A Framework for Application Centric Wireless Sensor Network Management

N. Parameswaran†, S. Srivathsan, S.S. Iyengar

***Abstract*—We propose a wireless sensor network management framework where we consider the management activity as a meta level thread of computation that monitors the lower (object) level protocol related thread of computations and performs management actions to affect the lower level computations. While the protocols perform the object level computations, the management level computations are performed by a set of rules. In this framework, the wireless network management can not only be tuned for specific applications, but also can be driven by flexible management policies, which can be coded explicitly in the implementation.**

***Index Terms*—Wireless Sensor Networks, Network Management, Finite State Machines.**

I. Introduction

A typical communication network requires clients as well as relay nodes, links among the nodes, protocols for communication, and information for relaying. A set of hardware and software is necessary for reliable, secure and efficient transport of information along the network nodes from a source to a destination (could be broadcast, multicast etc). The protocols themselves are not capable enough to support the normal functioning of intended behaviour. Network Management tools are necessary for monitoring, maintenance and employing (proactive and reactive) strategies in the face of an (impending) misbehaviour apart from provisioning, configuration, memory management, etc. addressing the following issues:

- Need to restore the functioning of the network in the face of unexpected events.
- Ensure graceful degradation of performance of the network (multiple failures of nodes / links / hostile jamming, etc).
- Autonomous agents must collaborate effectively for optimal performance in terms of many QoS parameters (latency, reliability, throughput, congestion control, admission control, etc).
- Collect useful data to make informed decisions in the future. Data can be used for statistical purposes (AI learning, Bayesian Belief networks, etc.) for making intelligent decisions with limited and uncertain data.

This work was supported by the PKSFI grant (No. 32-4233-40860-LSU) "Center for Excellence in Integrated Smart Sensor Surveillance Systems", July 2007-June 2012.

† Corresponding Author is from School of Computer Science and Engineering, University of New South Wales, Sydney. paramesh@cse.unsw.edu.au

S.Srivathsan and S.S. Iyengar are from Computer Science Department, Louisiana State University, Baton Rouge, Louisiana, USA 70803. ssrini1@csc.lsu.edu, iyengar@csc.lsu.edu

Section II presents a brief overview of related work. In Section III we discuss the WSN management issues and in Section IV we present our light weight management framework. We present a case study in Section V where we model the S-MAC protocol from managment perspective and show how application specific management rules can be written. The paper concludes in Section VI .

Contribution: In this paper, we propose a light weight management framework for wireless sensor network, where we monitor the behaviour of the protocol computation performed in each node, and propose actions whenever the computation reaches a predefined state requiring corrective actions.

II. Motivation and Related Work

The standard SNMP [1] protocol traditionally used for network management is not directly suitable in WSN management applications due to excessive communication overhead, need for excessively large data storage, lack of support for the extremely fragile nature of the sensor nodes, etc. MANNA[2] provides a policy based framework collecting data from the WSN and then proposing actions for management, and yet can not be employed when light weight solutions are required. The Ad Hoc Network Management Protocol (ANMP [3]) is suitable to hierarchical clustering of nodes and it attempts to reduce the number of messages exchanged between the manager and the agents. In [4] Campbell, et al. describe a goal-directed, policy-based approach to sensor network management where they have studied their techniques in wind farm domain. This is based on the ACCENT policy system [5], a more generic framework. In [6] Yuan, et al. propose a lightweight SNMS (L-SNMS) based on an RPC mechanism which addresses WSN issues such as restricted memory and low bandwidth. Deb et al. in [7] describe a distributed parameterized algorithm for sensor topology retrieval which makes a trade-off between topology delails and resources expended. The authors claim that the "topology determined at different resolutions is sufficient for approximating different network properties".

III. Network Management in WSN

Network management is characterized by monitoring network status, inferring faults and network parameters, and carrying out actions to remove the faults and improve the performance. The traditional SNMP collects information from a set of elements that are needed to be managed, processes the data, and presents them to the network management users.

