnts.TxApp
27.       Update i'sAppRequirements.TxApp with AppID.TxApp
28.       If newNeighborSet is equal to Neighbors in AppRequirements
          Send CH_END_CLUSTER to newNeighborSet nodes
29.       Send CH_NEW_APP to newNeighborSet nodes
30.     Else
31.       Send UPDATE_SENSORING with new NeighborSet in AppRequirements settings to all nodes in
```

AppRequirements

32. **# ASSOCIATION PROCEDURE**
33. **If** node role = CM
34. Choose the CH with higher *RSSI* on *AppRequirements*
35. Send the CM_JOIN to the chosen CH
36. Update your CH_ID on *AppRequirements* with chosen CH's *nodeID*.
37. **Else if** node role = CH
38. Wait for all CM_JOIN from neighboring nodes.
39. With the data inside incoming msgs from neighboring nodes, update the node's entry on *NeighborSet* as monitor node.
40. Send UPDATE_SENSORING to these nodes present in *AppRequirements*

Figure 2. Application Arrival Phase

1) *Verify Aptitude Procedure*

The objective of this procedure is to determine if the sensor nodes are able to meet the monitoring requirements of the new application. In this procedure, each sensor node waits to receive the BS_NEW_APP message from the sink node. The BS_NEW_APP message contains the monitoring parameters that each new application has. This message has the list of sensing unities (*AppRequirements.TpMnt*) demanded by the applications, its respective rates (*AppRequirements.TxApp*), its localization (*AppRequirements.Pos*) and the duration (*AppRequirements.TDur*). According to this information, the data structure *AppRequirements* is updated. Following, for each new application, the sensor node verifies if it has one or more sensing unit that can support one or more monitoring requirement of this new application. After verifying if it is able to support the requirements of the new application, the sensor node updates its data structure *AppRequirements.Aptitude* with application identifier (*AppRequirements.AppID*) and the monitoring requirement of the new application (*AppRequirements.TpMnt*). If the application identifier (*AppRequirements.AppID*) was included in *AppRequirements.Aptitude*, the next procedure (*Role Selection*) starts. Otherwise, this sensor node remains in a low duty cycle (idle) in order to save its remaining energy.

2) *Role Selection Procedure*

The objective of this procedure is to determine the role of each sensor node *i* for the new applications *j* according to a utility function W_i. First, we present the utility function used in this work to inform "how promising" is a given sensor node *i* in order to become the Cluster Head for the new application *j*. Next, the role selection procedure itself is described.

a) Utility Function

W_{ij} is calculated to measure the utility of a given *i* sensor node for the new application *j* as a function of: (i) the residual energy level of sensor node *i* and (ii) the percentage of neighboring nodes within the radio range of sensor node *i*. The utility function *W(i,j)* is presented in (1):

$$W(i,j) = Xij + Yi \quad (2)$$

Where $X_{i,j}$ indicates the percentage of neighboring nodes for the node *i* according to the new application *j*, Y_i informs the residual energy of the node *i*. $X_{i,j}$ is defined in (3) as the ratio between the number of neighbors of node *i* for the new application *j* divided by the total amount of network nodes represented by *N*. The residual energy is defined in (4) as the current amount of energy of the sensor node *i* divided by the maximum total energy of that node.

$$X_{i,j} = \frac{\sum Neighbors_{ij}}{N} \quad (3)$$

$$Y_i = \frac{\text{E}i \text{ residual}}{\text{E}i \text{ total}} \quad (4)$$

b) Role Selection

The objective of this procedure is to select the appropriate role of the node *i.* In this procedure (see Figure 2), the apt sensor node *i* calculates its utility through the function W_{ij} (2). After obtaining the utility value of the sensor node *i* for the application *j*, this information is stored at the structure *AppRequirements*. On following the sensor node *i* sends to its neighbors the CAPABILITIES_EXCHANGE message (line 7) containing its utility for the application *j* and its capabilities (*NodeCapabilities)*.

The sensor node *i* waits to receive the CAPABILITIES_EXCHANGE message from its neighbors regarding the arrival of a new application *j*. For each CAPABILITIES_EXCHANGE message received and for each application *j*, the sensor node *i* updates its *AppRequirements* structure with the identifiers of its neighboring nodes (*NodeCapabilities.NodeID*), and their respective utilities (line 8). Moreover, it is also updated the types of sensing units (*TpMnt*) that are present in each neighboring node.

