

A Group Programming Architecture for Multi-Functionality Wireless Sensor Networks

Vartika Bhandari

Dept. of Computer Science, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
vbhandar@uiuc.edu

Loren J. Rittle

Pervasive Platforms and Architectures Lab
Motorola Labs
Schaumburg, IL
rittle@labs.mot.com

Abstract—As networks of wireless sensor/actuator nodes become an integral part of the environment, efficient post-deployment management of these networks shall pose new challenges. Numerous interesting problems, and possibilities, will arise in this context. The difficulty of physically accessing the individual devices after deployment shall drive the need for solutions that allow for efficient remote re-programming when new functionality is required of the network. This work targets prospective general-purpose sensor network deployments in enterprise environments, that would allow for asset-tracking, environment-monitoring, intrusion-detection, and a plethora of other useful functionality to be performed concurrently. It is desirable to have a common framework within which multiple functionality may be programmed into and maintained in these networks, in an efficient and fault-tolerant way. This paper describes an early architectural design experience in attempting to address this issue. We adopt a semi-autonomous approach wherein much of the process is de-centralized and distributed. Drawing heavily on multicast protocols, we present a protocol and a consolidated software architecture to enable a group programming paradigm in general-purpose sensor networks. This architecture forms the basis of the Group Establishment and Management Subsystem (GEMS) used in the MuSE middleware developed at Motorola Labs.

I. INTRODUCTION

Post-deployment management of wireless sensor networks (WSNs) poses interesting challenges due to the difficulty of physically accessing the individual devices once deployed. Of particular interest is the issue of re-programming a sensor network as the required functionality varies over time. Besides, there may be scenarios where multiple tasks/applications need to simultaneously exist and be executed in the network by possibly overlapping subsets of nodes. The target scenario for this work is such a general-purpose deployment, where the deployed nodes may be used for diverse tasks in a time-varying manner. Such deployments are expected to be integral to the enterprise environments of the future. Task assignment is envisioned to be dynamic, and the target set of nodes for performing a particular task might be identified through generic criteria not tied down to value/range-restrictions on a pre-defined code-book or attribute-set. This paper describes

This work was done during Summer 2004 when the first author was an intern at Motorola Labs, Schaumburg, IL.

a software architecture and re-programming protocol targeted towards ZigBee based sensor networks, where the core computational engine at each node is a virtual machine (VM) similar to the Maté VM [1]. This architectural design forms the basis of the Group Establishment and Management Subsystem (GEMS) in the MuSE system developed at Motorola Labs [2].

II. RELATED WORK

The focus of much past work on re-programming WSNs has been on cases where the entire network is expected to perform a single task, with emphasis on reliable delivery of code to every node. This has been addressed in many ways. Trickle [3] is an algorithm that uses a polite gossip protocol to ensure reliable code delivery. PSFQ [4] is a reliable transport protocol that uses a NACK mechanism and addresses the issue of congestion caused by re-transmissions coinciding in time with fresh packets. MOAP [5] is another NACK-based approach. The latter two assume multi-segment transmission of code images and facilitate detection of a missing segment and the sending of a corresponding NACK. An interesting approach toward enabling flexible and efficient code distribution has been the Maté VM architecture for TinyOS-based sensor nodes. By allowing most programs to be expressed as short *capsules* of bytecodes, Maté reduces communication requirements while distributing new code in the network. The code distribution itself takes place through a mechanism of viral infection. Deluge [6] is another protocol based on Trickle that addresses the issue of disseminating large data objects.

None of these approaches address the issue of directing code toward a subset of nodes, or that of selecting the target set for re-programming. A notion of such diversified functionality has been discussed at an abstract level in [7] wherein *attribute hierarchies* are expected to be set-up to allow for task-differentiation. There is also a body of work on programming abstractions that has a different focus, yet is relevant to our objective of flexible re-programming. Prominent amongst these is the Hood abstraction that has been proposed in [8] for establishment and maintenance of neighborhood information based on various criteria. The concept of *abstract regions* is described in [9] as a means of defining regions based on desired characteristics, using only local communication, and sharing data within these regions.