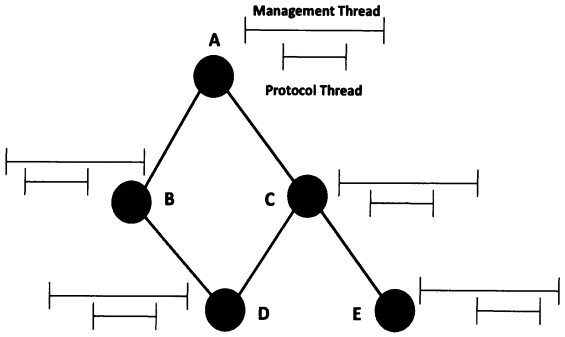


Fig. 1. A simple sensor network showing timelines at each node. The shorter interval corresponds to protcol (object level) computation, and the longer interval corresponds to the management (meta level) computation.

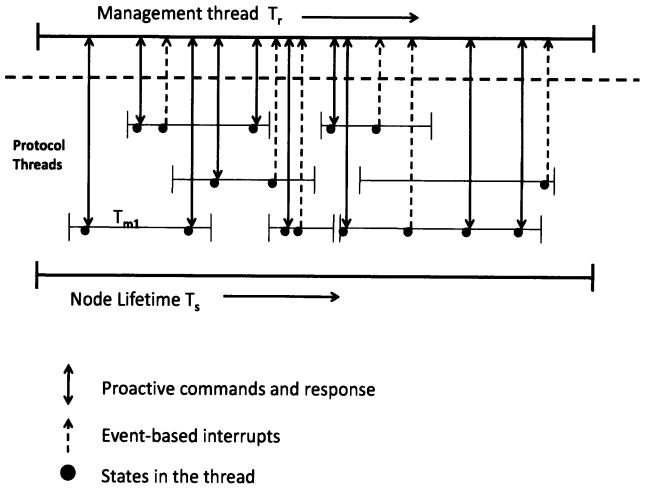


Fig. 2. Interaction between management rule base R and the object level fsm's.

Wireless sensor nodes have limited power, low transmission range, and are highly unreliable, and they have a need to work in hostile environments for long periods of time unsupervised. Due to constraints such as this, WSN management offers totally different types of challenges that are not ordinarily present in the traditional computer network management. For example, WSN management must respect limitations such as there can not be too many messages passed, long paths for transmissions can not be established for too long, the nodes can not hold much data, the nodes can randomly fail irreparably, etc.

Further, since the power source in each node is very limited, before taking up any application, the management component must assess the potential of the network as to whether it meets the requirements expected by the application. For example, in addition to power availability, one also needs to know if a node is reliable enough to sustain itself for the required period of the application, and whether current environmental conditions will permit a reliable operation of the network. The coverage area can be dynamically changing affecting the application goals. In larger applications, the nodes might have been densely populated making the network ad hoc with random fragile topology. In view of this, sensors will have to cooperate with each other to improve reliability, reduce power consumption, and improve overall performance.

IV. Lightweight Management Framework

In our approach, we decompose the computations occurring in a sensor node into two levels: the meta level and the object level. The management related computations take place at the meta level while the protocol related computations take place at the object level. We visualize the computations as timelines, and a point in a timeline corresponds to the state of computation at that point in time. For simplicity, we will use (a set of) rules R to model the meta level computation and (a set of) finite state machines (fsm's) $M_1, ..., M_n$ to model the object level computations. We use an fsm to model the dynamic behaviour of a protocol, where some of the intermediate states of the protocol form the states of fsm. The number of states and the transitions of the fsm's will depend on the application. By carefully choosing the states, we can monitor the behaviour of the protocol by reading its states, and working out the next state transitions. Let T_r be the timeline for the rules executing at the meta level, and T_{mi} be the timeline for the object level fsm M_i. If T_s corresponds to the sensor overall life time, then T_r and T_{mi} are all sub intervals of T_s. It is particularly important to realize that this is so, since in the context of wireless sensor nodes, T_s itself is very small and unpredictable, of the order of minutes to days.

Fig.1 above shows a simple sensor network where at each node we show the timelines (the shorter line interval) associated with the object level computations. (The longer interval denotes the life time of the sensor node.) Each timeline signifies one thread of computation at the object level. (We assume that time flows from left to right.)