With the utility information of each neighboring node, each sensor node *i* now is able to compare its utility value in relation to its neighbors. For each application *j,* the sensor node *i* that contains the highest utility value will send the NEW_COLLECTOR message to its neighbors in order to inform that it is the new CH for application *j* on that region (lines 9-13). The NEW_COLLECTOR message contains the CH identifier (*NodeCapabilities.NodeID*). For each application *j*, the neighbors that received the NEW_COLLECTOR message will become CMs (line 15) for application *j*.

The node *i* verifies in *AppRequirements* if it is a CH for another application, it will verify if the set of CMs in *AppRequirements.NeighborSet* contains only nodes that are apt to monitor for the new application (line 6). If all the CMs

 ISBN: 978-1-61208-439-8

in *AppRequirements.NeighborSet* are apt for monitoring for the new application, then CH updates its *AppRequirements.TxApp* with the more demanding sensing rate (*AppRequirements.TxApp*) and it includes the *AppID* in *AppRequirments* (line 26-27). Else if only some nodes in *AppRequirements.NeighborSet* are apt for monitoring for application *j*, the node *i* will send a CH_END_CLUSTER message (line 28) for those nodes. Then node *i* will send CH_NEW_APP (line 29) for those nodes to perform a new *Role Selection* and *Association* procedures. The CH of this new Cluster will have the *NodeCapabilities.NodeID* of node *i* in its *AppRequirements*, meaning that this new CH will forward its messages to the node *i* (lines 31-42) instead of the Sink node.

c) Association Procedure

For each new application *j*, the sensor node *i* verifies its role. If the node role *i* is CM, this node chooses one CH node to be associated with among the CHs nodes of a given region according to the Signal Strength, i.e., the one with the highest RSSI (*Received Signal Strength Indicator*) value. After choosing the CH node, the CM node sends a JOIN_CLUSTER message to it. This message contains the node's identifier (*NodeCapabilities.NodeID*) and the identifier of the new application *j*. Next, the CM node *i* waits to receive the UPDATE_SENSORING message from its CH node informing that the node can start to collect data for the new application *j*. This message contains the new application's identifier, the monitoring types (*AppRequirements.TpMnt*) and its respective rates (*AppRequirements.TxApp*). With this information about the new application *j*, the CM node *i* updates the fields of *NodeCapabilities.TxUse*. If the node role *i* is CH, this node waits to receive the CM_JOIN message from CM nodes that will be members of the new cluster to the new application *j*. After receiving each CM_JOIN message, the CH node *i* updates in *AppRequirements* the entries referring to each CM nodes responsible for sending the CM_JOIN messages. Following, the CH node *i* sends a UPDATE_SENSORING message for its CMs nodes.

E. Application departure

In this procedure, each sensor node *i* waits to receive the BS_END_APP message from the sink node or the application duration time defined (*AppRequirements.TDur* equals to zero) has finished. The pseudo-code of this phase can be seen in Figure 3.

Input: All Nodes with role defined
Output: free nodes in sleep mode

```
1.  # END APPLICATION# END APPLICATION
2.  If the node is a CM
        Wait for GO_TO_SLEEP coming from the CH
    Else:
    # BS_END_APP MSG ARRIVAL
        Wait for BS_END_APP coming from the BS
3.      For each MsgAppID in BS_END_APP msg
4.          For each AppID in AppRequirements
5.              If MsgAppID is equal to AppRequirements.AppID
6.                  remove AppRequirements.AppID
7.              Else if MaxRate is null OR (MaxRate.TpMnt =
        AppID.TpMnt AND MaxRate.TxApp < AppID.TxApp)
8.                  MaxRate= AppID
9.          If AppRequirements is null
10.             set all NodeCapabilities.TxUse = 0
11.         Else
12.             For each TpMnt in NodeCapabilities
13.                 For each TpMnt in MaxRate
14.                     If NodeCapabilities.TpMnt = MaxRate .TpMnt
15.                         NodeCapabilities.TxUse = MaxRate .TxApp
    # APPLICATION DURATION EXPIRATION
16.         For each AppID in AppRequirements
17.             For each TpDur in AppRequirements.AppID
18.                 If TpDur expirates
19.                     remove AppRequirements.AppID
20.             Else if MaxRate is null OR (MaxRate.TpMnt =
        AppID.TpMnt AND MaxRate.TxApp < AppID.TxApp)
21.                 MaxRate= AppID
22.         If AppRequirements is null
23.             set all NodeCapabilities.TxUse = 0
24.             removeCluster = true
25.         Else
26.             For each TpMnt in NodeCapabilities
27.                 For each TpMnt in MaxRate
28.                     If NodeCapabilities.TpMnt = MaxRate .TpMnt
29.                         NodeCapabilities.TxUse = MaxRate .TxApp
        NodeCapabilities.TxUse = MaxRate .TxApp
```

Figure 3. Application Departure Phase

For each application *j*, the sensor node *i* verifies its role. If the role of the node *i* is CM, it waits to receive the GO_TO_SLEEP message from the CH node (line 2). This message will stop the monitoring tasks of an application (*Nodecapabilities.TxUse* will receive 0). This message contains the *AppRequirements.AppID* of the applications leaving the WSAN. If the node monitors for a single application, it turns to idle. Else, the node stops monitoring for this application but it keeps monitoring for the other applications.