The MuSE middleware [2] has a Group Establishment and Management Subsystem (GEMS) that is based on the architectural design described in this paper. We have also become aware of some other recent work that addresses the formation of groups within a sensor network. In [10], the authors propose a middleware service for group management that focuses on identifying and maintaining a group of nodes that will continue to provide network services, while other devices may enter sleep mode. This work does not address the issue of multi-functional networks, and efficient code distribution, when different nodes may perform different functions. More recently, [11] has discussed the challenges that emerge when a network is required to handle multiple functionality. A scoping architecture is proposed, where a scope is a group defined on the basis of some condition on attributes being met. The attributes are organized in a hierarchical name-space. The proposal also has a notion of restricted code propagation, so as to avoid flooding. The TinyCubus system [12] also addresses the issue of programming subsets of nodes with different sets of tasks. A role specification language is used to determine what role a node must fulfil, and thereafter a role-based code distribution algorithm is used to propagate code. Algorithms for generic role assignment based on a specification language are also presented in [13].

Our architecture (which has found application in MuSE [2]) has a similar focus, but addresses a specific deployment scenario comprising VM-equipped nodes. This allows us to use the VM for generic membership (role) assignment, obviating the need for a special language. We also assume and use a logical routing tree based network layer, based on a beacon announcement process (as provided in some modes of the Zig network layer). This allows for code-propagation techniques tailored to the logical tree hierarchy.

III. SYSTEM ASSUMPTIONS

This paper assumes that the core computational engine at each sensor node shall be a virtual machine (VM). The description in this paper refers to a modified version of the Bombilla VM [14], which is a variant of the Maté VM [1] developed for sensor networks. The network stack loosely conforms to the ZigBee specification, and uses the the logical-tree hierarchy of the ZigBee network layer. There is a single gateway node at which new group initiation attempts are initiated, as well as new code for existing groups is injected into the network.

a) ZigBee Overview: We briefly describe features of the ZigBee stack (primarily at the network layer) that have significantly affected our design choices. ZigBee is based on the IEEE 802.15.4 PHY and MAC specifications. The ZigBee network layer allows for a multi-hop operation mode based on establishment of a tree hierarchy, and allocating logical network addresses that reflect the tree topology. Thus, the default routing protocol is trivially reduced to following the unique path to the destination along the tree. Since the network layer is responsible for maintaining neighbor information, it

is possible to leverage ZigBee logical tree maintenance procedures for exchanging group-related meta-data, and thereby allow optimized code distribution down the logical tree at little extra overhead. We discuss this in detail in Sections V and VI.

b) Bombilla Overview: Bombilla [14] is a variant of the Maté [1] VM proposed for use on resource-constrained sensor nodes. It has a stack-based architecture, with an operand stack and a result stack. There are four different execution contexts to which bytecode capsules may be installed and run. These contexts can be interleaved at instruction-level granularity. Of these, the *Once* context runs a capsule only once, thereby facilitating one-time code execution. A viral programming methodology enables new code to propagate in the network.

IV. THE GROUP PARADIGM

The many-one nature of the interaction between subsets of sensor nodes and the gateway exhibits strong similarity with multicast interactions. We therefore draw inspiration from IP multicast mechanisms and introduce a notion of *groups*, where a group is an embodiment of a collective responsibility to perform a task simultaneously over a window of time. Membership of a group is subject to satisfiability of certain criteria. These criteria are stipulated by the gateway at present. However, there exists the possibility of in-network origination of these criteria, to enhance the system's autonomous organization capabilities. Each group has a unique identifier and is also characterized by a sequence number. A particular group, once formed, remains in existence till a new instance of the group (with a fresher sequence number) is invoked. The $(gid, seqno)$ together thus constitute a signature that in principle uniquely identify a particular instantiation/epoch of a group. In practice, *seqno* is drawn from a fixed-size circular space, leading to a remote possibility that the sequence numbers might wrap around too soon. This possibility can be made extremely small by using the comparison rule stated in RFC 1982 [15], and by having a reasonable sequence-space size.

