Design of Adaptive Overlays for Multi-scale Communication in Sensor Networks

Santashil PalChaudhuri⋆, Rajnish Kumar⋆⋆, Richard G. Baraniuk⋆⋆⋆,
and David B. Johnson⋆

⋆Rice University, Department of Computer Science, Houston, TX 77005-1892
⋆⋆Georgia Institute of Technology, College of Computing, Atlanta, GA 30332-0280
⋆⋆⋆Rice University, Department of Electrical and Computer Engineering,
Houston, TX 77005-1892

Abstract. In wireless sensor networks, energy and communication bandwidth are precious resources. Traditionally, layering has been used as a design principle for network stacks; hence routing protocols assume no knowledge of the application behavior in the sensor node. In resource-constrained sensor-nodes, there is simultaneously a need and an opportunity to optimize the protocol to match the application. In this paper, we design a network architecture that efficiently supports multi-scale communication and collaboration among sensors. The architecture complements the previously proposed Abstract Regions architecture for local communication and collaboration. We design a self-organizing hierarchical overlay that scales to a large number of sensors and enables multi-resolution collaboration. We design effective Network Programming Interfaces to simplify the development of applications on top of the architecture; these interfaces are efficiently implemented in the network layer. The overlay hierarchy can adapt to match the collaboration requirements of the application and data both temporally and spatially. We present an initial evaluation of our design under simulation to show that it leads to reduced communication overhead, thereby saving energy. We are currently building our architecture in the TinyOS environment to demonstrate its effectiveness.

1   Introduction

Sensor networks [1] consist of a large number of small, low-powered wireless nodes with limited computation, communication, and sensing abilities. Their ubiquitous, on-demand sensing capabilities have enabled numerous new applications, from vibration monitoring throughout buildings in active earthquake zones to air pollution tracking to microclimate investigations in tropical rain forests. In a battery-powered sensor network, energy and communication bandwidth are a scarce resources. Thus there is a need and opportunity to adapt the networking to match the application in order to minimize the resources consumed and extend the life of the network.

Sensor network applications have several characteristics that distinguish them from other networks (such as LANs or ad hoc wireless communication networks) and make matching the networking to the application challenging. For example, different applications demand a wide range of different communication patterns among the nodes,

including data aggregation, dissemination, attribute-based routing, local collaboration, and multiple resolution. Several networking protocols have been developed to handle these kinds of communication efficiently. However, since each involves widely different communication abstractions and trade-offs, no single protocol can be optimized for all applications. Moreover, many distributed signal processing applications demand multiple communication patterns. Finally, designing or matching protocols to applications is a very difficult task for applications developers.

In this paper, we propose an *adaptive network architecture* that matches the communication characteristics of many different applications by optimizing based on application feedback. We design a hierarchical overlay to handle aggregation, dissemination, and multiple resolution, and we leverage the Abstract Region [2] architecture to handle local collaboration between nodes. These overlays coexist and serve different purposes; together they efficiently support a wide range of different application data communication patterns. To simplify the application design, we provide a set of Network Programming Interfaces to abstract the details of low-level communication. Applications specify their communication needs through these interfaces; the architecture then uses this information to optimize the communication data flow.

Many applications, such as large-scale collaborative sensing, distributed signal processing, and data assimilation, require the sensor data to be available at multiple resolutions, or allow fidelity to be traded-off for energy efficiency. We form a self-organizing network hierarchy that can scale to very large numbers of nodes using multi-scale data communication. Our multi-scale hierarchical overlay adapts to form clusters such that data communication becomes efficient. While a self-organized hierarchy has been known to scale well, ours is the first proposal to align the network hierarchy with the application data flow.

After overviewing related work in Section 2, we describe our hierarchical overlay in Section 3. Section 4 details our initial design evaluation. We conclude and suggest directions for future work in Section 5.

2   Related Work

Various protocols and architectures have been proposed over the years for sensor networks. Earliest were the diffusion class [3] of algorithms, which are effective in aggregation and dissemination communication abstractions but cannot support multi-resolution communication required by many applications. There are various ways of implementing these diffusion algorithms depending on the application behavior, such as two-phase pull, one-phase pull, and push. The specific diffusion behavior can be chosen by the application to match its requirements [4]. GARUDA [5] is an architecture that handles reliable delivery of downstream data under various notions of reliability.