In Fig. 2 above, we show the management timeline (referred to as the management thread resulting due to the execution of the rules R), several object level timelines (due to the execution of fsm's), and the interaction between them. The line labelled as Node Lifetime shows the over all life time of the node, which is a finite time interval. Every activity in the node should essentially happen within a subinterval of this interval. The bi-directional arrows denote probing and getting data from the object level timelines. Unidirectional arrows (from object level timelines to management line) denote interrupts due to exceptions encountered.

We have shown the timeline corresponding to the management as the longest timeline in Fig. 2, and this can be viewed as a greedy management strategy. That is, even after every other computation has terminated itself, the management component is still alive until the node dies out due to the finite nature of the battery lifetime. (It is up to the management strategy to decide as to how far the management activity should be alive in a node, and for the duration it exists, whether it should exist intermittently, etc).

Fig. 3 shows the states of an fsm where we have classified the states into normal states and exception states. The normal states refer to states of computations required as part of regular protocol computations, and exception states refer to states

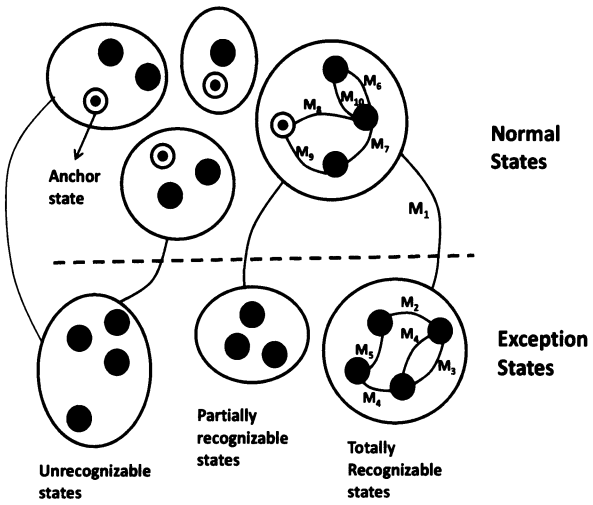


Fig. 3. Exception (error) states in protocol computation. M_i denotes management action.

that need the attention of the management thread requiring management actions to be invoked. We can classify exception states as follows.

a) Totally recognizable states: During its computation, a protocol may derail into a state which is completely readable to the management thread, in the sense that all attributes of these states will be accessible to the manage thread.

b) Partially recognizable states: These are states which are not completely known, and thus some attributes of these states are not immediately accessible or calculable for the management thread. For example, if we have defined a state to include whether the sensor data has been read successfully, and if the sensor hardware component becomes faulty, then the current program execution state may not be completely defined. In such situations, we refer to the resulting state as partially recognizable.

c) Totally unrecognizable states: These are states which are not even partially recognizable and thus can not be reasoned with. In any sensor deployment, we need to make sure that the chances of nodes slipping into this class of states is minimal.

A. Management Activity

We classify management activity into two types: proactive, and reactive. In the proactive type, the management thread senses the state of the protocol computation, and proposes actions. Such actions will, for example, include tuning the protocol for the current environmental conditions, modify sleeptimes, etc. In the reactive type the management responds to an exceptional event that has occurred in the object level protocol timeline.

Fig. 4 shows an example where the normal states are distributed in the protocol timeline L_p, and the probes from the management timeline L_m are aimed at those states. Since management activity is likely to be an overhead in an already resource limited sensor node, these states where probes can be aimed at must be selected carefully. What exactly these states

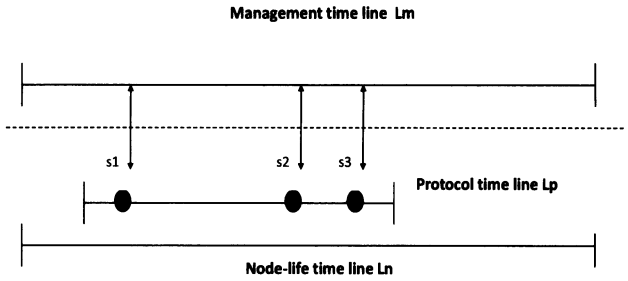


Fig. 4. State distribution in the protocol timelines.

should signify and how exactly they should be distributed on the timeline depends on the application scenario. However, it is obvious that when there are many states entertaining the probes, management will be more effective, however with larger overhead. With lesser states on L_p, the management overhead will be less, but the management activity itself may be inadequate. (We will discuss a few state distribution strategies later.)