If the role of the node *i* is CH there are two possibilities. First, if the node is CH for a single application (line 9) (there is only one *AppID* in *AppRequirements*), the application is not ended to avoid a new clustering procedure. In this case, it is preserved the cluster structure but with no collecting of tasks or data transmission (setting *NodeCapabilities.TxUse* to 0) (line 14-15). In addition, the nodes enter a state of low duty cycle.

 ISBN: 978-1-61208-439-8

Else, if the node is CH for more than one application, it searches for another application (on *AppRequirements.ApppID*) that monitors for the same monitoring type (*AppRequirements.TpMnt*) (line 13) that the departing application monitors.

If there is no other application that monitors for the same monitoring type, node *i* sends UPDATE_SENSORING message (containing *AppRequirements.AppID*) to all CMs in this application's cluster.

Else, if there is another application monitoring for the same monitoring type (*AppRequirements.TpMnt*), there are two possibilities. First, if the application that is leaving the cluster had the most demanding monitoring rate (*AppRequirements.TxApp*), then the node *i* will update monitoring rate (*AppRequirements.TxApp*) for this monitoring interface (*AppRequirements.TpMnt*) (line 15) using the transmission rate of the application that remains on the cluster. Then it sends a UPDATE_SENSORING (containing the *AppRequirements.AppID, AppRequirements.TxApp, AppRequirments.TpMnt*) message and sent for all CMs. Second, if the application that is leaving the cluster has a less demanding monitoring rate (*AppRequirments.TxApp*) than the departing application, there is no need to update the monitoring rate (*AppRequirments.TxApp)*. In this case, the node *i* sends a GO_TO_SLEEP message containing the *AppRequirements.AppID* of the departing application to all CMs. The CMs will then stop monitoring for it.

IV. EXPERIMENTS

This section describes the experiments conducted to assess CAMAW in terms of network lifetime, energy consumption balance and the node memory used.

A. Experimental Settings

The experiments were conducted in the SUN SPOT platform [13], a sensor platform particularly suitable for rapid prototyping of WSANs applications. The SUN SPOT SDK environment includes Solarium that contains a SPOT emulator useful for experimenting software and/or to create scenarios with a large number of nodes whenever the real hardware is not available. The proposed algorithm was deployed on the SUN SPOT platform rev8 hardware [13]. As mentioned in Section 3, the data collection and data fusion procedures are not CAMAW's responsibility. Although, we implemented those procedures in order to better evaluate the energy consumption of a WSAN using CAMAW. In our experiments, we have used a maximum of 10 applications (1, 2, 3, 5, and 10 applications) simultaneously running in the network. For each application, we assigned two randomly sensing units. Our implementation considered 1 to 5 different sensing units (accelerometers, temperature, light, humidity and presence). For each assigned sensing unit, we randomly assigned sensing rates varying from 1 to 5 seconds, using the procedures explained in [14]. It is discussed in the literature that random monitoring tasks may not always represent real applications; however, the diversity they provide is sufficient for this group of experiments as explained in [14]. The sensing units used in our applications represent the SUN SPOT embedded sensors.

All experiments were performed in a 100m x 100m field. The network sensor nodes are in the Cartesian plane defined in the area {(0,0), (100,0), (0,100), (100,100)}. The sink is located far from any sensor node, at coordinates (200,100). All network sensor nodes starts with 0.5 joules as initial energy within its batteries. We have randomly distributed 51 nodes in the network (50 nodes and 1 sink node). We have used the energy model presented in [6], which is the **first order radio model**. In this model, a radio dissipates $E_{elec} = 50$ nJ/bit to run the transmitter or receiver circuitry and $\epsilon_{amp} = 100$ pJ/bit/m² for the transmitter amplifier. The equations used to calculate transmission costs and receiving costs for a *k*-bit message and a distance *d* are:

$$E_{transmission}(k, d) = E_{elec} * k + \varepsilon * k * d^2 \quad (4)$$

$$E_{reception}(k) = E_{elec} * k \quad (5)$$

Sending and Receiving messages are costly operations; therefore, the usage of these operations should be minimal. Also, it is assumed that the radio channel is symmetric so that the energy required to transmit a message from node *i* to node *j* is the same as energy required to transmit a message from node *j* to node *i*.