Fractional Cascading [6] and DIMENSIONS [7] have recently been proposed to handle multi-scale data communication. We extend their design, which is essentially for storage and retrieval of sensor data, to have more general applicability. Their architecture is based on a regular grid structure and assumes regular data sampling. However, practical multiscale transforms need to accommodate networks with arbitrary irregular

placement of sensors; we achieve this using our self-organizing hierarchy. Our architecture adapts to the node collaboration requirements to make the networking more efficient, which is not addressed in these two previous approaches.

SDIMS [8] is another hierarchical approach that is more targeted toward wired networks but can be adapted for wireless sensor networks. This approach provides a flexible API for configuration but does not address the adaptibility to application requirements. It does have a tunable interface for a tradeoff between latency and overhead for added flexibility.

The goal of Abstract Regions [2] is to simplify the application design by providing abstract interfaces to hide the details of low-level communication. It proposes the concept of neighborhood as a programming unit, and shows how various applications can be efficiently written using it. Many applications need multi-resolution data though, and our architecture addresses the abstraction requirement for such applications. The Abstract Region concept of neighborhood is thus complementary with our approach, and both together cover a much wider range of application requirements. Hood [9] is another approach to providing a programming abstraction, but it is also targeted toward neighborhood-based programming models only.

3   Hierarchical Overlay

We design the hierarchical overlay for efficient aggregation, dissemination, and multiple resolution of application data. In this overlay, a self-organizing hierarchical clustering is formed, inspired by the self-organization component of protocols like Safari [10] and L+ [11].

The hierarchy is a recursive organization of nodes into cells, cells into supercells, and so on, based on an autonomous self-election of a subset of the nodes into *drums*, and iteratively drums self-electing to become higher level drums, and so on. The drum is also called the *parent* for all nodes within it's cell. Figure 1 shows an example cell hierarchy. In the figure, nodes 1, 2, 3, 4, 5, and A group together to form a cell (called *fundamental cell*) with node A being the drum for the cell. Each of the drums in the network, namely nodes A, B, C, and D form a higher level cell with node C being the drum for that higher-level cell (called a *super-cell*). This hierarchy formation goes on iteratively until all the nodes come under one highest level cell. The self-selected drums aid in this hierarchy formation by sending periodic beacon packets.

Each node can be thought of as a level 0 cell and a level 0 drum (level 0 drums do not send beacon packets). Every level k drum is at the same time also a level i drum, for all $i < k$. This hierarchy formation algorithm is distributed, with no central coordination. Drums of the same level are roughly uniformly spaced, with higher level drums more sparse than lower level drums. A *coordinate* of any drum at level i is the concatenation of the coordinate of the level $i + 1$ drum with which it associates, along with a unique identifier. This unique identifier can be any random string large enough to avoid collisions. We use an address assignment technique proposed in TreeCast [12], whereby the nodes are given compact addresses minimizing the length of address strings. Various sensor applications require the identifiers of nodes along with their values, and this technique leads to efficient encoding of the identity (and location) of the nodes.
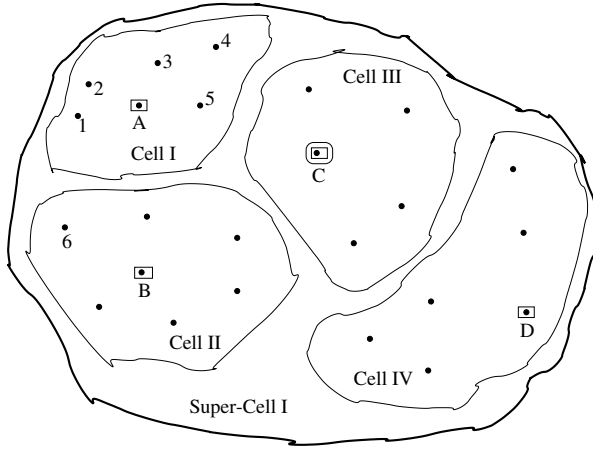
Fig. 1. Cluster hierarchy

In the initial startup period, each drum sends beacon packets (*beacons*) periodically, containing the beacon sequence number, the drum level, and a hop count, which aids in the hierarchy formation. These beacons are forwarded by all nodes within the hop count limit or those within the cell defined by the parent of the drum sending the beacon. In Figure 1, the beacons from node B reach all the nodes in super-cell I. The beacons sent during this initial phase also give the shortest path from any drum of level i to its associated higher level drums and the drums in the same level within it's super-cell.