*1) **Reactive Management:*** In the reactive type of management, whenever the protocol runs into exception states, it generates interrupts to the management thread. While the proactive management may be viewed as optional, recovering from exception states is mandatory. Accordingly, the management rules R are classified into two types: reactive rules R_r, and proactive rules R_a. The reactive rules are invoked by the protocol thread at a point in the protocol timeline L_p by an interrupt from the protocol. The management thread responds to the interrupt by invoking its reactive component R_r, examines the current state of execution of the protocol where the exception occurred, and invokes an appropriate management action to rescue the protocol from the exception state to a normal state.

1) **Rescuing from exception states:** We associate a cost with every exception state. Obviously the cost for the totally unrecognizable state is ∞ when the strategy is not to make any attempt to rescue the protocol from a totally unrecognizable state. This is because, since the state is unrecognizable, none of the rules in the management thread will be able to fire and no management actions will be taken leaving the fsm in its current state for ever. This for example can happen when there is a bug in the protocol code, and the resulting state of computation becomes unrecognizable for the rules. However, attempt should be made to rescue the protocol from all other states. When the state is totally recognizable, the management thread rescues the protocol by taking it to the normal states through an optimal or shortest path.
2) **Rescuing from partially recognizable states:** When the state is only partial recognizable, rescuing the protocol becomes somewhat tricky. Let a be the last action the protocol performed, and let s_1 be the previous state and s_2 the current state. The management will request the protocol to undo the last action a that it performed. Undoing action a will thus take the protocol from s_2 to s_1. The management thread repairs the resource so that

S.No	State	Type	Sample Action	Protocol
1	sleep	proact	set duration	S-MAC
2	periodic synchronization	proact	change period of synchronization	S-MAC
3	schedule exchange	proact	change schedule dynamically	S-MAC
4	no message received	react	set listening period longer, alert operator	S-MAC
5	contention state	proact	increase priority of the node that has been contending for a long time	S-MAC
6	wait for RTS/CTS	proact	alter wait time	S-MAC
7	message length is too long	proact	alter message length	S-MAC
8	carrier-sense delay is too long	proact	alter delay	S-MAC
9	set-parameter state	proact	alter backoff delay, processing delay, queueing delay	S-MAC
10	low energy	react	release slots	TDMA
11	node is faulty	react	release slots	TDMA
12	invalid data	react	try again	General
13	no response from neighbor	react	try another neighbor	General
14	hidden node interference	react	send message to the hidden node to goto sleep	General
15	exposed node problem	react	tell one neighbor to goto sleep	General

TABLE I
STATES AND POSSIBLE MANAGEMENT ACTIONS.

the protocol will transition from s_1 to q, another normal state. (Note that it may happen that not all actions will be undoable. In such cases, the exception state will remain unrepaired.)

*2) **Proactive Management**:* In this type of management, probes from the management thread are sent to the object level timelines to sense the state parameters, and then management commands are sent (as shown by the bidirectional arrows) to improve the performance of the protocol.