Once a group x has been formed, it can be provisioned and subsequently re-provisioned with new code by the gateway by merely specifying the $(gid, seqno)$ signature of the target group x and allowing each node to determine its eligibility for that code independently, based on locally maintained membership status. However this still calls for flooding the entire network with code packets and is not desirable, particularly when group membership may be localized to certain parts of the network. We therefore adopt and adapt another notion from multicast viz. that of *subtree-pruning* as in [16]. However the tree structure that we refer to is the routing hierarchy imposed by the ZigBee network layer. This allows us to use tree-based procedures, with minimal additional overhead. In our mechanism, each node is responsible for maintaining aggregate group membership statistics for the subtree rooted at itself. To facilitate this, beacons used for neighborhood management are augmented with group membership information to enable nodes to communicate membership statistics within their subtree to their parent. A node that has no members in its

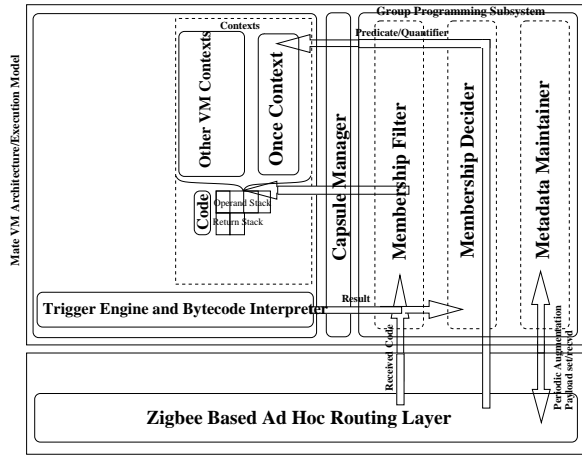


Fig. 1. Simplified View of Architecture

subtree will not propagate code during the distribution phase thereby *pruning* out the unwanted region.

The group paradigm allows for a flexible group addressing mode of operation wherein the gateway injects requirements, the network makes a distributed decision regarding the set that meets the requirements and thereafter the set is addressable succinctly via the group identity without any knowledge of individual members by the gateway.

Group Membership Criteria: Given the general-purpose nature of the envisioned deployment, it is desirable to allow generic criteria for membership. This facilitates greater flexibility in creating new membership definitions, as requirements change. We therefore leverage the availability of a VM at each node and view the criteria as being expressed in VM-executable bytecode. We consider the criterion for a particular group’s membership as being a pair (p, q) . We refer to p as the *predicate* or *qualifier* as it is a *necessary* condition that a node must satisfy (evaluate as true) in order to qualify for membership. The second component q of the pair is optional and may be used to enforce statistical requirements (e.g. approximately y % of nodes satisfying p must join the group) or other auxiliary biases as to which nodes amongst all qualifying nodes are preferred for the task. Since statistical criteria will often be specified using q , we term it the *quantifier*. Each of p and q is a bytecode capsule that executes on the local node’s VM, in the *Once* context, and returns a boolean result.

V. OVERVIEW OF ARCHITECTURE

The core computational engine at each node is a modified version of the Bombilla VM, described in Section III. The group programming functionality resides in a separate component external to the VM. The VM and the group programming subsystem interact with each other via custom interfaces exposed for the purpose. Fig. 1 is a representation of the architecture with the resident group programming component, and shows the major paths of control/data flow and interaction in the system.

A. The Group Programming Subsystem

The group programming subsystem may be viewed as having three distinct sub-components, viz., a *membership decider*, a *membership filter*, and a *meta-data maintainer*. An interface (Capsule Manager) has been inserted between the virtual machine (VM) and the programming subsystem. This interface allows the membership decider to pass on predicates/quantifiers for execution within the *Once* context of the VM, and passes the result back. It also allows the membership filter to cause installation and execution of code in various VM contexts, when it is determined that the received code is meant for one of the groups of which the node is a member. The meta-data maintainer is responsible for network-layer information needed to run the programming protocol, as well as maintenance of meta-data for optimization. An interface with the network layer allows it to be notified of relevant events such as changes to the logical tree topology. The meta-data maintainer is also capable of passing down a payload to the network layer for intended piggy-backing on a beacon. The network layer provides this functionality on a best-effort basis. However, as beacons are exchanged periodically, cyclically continuing to send meta-data allows one to overcome losses, albeit with some latency. This is generally not an issue, since after a group has been instantiated, meta-data is soft-state that is used only for optimized code propagation, i.e., to decide whether to prune or not.