Algorithm 1 shows the *Cluster Formation* algorithm. A drum of level i is denoted by $drum_i$, and $drum_0$ denotes the nodes. Beacons sent by $drum_i$ are denoted by $beacon_i$. $\mathcal{D}_1$ is the fundamental cell diameter and is dependent of the cluster size required by the application.

The drums wait a random time between 0 and T_{max}, before deciding whether to become a higher level drum or associate with another drum. The association scope of the drums, determined by the hop limit of the beacons, increase geometrically with the level of drum. Decreasing α increases the number of levels in the hierarchy while reducing the number of nodes in each level, whereas increasing the α has the opposite effect. Lines 8-10 of the algorithm ensure that the beacons of any drum reach all nodes within its super-cell. Lines 11-13 ensure that no drums of same level form too close to each other, as that reduces the efficiency of the clustering. Lines 14-16 make the hierarchy evolve such that the drums are always associated with closest higher level drum. The number of levels formed in the hierarchy is of $O(\log(N))$, where N is the number of nodes in the network. As a simple analysis of the cost of the algorithm, if instantaneous propagation is assumed, then all $level_i$ drums are separated by $\mathcal{D}_i$ hops, with very high probability, and so the total startup phase is of $O(T_{max} \log(N))$. Therefore the latency can be made smaller by reducing T_{max} to an optimal value.

For null *Selectors*, the hierarchy formed can be analyzed using the Random Sequential Adsorption (RSA) model, since under the instantaneous propagation approximation, the hierarchy formation conforms to the RSA model [10]. The average number of level 1 drums formed is

Algorithm 1. Cluster Formation

1: **repeat**
2:     $Drum_i$ *wait for a random time up to* T_{max}
3:     **if** $Drum_i$ does not hear a higher level drum beacon, or is not the highest level drum **then**
4:         *Steps up to* $level_{i+1}$ *and starts sending periodic beacons with hop limit of* $\mathcal{D}_{i+1} = \alpha \times \mathcal{D}_i = \alpha^i \times \mathcal{D}_1$
5:     **else**
6:         *Associates with the nearest* $drum_{i+1}$.
7:     **end if**
8:     **if** $Drum_i$ hears any non-duplicate beacon by a drum in it's super-cell **then**
9:         $Drum_i$ *forwards the beacon.*
10:     **end if**
11:     **if** $Drum_i$ hears any $beacon_i < \mathcal{D}_i$ hops away **then**
12:         *The drum with the lower id steps down to* $level_{i-1}$
13:     **end if**
14:     **if** $Drum_i$ hears any $beacon_{i+1}$, which is closer than current $beacon_{i+1}$ **then**
15:         $Drum_i$ *associates itself with the closer drum*
16:     **end if**
17: **until** All nodes are assigned stable coordinates

$$n = 0.547 \left(\frac{N}{\pi \frac{\mathcal{D}_1^2}{4} \rho} \right) \tag{1}$$

where N is the number of nodes in the network, ρ is the node density in hop-metric sense, its value depending on the transmission range and spatial density of nodes. When applied to multiple levels of the hierarchy, this gives us

$$\frac{n_i}{n_{i+1}} = \left(\frac{\mathcal{D}_{i+1}}{\mathcal{D}_i} \right)^2 \tag{2}$$

where n_i is the number of level i drums.

3.1   Adaptive Hierarchy

During startup, each drum sends beacon packets periodically, which are forwarded by all nodes based on a policy of geographical (hop count) proximity. These beacons then induce a cell hierarchy that is proximity based. In various sensor network applications, proximity is a good measure of correlation and hence of data interchange. So, the above approach of cell structure formation matches the collaboration between and nodes.