1) **Selection of states** For a given protocol and a sensor node deployment, we may be able to choose some proactive states (that is, normal states) while the exception states occur on their own. We now discuss the selection and distribution strategies of proactive states.
2) **Selection and Distribution of states in proactive management:** Proactive states are chosen by considering how the protocol performance can be improved when the environment changes. For example, in response to a new application, a better QoS may be requested by the application program. This may be achieved by dynamically altering certain parameters of the protocol. In this case we even have the luxury of distributing the states on the protocol timeline L_p. For the sake of minimizing the overhead, the management thread R may not be able to interact with object level fsm at all of its states. We thus permit only a few selected states to be visible on the protocol line L_p. These states are the ones that are easy to access and set the protocol program variables so that the performance improves for the changed environment. We can choose the distribution of these states in different ways.
   a) **One state per iteration for each loop:** In this strategy, the management thread R tracks the protocol for each iteration at least once. This also means that we should make sure that the fsm model of the protocol do not hide the iterations while we create an fsm abstraction of the protocol. Use of this strategy will flood the protocol timeline L_p with too many states, thus increasing the management overhead as the management rules R will be triggered for each iteration to take possible management actions for each state. While this may be desirable in certain applications, it may not be the best option always. Note however that if the management thread chooses to monitor only a few occurrences of (some) states, we may be able to control this overhead. (We discuss this strategy later in Section V.).
   b) **Monotonically decreasing density of states:** In this strategy, we attempt to populate the protocol timeline L_p with states such that the density of states decreases monotonically as we head towards the end of the life time of the sensor node. This means, while we actively engage in management activity when there is sufficient power in the battery, we decrease the management activity restricting it to some chosen states only as we approach the end of the life of the sensor. This for example could mean that to start with we monitor each iteration, but as the battery runs out, we skip many iterations but only monitor a few chosen ones.
   c) **QoS demand driven state distribution:** The ideal distribution of states appropriate for a particular application will depend upon how the environmental changes affect the working of the node in that application. There are situations where intense monitoring may be required, where as in other situations practically all monitoring may be suspended. This kind of behaviour requires predefined controls implemented already in the protocol such that state distributions themselves may be controlled upon receiving triggers from the management rules R.

B. Management Policies

We can represent a management policy by a set of rules, where each rule infers either an intermediate fact or an action. What is special about these rules is that they incorporate time intervals in their conditions (that occur on the left hand side of the rule) and in their facts and actions (that occur on the right hand side of the rules). For example,

(!ack-received $[t_1\ ,\ t_2]$) $\longrightarrow$ sleep$[now,\ t_3]$)

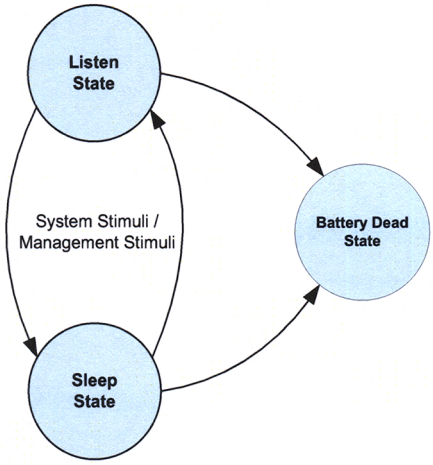


Fig. 5. Typical life cycle of a wireless sensor node.

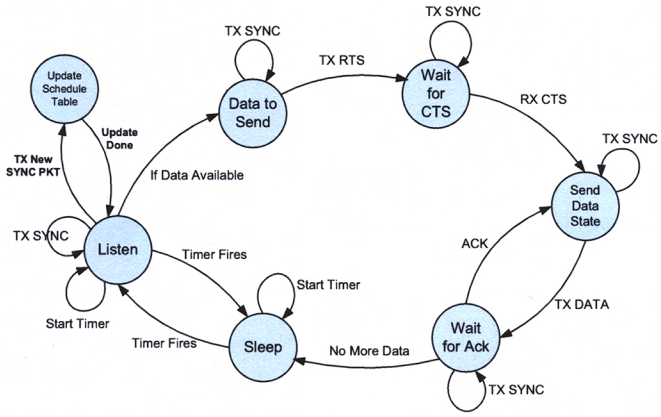
is a rule which means if an acknowledgement was not received over the time interval from time t_1 to t_2, then the node must go to sleep from now until time instant t_3 arrives. Typical management actions include reset, stop, suspend, resume, abort, manipulate constraints and exception handlers, set priorities, check if the node is part of any collective behaviour, etc. Thus, a formal description of a policy would require us define the following parameters: definition of what we mean by state of the node and of the network, what is an exception, and how the states are classified (what ontology was used, how appropriate the ontology was for the chosen domain), and management actions. Given these formal entities, a policy can then be defined in terms a set of rules called the reactive and the proactive components of the management. These rules use temporal intervals to specify the time duration over which conditions are expected to hold, the relationship between the time intervals, and the intervals actions have to take place.