B. Protocol for Task Assignment and Coordination

A two-phase protocol has been designed to enable the formation and subsequent use/re-use of groups. The first phase involves injection of the group definition into the network, and establishment of membership status at all nodes. The subsequent second phase involves use of the established group, till it is reset. This second phase can potentially involve multiple code-provisioning steps widely spaced in time. The protocol also specifies the signaling required for meta-data maintenance. The current design attempts to maintain minimal state at each sensor node so as to avoid straining memory resources. Hence group membership statistics are maintained by a node A for its subtree as a whole. At the expense of increased state, it is possible to keep more detailed statistics (for the set of subtrees rooted at the node’s children), and thereby achieve greater leeway in pruning, as well as in incorporating/providing other enhancements/guarantees. Note that since this protocol resides above the network layer and leverages network-layer mechanisms to build its own, one can also view it as an overlay for task assignment and coordination. Details of this protocol are presented in Section VI.

C. Test Implementation

An initial test implementation of major features of the system architecture was undertaken within the TinyOS (version 1.1.0) framework. The primary goal was validation of the design, and so we only compiled the code to, and tested it on the TOSSIM simulation environment [17], which allows TinyOS applications to be tested over a simulated network.

This provided early insights into the interactions between the different components in our system. We implemented a thin network layer that performs the logical tree establishment and maintenance function of ZigBee. The meta-data was piggybacked over network layer beacons in this implementation. In an actual ZigBee stack, it is likely that the network-layer may not send separate beacons, but piggyback network-layer information on MAC-layer beacons. However, this is a lower-layer detail that does not directly affect our system. Since the system only interacts directly with the network layer, this allowed us to test the major aspects of our design without implementing any other layer in the ZigBee stack. The Bombilla virtual machine [14] was modified as needed, and the set of bytecodes was augmented, to allow interfacing with the group programming subsystem. The original viral code propagation feature of Bombilla was disabled. The TinyViz interface was modified to act as a SIGNAL/CODE injection interface. The TOSSIM simulations yielded encouraging results.

VI. PROTOCOL DETAILS

The protocol assumes the existence of a ZigBee routing tree. If the protocol is initiated while the tree is still in the process of being formed, it may lead to partial programming and a later re-programming attempt (post tree formation) shall rectify the situation. We describe here the salient features of the protocol.

The formation of a group is initiated at the gateway by the injection of a SIGNAL packet containing the $(gid, seqno)$ signature of the proposed group, and the membership criteria (p, q) . On receipt of a SIGNAL packet, a node checks its own locally maintained sequence number for that gid . If the received sequence number is fresher, or if the received and local sequence numbers are equal, and the $anticipateSIGNAL(gid)$ bit is set (see Table I), it resets its complete local state (including the $anticipateSIGNAL(gid)$ bit) for that gid , de-installs any code that was running locally for that gid , and invokes the virtual machine for evaluation of (p, q) . If the result is (T, T) , the node sets itself as a member of the group, else as a non-member. Each node re-propagates a received SIGNAL packet at least once if it is not a leaf-node in the ZigBee hierarchy. Leaf-nodes do not re-broadcast. In order to avoid undesirable synchronization, random jitter is added. Duplicate-detection occurs via comparison of the received sequence number with the current local sequence number for that gid . The sequence numbers are maintained as a fixed-size circular space. Sequence number comparison proceeds as per RFC 1982 [15].

A leaf node that receives a SIGNAL and makes the membership decision is in immediate possession of $num-acceptors$ information about its subtree, which is 1 or 0 depending on what decision was. Non-leaf nodes need to obtain this information from *all* their children before sending subtree information to their parent. Each node periodically piggybacks onto network-layer beacons (which may be further piggybacked on MAC-layer beacons) a payload containing meta-data tuples of the form $(gid, seqno, num-acceptors, code-$

Data Structure	Use ($m = MAX_GROUPS$)
num-acceptors[1...m]	m-element array; stores group populations in sub-tree of node
anticipateSIGNAL	Bit-vector of size m indicating whether to anticipate a SIGNAL for the currently known instance of each group