In many other applications though, the collaboration and communication take place between nodes based on various other constraints. For example, nodes with similar magnetic field or temperature readings within a neighboring scope might need to communicate more often. So, if the clustering hierarchy matches the collaborative set of nodes, communication can be efficiently abstracted and implemented. We use some application specified filters, called *Selectors*, to align this hierarchy to match the collaboration and communication sets of nodes.

Algorithm 2. Recluster ($Selectors$)

1: *Drum sends out beacons with $Selectors$*
2: *Nodes matching the $Selectors$ (re)select the Drum as their parent*
3: **if** Any node become orphan or cell is sub-optimal **then**
4:     *It sends Solicit Beacons with specified $Selectors$*
5:     *Nodes matching the $Selectors$ respond with Beacons*
6:     *Choose the drum most suitable with respect to the $Selectors$, and become a child of the drum*
7: **end if**

We define a *Selector* as a tuple of ⟨ *attribute*, *value*, *operator* ⟩, where *attribute* is any application specified variable, and *value* is a valid element from the range of the attribute. The *operator* is a binary operation (such as $>$, $<$, or $=$) with *value* being one of the operands. We extend the definition of a Selector to form:

$Selectors = Selectors \wedge Selectors \mid Selectors \vee Selectors \mid Selectors \mid Selector \mid null$

The values for the attributes are assumed to be shared between the application, sensor hardware, and the networking layer at a node, which enables the Selectors to be evaluated at the networking layer. An operator needing a time-series of previous attribute values might entail the sharing of whole data structures of application computed values. Currently only scaler operators are supported; more complex operators could have application-defined call-back functions to enable them to be evaluated by the application.

The beacons of drums are forwarded by a node if the hop count in the beacon packet is less than a specific value and the specified *Selectors* evaluates to true for the attribute values in that node. For an empty *Selectors*, the effect is to forward the beacons based only on hop count.

Reclustering of the network hierarchy to adapt it closely to the communication flow is initiated by the application locally in cells where adaptation is needed. This is best judged by the application, as the networking layer does not have any knowledge of the application logic or how the sensed values influence the communication. Algorithm 2 shows the *Recluster Algorithm*. The parameters for cluster formation are changed locally to reflect current communication patterns. The *Selectors* encode the criterion for the new cluster formation in the *Split Phase* of the algorithm. After reclustering, some nodes might become *orphans* (nodes without parent) or some cells might be smaller than optimal. These nodes or cells then merge with neighboring cells meeting the criterion in the *Merge Phase* of the algorithm. This reclustering is triggered by the application locally, and only occurs in the cells needing it for efficiency. The rest of the clusters in different areas are not changed. Hence, this reclustering takes place locally only where necessary and invokes no long range messaging.

3.2   Network Programming Interfaces

We design a set of address-free Network Programming Interfaces (NPIs) to adhere to the paradigm that communication for the typical sensor network applications should be expressed without referencing specific nodes [13]. The interest is in data over space and time, rather than individual node values. The subject of direct one-to-one communica-

tion has been extensively studied in the literature, and we will not propose any new protocol for this but will leverage the existing body of work. We provide primitives to ease the programmability in a sensor network, by capturing the interfaces that are needed by sensor applications in general. Abstract Regions [2] proposed a flexible means of node addressing, by supporting data sharing using a tuple-space-like programming model. Their approach is similar to the MPI approach for parallel machines, by hiding the details of the sharing primitives. We support similar primitives and extend them for our multi-scale architecture, although sharing is explicit using put and get primitives, to provide a more efficient implementation of the programming model. The two groups of interfaces we support are *discovery* and *communication*.

Each of the interfaces can be implemented in either blocking or non-blocking fashion. In non-blocking mode, the operation is invoked through a command, and when the operation is complete, a callback is invoked on the original requesting component. In blocking mode, the operation may block and then resume on an interrupt either by a timer or message arrival. The blocking mode is significantly easier to program, as in the non-blocking mode the programmer has to handle the synchronization and callback explicitly. TinyOS [14] supports the non-blocking concurrency model, but a lightweight thread-like abstraction called Fiber [2] has been implemented recently as a blocking model.