B. Metrics

The metrics used for assessing the impact of CAMAW in a WSAN are: (i) the lifetime of the network, (ii) the standard deviation in terms of consumed energy by the nodes at the end of experiments (iii) the memory consumption. In this paper, we adopted the same definition of network lifetime used in [15], which is the time elapsed until the first node in the WSAN is completely depleted of its energy. We have used the Energy Standard Deviation (ESD) as metric for showing CAMAW's energy consumption balance in a WSAN. In this case, all the WSAN sensor nodes form the statistical population. The more the value of the ESD approaches zero, the better the energy consumption balance among nodes is. The memory consumption is defined as the amount of memory used by CAMAW installed in the nodes.

C. Experiments results

The main goal of the first set of experiments is to assess how long the WSANs last using the LEACH, CAMAW and SCCH [11] algorithms by varying the number of applications (1, 2, 3, 5 and 10) simultaneously running on WSAN. Figure 4 shows the network lifetime using LEACH, CAMAW and SCCH and the lifetime gained of the network by CAMAW against LEACH and SCCH for scenarios with 1,2,3,5 and 10 currently running applications.

 ISBN: 978-1-61208-439-8

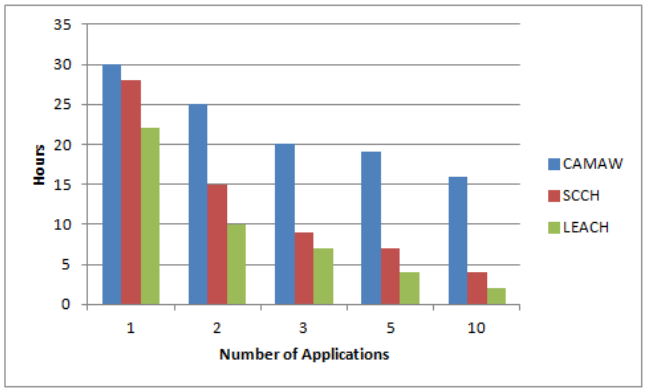

Figure 4. Evaluating System Lifetime

The results of this experiment (see Figure 4) show that as increases the number of applications simultaneously running in the WSANs, in both algorithms the **network lifetime** values are reduced. From Figure 4, it is possible to observe that by increasing the number of applications simultaneously running in the WSAN, there is naturally an increase in the possibility of finding common sensing unit among them. CAMAW algorithms well utilizes this idea to reduce energy consumption of nodes by executing the collected common data only once and sharing the result among all applications so as to further improve the use of the limited node resources. Beside that, instead of transmitting the same data several times (each one for one of the applications), as SCCH [11] would do, CAMAW transmits this data only once for the several sharing applications. The existence of common sensing units is not properly addressed by SCCH and then it will consume system energy in a less efficient way by repeatedly performing the data collection. At the end of the experiments, the remaining energy of nodes was collected to calculate the **standard deviation** about energy consumption.

TABLE I. STANDARD DEVIATION OF THE ENERGY CONSUMPTION OF THE NODES

	CAMAW	SCCH	LEACH
1 Application	1.5%	3.6%	8.5%
2 Applications	2.3%	4.1%	8.9%
3 Applications	2.9%	4.9%	11.2%
5 Applications	3.8%	5.6%	14.3%
10 Applications	5.4%	9.1%	15.7%

The results shown in table 1 indicate that with fewer applications only a small part of the sensing field was clustered resulting in a low standard deviation. As the number of network applications has increased and new areas in the network became clustered, it results in a higher standard deviation. Considering the **memory consumption** in bytes for the sensor, we noticed that the memory consumption of CAMAW (2876 bytes) was 37.4% higher than LEACH (1841 bytes). Although CAMAW consumes more memory than LEACH and SCCH, CAMAW extends network lifetime.