TABLE I
SOME PER-NODE DATA STRUCTURES USED BY THE PROTOCOL

version). Each payload also includes a *new-cycle-flag* which allows a node that detects a topology change to inform its parent of the need to reset membership statistics and re-accumulate them again in a new meta-data collection cycle. We stipulate a maximum payload length thereby restricting the number of tuples per beacon payload to a number k . Nodes cycle through groups and accommodate them cyclically on the payload. However, groups for which new meta-data has just become available get priority for the first send. This allows timely propagation of meta-data about new groups up the tree. The cyclic re-transmission of meta-data helps resolve the issue of meta-data loss due to beacon-losses. This payload is received by neighbors of the node. For each received tuple, all neighbor nodes check their local sequence number for that gid and if the received sequence number is fresher, this is indication that either a SIGNAL packet with that sequence number got lost or is yet to arrive. Such nodes update their sequence number, reset their local state, de-install any code running for that gid , and set an $anticipateSIGNAL(gid)$ bit so as to avoid discarding of the SIGNAL when it first arrives (since the same sequence number is used for duplicate detection). If the received sequence number is older than the local number, the nodes have the option of unicasting an update to the sending neighbor to allow it to reset its state, else it can be assumed that the neighbor will at some point receive the payload sent by the receiving neighbor and perform an update/reset based on it. As of now, the latter approach has been adopted. A similar check performed on the *code-version* can allow for detection of stale code and a request for new code may be sent out. In addition, if the sending neighbor is a ZigBee child of the receiver, the latter checks whether it has already received meta-data regarding the current instance of this gid from this child before. If not, it adds the received $num-acceptors$ count to its local $num-acceptors$ accumulator for that group and sets a $rcvd[gid, i]$ bit to indicate that such a receipt has now occurred.

When a node has received meta-data about a group from *all* children, it is able to make decisions regarding *pruning* i.e. re-broadcast CODE only if there is a non-zero member population in its ZigBee sub-tree. When a particular group needs to be instructed to download and install a piece of code, a CODE packet containing the bytecode capsule is injected at the gateway. This packet contains the $(gid, seqno)$ of the group being targeted. If the received sequence number is older than the local one, the packet is discarded as a stale packet. If it is newer, it implies a SIGNAL was missed, and

the node resets its local state for that gid . If the sequence numbers match and the node is a group member, it checks for a fresh code version number and invokes the VM for code installation. The node also checks its meta-data. If it has complete subtree statistics for this group, it re-broadcasts the packet only if $num_acceptors(gid) - I_{\{itself_is_a_member\}} > 0$. If subtree information is incomplete, it always re-broadcasts the packet.

When a notification is received from the network layer regarding the addition of a new ZigBee child, the node updates its *validChildren* vector and sets its *propagateVector* to allow re-propagation of packets for all groups till meta-data from the new child has been received. Besides, if the node finds that for some group its current (pre-new-child-addition) *num-acceptors* count is 0 implying the possibility of a prune occurring at its parent, it unicasts a NO_PRUNE packet to its parent. Receipt of a NO_PRUNE from a child is indication that no pruning should be done for any group till complete meta-data becomes available again. Any node that receives a NO_PRUNE from a child will reset its *propagateVector* and recursively re-send the NO_PRUNE to its parent unless it finds that it has a non-zero *num-acceptors* count for each group, which precludes the possibility of a prune happening at its parent. Thus, we ensure that erroneous pruning does not happen. In fact, the node can reset its entire meta-data state and accumulate fresh meta-data all over again. When the network layer notifies of the loss of a child, less urgent action is warranted, as this might open up an opportunity for further pruning but will never require increased propagation by the affected parent. Hence after updating its *validChildren* vector, the node sets an internal flag to indicate that it must initiate a new meta-data cycle by setting the *new-cycle-flag* in the next payload to be sent. Currently the *new-cycle-flag* is set only once for each topology change. Another option is to initiate new collection cycles periodically (since we cyclically send meta-data anyway).

A number of features can be enabled in the protocol as requirements/constraints change. On receipt of a SIGNAL, when the (p, q) pair is evaluated, it is possible to associate a ternary state with each group such that $(T, T) \rightarrow$ member, $(T, F) \rightarrow$ potential-member, and $(F, F) \rightarrow$ non-member. This can allow for the *potential members* to act as a backup set and take on functionality when some member(s) start running out of battery power etc. Thus, the network can be made more fault-tolerant. Note that the receipt (non-receipt) of group meta-data from all children can also act like an implicit ACK mechanism during the group initiation phase. After injecting a SIGNAL packet, the gateway (which is the root of the ZigBee tree) expects to receive meta-data with *num-acceptors* for that group within a certain time interval. In the event of timeout without receipt of this information, the gateway can assume possible packet losses, and re-send the SIGNAL. For greater efficiency, it is possible to have this mechanism operate locally, i.e., to stipulate that a non-leaf node retain a copy of a SIGNAL packet for a certain time window after first re-propagating it, and if it does not