Figure 2 shows the *virtual* hierarchy schematic of Figure 1, which will help explain the interfaces. All the levels above level 0 are virtual, and the nodes only exist at the lowest level.
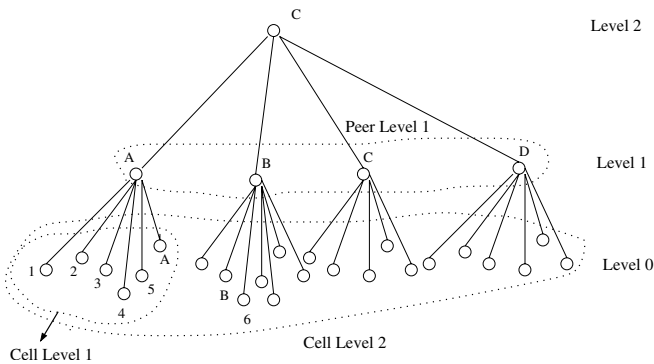

Fig. 2. Virtual hierarchy

Discovery Interfaces. Discovery interfaces can be invoked by any node to give itself information about related nodes. This information is gathered periodically, with a period as specified by the application, and the application is informed of any changes in the information. This procedure is continuous, either triggered by node failures or additions. The information might contain the identifiers of the set of nodes, their locations, link quality and number of hops to each of them, and resource (e.g., remaining battery or available sensors) present in each of them. The information learned from these discovery interfaces can be used to configure the Selectors with any specified criterion. For

example, the Selectors might be specified to filter nodes within a specified geographical distance or with higher than a specific link quality. There are three possible interfaces supported. As a node can be simultaneously in different levels, a level is specified for each interface to specify the level for which the information is required. The format for representing the gathered information is dependent on the implementation.

- *Parent Information (Level)*: Returns information about the parent of the node at the specified level. For example in Figure 2, this interface when invoked on node 6 with level 1 returns information about node B, and with level 2 returns information about node C.
- *Peer Information (Level)*: Returns information about the peers of the node at the specified level. For example in Figure 2, this interface when invoked on node B at level 1, returns information on the set of nodes A, B, C and D.
- *Cell Information (Level)*: Returns information about the cell of the node at the specified level. For example in Figure 2, this interface when invoked on node A, returns information on the set of nodes 1, 2, 3, 4, 5 and A.

Communication Interfaces. In our multi-scale architecture, we support both kinds of communication models: *Put* and *Get*. In Put, a node sends data to its cell, parent, or peers, whereas in Get, a node solicits data from its cell, parent, or peers. The Put interfaces correspond to the push paradigm, and the Get interfaces correspond to the pull paradigm that has been proposed by the diffusion type of algorithms [3]. We support both types, as different applications might be optimized using different paradigms, as pointed out by Heidemann et. al [4]. In some situations where the data generation rate is infrequent and unknown, polling using Get will be inefficient; using Put by the source of the data when the data is generated will be optimal. In another scenario, where the data generation rate is high and consumption rate is low, pushing data using Put will entail redundant data communication; using Get by the consumer of the data when the data is required is optimal in this case. The Put Interface can be implemented in multiple ways: stored locally, sent immediately to the designated scope, or cached at different intermediate locations. Similarly, the Get Interface implementation might involve either fetching remote data or local retrieval. The specific implementation depends on the application characteristics.

We also support *Reduction* interfaces that use an associative operator (such as *sum*, *max*, or *min*) to reduce an attribute across all the nodes in a specified region. This Reduction interface can be implemented using Get and Put, but efficient implementations can take advantage of local reductions while propagating the values. This abstraction also provides ease of programmability.

There are three groups of primitives a node might address: its parent, its peers, or its cell. This leads to six different interfaces for Put and Get. Reduction interfaces are done on either cells or peers. All examples below refer to Figure 2. For node A, the parent is node C; the peers at level 1 are B, C and D; and the peers at level 0 as well as the cell at level 1 are nodes 1, 2, 3, 4, and 5.

- *PutParent (Attribute, Value)*: The value of the attribute is sent to the parent node. When called on Node A, the data is sent to node C.