D. Comparison between simulated and real nodes

In this section, the same scenario simulated using Solarium was implemented on a real sensor WSAN platform. Our goal was to confirm that the results obtained from simulations actually reproduce the results that would be returned if all experiments were performed on a real WSAN platform. This real experiment was performed in a controlled environment (our research laboratory at UFRJ). In this case, the nodes were kept stationary and disposed on the floor. The experiment on simulated nodes consumed less energy than the real experiment, since there was no interference on the simulated environment. In order to compare the results of real and simulated experiments, we have used 0,5 J as initial node energy in the experiments. The maximum difference in our tests was 2% between real and simulated nodes.

V. CONCLUSIONS

In this paper, we have presented an application aware clustering algorithm for multiple networks in a WSAN called CAMAW. The results of our experiments show that CAMAW increased the network lifetime of the experimented scenarios. These results were achieved by sharing the monitoring interfaces with several applications, avoiding unnecessary data collections and transmissions. As future work in this context of network level virtualization, we intend to develop the multi-sink capability. We expect to improve the connectivity and efficiency, since it will enable CAMAW both to choose deliver the data through the less costly sink node, thus spending less energy, and/or to work with more sinks at same time. Also, this will enable CAMAW to interconnect among VSNs.

ACKNOWLEDGMENT

This work is partly supported by the Brazilian Funding Agencies CNPq and FAPERJ under grants numbers FAPERJ - E-26/110.468/2012; CNPq - 307378/2014-4; CNPq 304941/2012-3; CNPq - 473851/2012-1; INMETRO - PRONAMETRO; CNPq 477223/2012.

REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” Comput. Networks, vol. 38, no. 4, 2002, pp. 393–422.

[2] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, “SenShare: Transforming sensor networks into multi-application sensing infrastructures,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 7158 LNCS, 2012, pp. 65–81.

[3] Farias, C. et al. “A control and decision system for smart buildings using wireless sensor and actuator networks”. Transactions on

 ISBN: 978-1-61208-439-8

Emerging Telecommunications Technologies, 25(1), 2014, pp. 120-135.

[4] A. P., Jayasumana, Q., Han, and T. H. Illangasekare, “Virtual Sensor Networks a Resource Efficient Approach for Concurrent Applications,” Proc. 4th Int’l. Conf. Info. Tech., 2007, Las Vegas, NV, 2007, pp. 111–15.

[5] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, P. Polakos, A. Dhabi, and U. A. Emirates, “Wireless Sensor Network Virtualization : Early Architecture and Research,” no. June, 2015, pp. 23–25.

[6] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-Efficient Communication Protocol for Wireless Microsensor Networks,” Syst. Sci. 2000. Proc. 33rd Annu. Hawaii Int. Conf., 2000, p. 10.

[7] K. a. Bispo, N. S. Rosa, and P. R. F. Cunha, “A semantic solution for saving energy in wireless sensor networks,” Proc. - IEEE Symp. Comput. Commun., 2012, pp. 492–499.

[8] Z. Khalid, N. Fisal, H. Safdar, R. Ullah, and W. Maqbool, “Middleware Framework for Network Virtualization in SHAAL,” IEEE Symp. Comput. Ind. Appl., 2014, pp. 175–179.

[9] G. Caldas, C. M. de Farias, L. Pirmez and F. C. Delicato , “S-LEACH: A LEACH extension for Shared Sensor Networks”, Wireless Networks (ICWN), 2015 International Conference on, July 2015.

[10] C. Farias. et al., "Multisensor data fusion in Shared Sensor and Actuator Networks," Information Fusion (FUSION), 2014 17th International Conference on , 2014, pp.1-8.

[11] D. Izadi, J. Abawajy, and S. Ghanavati, "An Alternative Clustering Scheme in WSN," Sensors Journal, IEEE , vol.15, no.7, 2015, pp.4148-4155.

[12] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, “Collection tree protocol”. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems,SenSys ,SenSys '09, Berkeley, USA, 2009, pp. 1–14.

[13] E. Wilde, D. Guinard and V. Trifa. Architecting a Mashable Open World Wide Web of Things, Institute for Pervasive Computing, ETH Zürich, Zürich, Switzerland, No. 663, 2010.

[14] V. Raghunathan, C. Schurgers, S. P. S. Park, and M. B. Srivastava, “Energy-aware wireless microsensor networks,” IEEE Signal Process. Mag., vol. 19, no. 2, , 2002, pp. 40–50.

[15] S. Xiong, J. Li, M. Li, J. Wang and Y. Liu, "Multiple Task Scheduling for Low-Duty-Cycled Wireless Sensor Networks, " in INFOCOM '11, 2011, pp. 1323-1331.

 ISBN: 978-1-61208-439-8