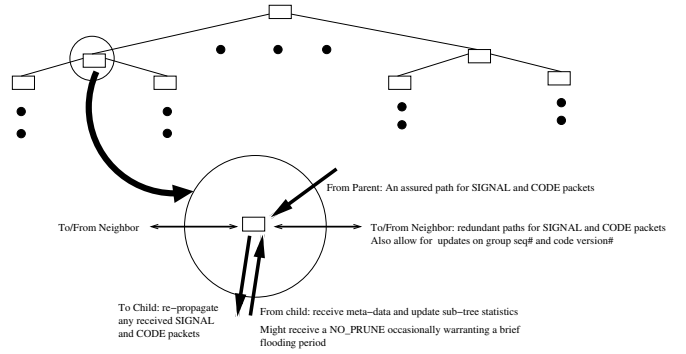


Fig. 2. Abstract Overview of Protocol Operation

receive meta-data about this group from all children within a certain timeout, it may attempt to re-transmit the packet. This would require more memory resources at each node, but would reduce communication overhead and time to recovery from packet loss. Similarly, as already discussed above, on detecting that a neighbor has a stale group sequence number, a node may choose to proactively inform it. Besides, the notion of more proactively (even periodically) initiating new meta-data cycles can be useful if current group statistics are to be used for purposes other than decisions to prune (when used only for pruning, the actual numbers are not important, so long as presence of members in a sub-tree is correctly reflected).

VII. POSSIBLE ENHANCEMENTS

The described architectural design provides a basic framework, which may be further enhanced.

While the description in Section VI outlines an implicit ACK mechanism, an explicit ACK mechanism may be incorporated for SIGNAL packets if packet loss rates are very high and absolute reliability is a major concern. More efficient code propagation may also be possible when group membership is spatially localized. In-network modification of the quantifier is possible, to attain desirable membership configurations, e.g., to obtain greater spatial localization for efficient propagation or obtain greater spatial spread for greater survivability in case a particular region suffers physical damage rendering all nodes in the region incapable of functioning. An alternative approach is multi-phase programming whereby in the initial phase a single quantifier is injected and a partial acceptor set obtained. Subsequent phases involve injection of a quantifier modified by the gateway itself based on membership statistics from earlier phases so as to incrementally enlarge the acceptor-set and thus progressively achieve the final configuration. An amalgamation of the two notions might lead to a hybrid approach wherein certain nodes in the network take on responsibility for short-term caching of the SIGNAL, and locally re-propagate it with suitable quantifier modification, if the desired target-set is not achieved in the initial phase. This hybrid approach would limit network traffic and yet allow for quick convergence.

Another issue is that of change in a node's eligibility for membership of a group e.g. if the group comprised nodes

with temperature sensors, and a node's temperature sensor fails. In the current model, once the criteria are evaluated and membership status is set, the node loses memory of what the original group definition was. This is natural as memory constraints do not allow us to cache the predicate and quantifier. The obvious way to handle such situations is to stipulate that a node that detects a change in some of its attribute(s) should conservatively assume itself disqualified from membership of all groups, and reset its complete state. Prior to that it might attempt to pass on its functionality to some neighbor(s) if possible. Another option is to periodically re-inject *group definitions* in the form of the original (p, q) pair and allow members to re-evaluate it and thereby check their continued eligibility.

VIII. DISCUSSION

The group paradigm can potentially lend itself to interesting usage scenarios. One can envision multiple pre-formed groups acting in a coordinated manner to achieve some desired functionality without explicit intervention by the gateway. Association of a ternary status with each group (as discussed in Section VI) could allow for incorporation of a backup functionality for fault-tolerant operation. The group paradigm also lends itself to such possibilities as using a notion of *suspended predicates* resident in installed code that either run periodically or on detection of stimulus condition s , possibly over a specified window of time (t_1, t_2) , and take up membership of a group x if a certain condition y is satisfied. Thereafter, the in-built mechanism takes care of efficient communication without explicit action by the nodes or the gateway. In fact, there could be a multi-way membership decision involved, wherein condition y_1 invokes group x_1 and condition y_2 invokes group x_2 . One major advantage of such a mechanism is that it would allow nodes to join a nascent group over a period of time, as and when they start satisfying the criteria for the group instead of being forced into decision at the time of SIGNAL receipt. Thus a dynamically evolving group is obtained, which might be useful in certain usage scenarios. Such notions, if carefully developed, may possibly allow for a powerful semi-autonomous organization mechanism.