C. *Example - States from S-MAC and TDMA protocols*

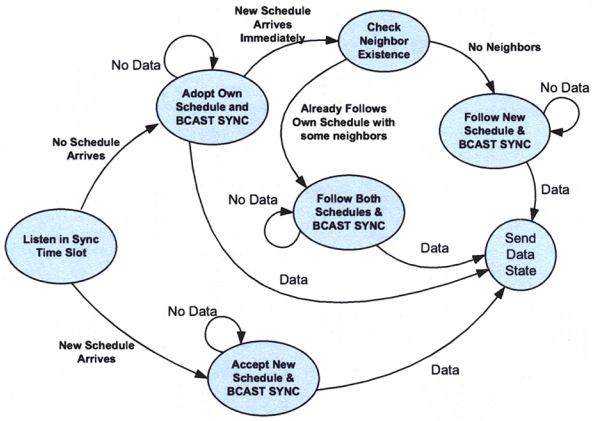
In the Table 1, we have identified several states from S-MAC [8] and TDMA [9] protocols, classified them, and suggested possible proactive or reactive management actions. For example, sleep state occurring in the S-MAC protcol can be sensed by the proactive management rules, and its duration can be set to a different value appropriate for the current situation. Similarly, in the no-message-received state, the protocol raises an interrupt, and the reactive component of the protocol might choose to set the listening period longer.

V. Case Study

While selecting states for monitoring, we need to make sure that the definition of states capture the values of the important parameters that we want to monitor and that the they remain manipulable for a reasonable interval of time. We identify two contexts to be considered: the computational and temporal. The computational context captures information that are relevant in the underlying algorithm while the temporal context captures the point in the time line at which the state



(a) State diagram for data transmission in S-MAC.



(b) State diagram for schedule exchange in S-MAC.

Fig. 6. State diagrams for (a) Data transmission in S-MAC and (b) schedule exchange in S-MAC.

occurs. For management in wireless sensor nodes with limited lifetime, temporal context is necessary. In this case study, we model the S-MAC protocol from the management view point. Fig. 5 shows the states of a typical sensor node.

The S-MAC protocol is executed in the sensor nodes by transitioning from the listen/transmit state to the sleep state, and vice versa. An abstract state diagram for the S-MAC protocol is shown in Fig. 6(a) and Fig. 6(b). In order to propose management rules, we need to interpret the S-MAC state transitions in Fig 6 in the computational as well as in the temporal contexts. In order to provide a temporal context, we use the battery consumption model as shown in Fig. 7 and Fig. 8.

We can now propose a simple management strategy which reasons with the state transition behaviour of the S-MAC state machine in the temporal context defined by Fig. 7 and Fig. 8. Before we write our management rules, let us introduce the notion of situation k where k is defined as a sequence of states as they occur on the time line. Thus, for example, if $s_1, s_2, ...s_{m-1}, s_m, s_{m+1}...s_n$ is a sequence of states on the time line, then $k = s_1, s_2...s_{m-1}, s_m, s_{m+1}, ...s_n$ is a situation where s_m is the current state, and $s_1, s_2...s_{m-1}$ are the states

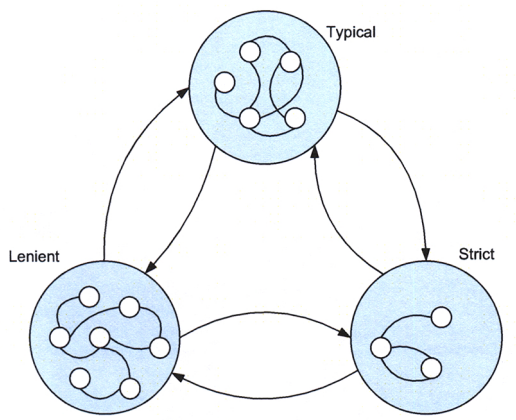


Fig. 7. Management strategy in different regions of battery power consumption.