- *PutCell (Level, Selectors, Attribute, Value)*: Level can be at most one level higher than the node using this interface. So, a node of $level_0$ can send message to the fundamental cell. In general, a $drum_i$ can send message to all nodes in the same $level_{i+1}$ cell. For Node A, PutCell called with level 1 delivers data to nodes 1, 2, 3, 4 and 5, while called with level 2 delivers data to all the nodes marked by Cell level 2. The targeted nodes can filter the receipt of the Data using the $Selectors$.
- *PutPeer (Level, Selectors, Attribute, Value)*: The level can be at most same as the level of the node using the interface. This interface provides the same functionality as PutCell for $level_0$ nodes. For node A, level 1 delivers the data to nodes B, C and D (Peer Level 1).
- *GetParent (Attribute)*: The value of the attribute is solicated from the parent node.
- *GetCell (Level, Selectors, Attribute)*: Level can be at most one level higher than the node using this interface. In this interfaces, Data is received from the cell nodes matching the $Selectors$.
- *GetPeer (Level, Selectors, Attribute)*: The level can be at most same as the level of the node using the interface. This interface provides same functionality as GetCell for $level_0$ nodes.
- *ReduceCell (Level, Selectors, Attribute, Operator)*: This interface is applied on the attribute for all nodes in the cell specified by level and $Selectors$. And the reduced attribute value is stored locally. For example for operator *max*, the maximum attribute value within the cell is returned by this interface.
- *ReducePeer (Level, Selectors, Attribute, Operator)*: Similar interface where the scope is all the peer nodes at the specified level.

3.3   Efficient Communication Operations

The Network Programming Interface in the previous section is used by the applications to form a clustered hierarchy and to adapt it for efficient communication. In this section, we describe mechanisms for efficiently supporting the different communication interfaces. We assume the existence of bidirectional wireless links, which is true for most commonly used wireless MAC protocols.

- *With the parent node*: The drum beacons that are used to form the cluster hierarchy is utilized to route from and to parent node, by following the path or reverse path of the beacons respectively. If the path breaks, due to nodes in the path moving away or dying, then local route repair is done to find a new route. The drum whose path breaks, sends out beacons for a short interval to repair the broken path.
- *With the peer nodes:* At any level, the peer nodes need to be able to communicate with each other efficiently. This is achieved by expanding the scope of the beacons for the drums. The beacon packet of a level n drum is also forwarded by all nodes in the level $n + 1$ cell of the originating drum. The reverse path is followed to reach each peer. This is only done in the startup period or when a path breakage is detected. Multicasting at network or MAC layer (if possible) is done to prevent duplicate packets along common part of the paths. For example in Figure 1, beacon packets from node A in cell I is flooded to the whole super-cell I. And hence B, C and D know of the shortest path to A.
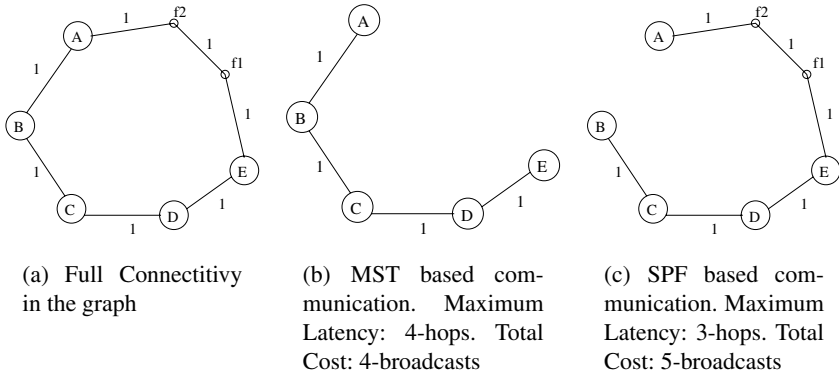
(a) Full Connectitivy in the graph

(b) MST based communication. Maximum Latency: 4-hops. Total Cost: 4-broadcasts

(c) SPF based communication. Maximum Latency: 3-hops. Total Cost: 5-broadcasts

Fig. 3. Example topology connecting peer nodes (A, B, C, D and E) and forwarding nodes (f1 and f2)

The above technique leads to minimum latency communication from any node to the rest of it's peers, using shortest path. But, it also leads to higher cost in terms of number of forwards. Alternatively a Minimum Spanning Tree can be formed between the peer nodes. This leads to lesser number of forwards, but also leads to higher maximum latency. Figure 3 shows an example topology to illustrate this. This tradeoff can be exposed for the application to choose from.