IX. CONCLUSION

This paper describes an initial design effort aimed at addressing the need for a general framework to manage multi-functionality wireless sensor networks. We have proposed an approach in which different concurrent tasks in the network may be viewed in terms of (potentially overlapping) groups. We have designed a software architecture and protocol to enable remote programming of nodes, and allow distributed self-selection of the members of a group based on generic criteria expressed as VM bytecode capsules. The test implementation yielded encouraging results, and the design has found application as the basis for the Group Establishment and Management Subsystem (GEMS) in the MuSE middleware for wireless sensor networks, developed at Motorola Labs.

ACKNOWLEDGMENTS

The authors would like to thank: Venu Vasudevan for suggesting differentiation of membership definitions into a predicate (necessary condition) and a quantifier (statistical/spatial criterion), Chen Jia for further work on the implementation including the implicit ACK mechanism, and other retransmission strategies for reliable message propagation, Ralph D'Souza for providing clarifications about issues related to the ZigBee specifications, and Nitya Narasimhan for useful discussions on improvements to, and applications of this system. The authors also thank Phil Levis for releasing source code related to Maté.

REFERENCES

- [1] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, 2002.
- [2] L. J. Rittle, V. Vasudevan, N. Narasimhan, and C. Jia, "Muse: Middleware for using sensors effectively," in *Proceedings of INSS 2005*, 2005.
- [3] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the First Symposium on Network Systems Design and Implementation*, 2004.
- [4] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Psfq: a reliable transport protocol for wireless sensor networks," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. ACM Press, 2002, pp. 1–11.
- [5] T. Stathopoulos, J. Heidemann, and D. Estrin, "A remote code update mechanism for wireless sensor networks," CENS Technical Report no. 30, 2003.
- [6] J. W. Hui and D. E. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *SenSys*, 2004, pp. 81–94.
- [7] Little, Basu, Ke, and Ayyash, "Attribute hierarchies as adaptive overlays for energy efficient inquiry forwarding in sensor networks," BU Technical Report, 2003.
- [8] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: a neighborhood abstraction for sensor networks," in *Proceedings of MobiSys '04*. New York, NY, USA: ACM Press, 2004, pp. 99–110.
- [9] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *NSDI '04*, 2004, pp. 29–42.
- [10] M. S. Vieira and N. S. Rosa, "A reconfigurable group management middleware service for wireless sensor networks," in *3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing*, 2005.
- [11] J. Steffan, L. Fiege, M. Cilia, and A. Buchmann, "Towards Multi-Purpose wireless sensor networks," in *International Conference on Sensor Networks (SENET'05)*. Montreal, Canada: IEEE, Aug. 2005.
- [12] P. J. Marrón, A. Lachenmann, D. Minder, J. Hähner, R. Sauter, and K. Rothenmel, "TinyCubus: A flexible and adaptive framework for sensor networks," in *Proceedings of EWSN 2005*, January 2005, pp. 278–289.
- [13] C. Frank and K. Römer, "Algorithms for generic role assignment in wireless sensor networks," in *Proceedings of ACM SenSys 2005*, San Diego, CA, USA, Nov. 2005.
- [14] P. Levis, "Bombilla: A tiny virtual machine for tinys," Bombilla Documentation, 2002.
- [15] R. Elz and R. Bush, "Serial number arithmetic," RFC 1982, Aug. 1996, network Working Group. [Online]. Available: <http://ietf.org/rfc/rfc1982.txt>
- [16] A. Adams, J. Nicholas, and W. Siadak, "Protocol independent multi-cast - dense mode (pim-dm): Protocol specification (revised)," Internet Engineering Task Force, INTERNET DRAFT (draft-ietf-pim-dm-new-v2-05.txt), 2004.
- [17] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinys applications," in *Proceedings of the first international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 126–137.
- [18] T. Imielinski and J. C. Navas, "Gps-based geographic addressing, routing, and resource discovery," *Commun. ACM*, vol. 42, no. 4, pp. 86–92, 1999.