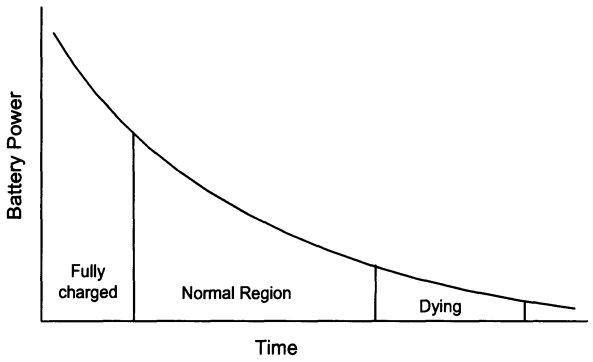


Fig. 8. Assumed battery model of a sensor node.

that occurred in the past, and $s_{m+1}...s_n$ are the states that may possibly occur in the foreseeable future. A management policy can restrict the number of states it wants to examine in the past and in the future before proposing an action. In Fig. 8, we divide the battery lifetime into three segments which we have named as *fully charged region, normal region,* and *dying region*. Accordingly, the management policy can be lenient, normal, and strict, respectively. In the lenient region, we choose a greedy strategy where we attempt to transmit more data making sure data reaches safely (waiting for acknowledgements from the recipients). This is also the region where we would want to perform expensive activities such as building neighbourhood tables, address assignment, cluster formation, etc. In the normal region, we focus on performing data aggregation, distributed data storage, etc. In the strict region, since the node lifetime is closing in, we may want the nodes to perform minimal transmissions, and try to retain the available data sensed from the environment as long as they could and spend most of their time in sleep mode. Once we decide upon a policy such as this one, we can express them in the form of rules as given below.

Problem sending data

1) If (battery is in the fully charged region) then set the value of time-out for ACK high.

2) If (battery is in the safe region) then sample data as required by the application and transmit.

3) If (battery is in critical region and ack missing too often in data transmission) then sample data at a rate as required by the application and transmit

Listen State

1) If (sensor has been listening for too long and battery is in critical region) then increase sleep time.

In the above, we have used an informal notation to specify the policy. These rules can in fact be written more formally using the standard production systems programming paradigm with temporal aspects coded in them. We may modify or add more rules to the above to implement a different type of policy.

VI. Conclusion

In this paper we have proposed a management framework that can be employed for the management of wireless sensor nodes. In this framework, the management thread monitors the object level thread of computations by probing its predefined accessible states, and executes management actions on the object level thread. Further, the management activity are proactive as well as reactive. Since we model the behaviours on the time lines exhaustively, the global context at any point in the over all computation becomes more explicit, thus making it possible to propose a management action with relative ease. The framework has yet to be extended for management of clusters, and even larger networks. We have only discussed single node management, but the idea can be extended to multiple nodes as well using distributed rule based systems.

References

[1] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.

[2] L. Ruiz, J. Nogueira, and A. Loureiro, "Manna: a management architecture for wireless sensor networks," *Communications Magazine, IEEE*, vol. 41, no. 2, pp. 116–125, Feb 2003.

[3] W. Chen, N. Jain, and S. Singh, "Anmp: Ad hoc network network management protocol," *IEEE Journal Sel. Areas Comm*, vol. 17, pp. 1506–31, 1999.

[4] G. Campbell and K. Turner, "Goals and policies for sensor network management," *Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on*, pp. 354–359, Aug. 2008.

[5] K. Turner, "Accent (advanced component control enhancing network technologies)," http://www.cs.stir.ac.uk/accent/.

[6] F. Yuan, W.-Z. Song, N. Peterson, Y. Peng, L. Wang, B. Shirazit, and R. LaHusen, "A lightweight sensor network management system design," *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pp. 288–293, March 2008.

[7] B. Deb, S. Bhatnagar, and B. Nath, "Multi-resolution state retrieval in sensor networks," *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pp. 19–29, May 2003.

[8] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1567–1576 vol.3, 2002.

[9] I. Rubin, "Access-control disciplines for multi-access communication channels: Reservation and tdma schemes," *Information Theory, IEEE Transactions on*, vol. 25, no. 5, pp. 516–536, Sep 1979.