– *Within the cell*: Communication from any node to the whole cell can be achieved by simple flooding within the scope of the cell, whereby each node forwards a packet exactly once. However, this is very sub-optimal and leads to many redundant broadcasts specially in a dense network. We describe next our strategy for optimal cell flooding.

Optimal Cell Flooding. Typical broadcasting using simple flooding, where each node forwards a packet exactly once, leads to broadcast storm problems [15] and is very energy inefficient. To form a more efficient flooding algorithm, there has been substantial work with regard to carefully choosing the forwarding nodes to reduce the number of forwards without reducing it's effectiveness. Williams and Camp [16] categorized the techniques recently. In probability-based methods, nodes forward with some variable probability parameter chosen randomly or depending on the number of broadcasts heard. In area-based methods, distance or location of the nodes are taken into account by the node before deciding on forwarding. Both of these methods are completely localized without the need for any coordination. In neighbor knowledge methods, a distributed algorithm forms a Connected Dominating Set (CDS) to choose a subset of nodes to be forwarders. This has more overhead than the previous methods, but leads to more optimal flooding due to better knowledge about the neighborhood. Ideally, a Minimal Connected Dominating Set (MCDS) will give the most efficient set of nodes to forward packets such that all nodes are reached. Building a MCDS is an *NP-Complete* problem, however, and the problem gets harder in sensor networks in the absence of global

knowledge. There has been a significant body of work on approximating the MCDS using heuristics [17].

There are two types of neighbor knowledge methods proposed: *Relaying* in which a node determines the forwarding status of it's neighbors, and *Pruning* in which a node makes a local decision on it's forwarding decision. Multipoint relaying is an efficient approach for relaying, and has been used in the OLSR ad hoc network routing protocol [18]. The re-broadcasting nodes are explicitly chosen by the upstream nodes, either via "hello" packets, or within the header of each broadcast packet. In relaying, there is either additional overhead in each packet to designate the forwarding nodes, or relevant state that has to be maintained by each node. The statelessness of protocols has prime importance in an unreliable network of sensor nodes, as a stateless protocol never operates on out of date state. In the pruning methods, nodes decide on their own locally whether to forward or not, leading to better reliability in the face of failure. There is automatic correction for small changes and robustness to big changes. Nodes can go to sleep independently in pruning methods, but there needs to be additional coordination in relaying methods so that no delegated forwarder is sleeping. In the absence of MAC layer multicast for pruning, nodes have to broadcast every packet for which all the neighbors have to process the packet before knowing that they are not designated forwarders. And finally, a comparison paper [16] showed very similar performance for both of the methods. We chose to use pruning method for the above reasons.

Various approaches for pruning based broadcasting use knowledge of k-hop neighborhood information, m-hop last visited nodes information for each packet, and priority between nodes. Larger value of k leads to more optimal forwarding set, but also entails higher cost for maintaining this neighborhood information. Larger value of m is also useful, but entails packet overhead. Wu and Dai [19] have proposed a generic scheme to cover all pruning based approaches. A node determines it's forward status by finding existing replacement paths between all pairs of its k-neighbors. If all the replacement path nodes have higher priority values than itselt or is already visited, then the node chooses to be a non-forwarder. Else, it forwards.

In designing an efficient flooding algorithm, the factors we chose were the following. The computational complexity is O(k^2) for such algorithms, thereby dictating a small value of k. Also, recent proposals for sensor network MAC protocols [20, 21] maintain a 2-hop neighborhood to deal with efficient assignment of conflict-free slots. We choose value of k to be equal to 2. The information for m equal to 1 is available at no cost when the packet is received, by looking at the source. A greedy approach is taken to prioritize the nodes based on their degree of connectivity. Our solution is similar to the approach taken in the Scalable Broadcast Algorithm [22]. In a more or less static network, the neighborhood information is invariant. On detection of neighborhood change, this two-hop neighborhood is recalculated by all nodes broadcasting their neighborhood nodes. Our *Cell Flood Algorithm* is shown in Algorithm 3.

$maximum(|\mathcal{N}_X - \mathcal{N}_A| \, for \, X \in \mathcal{N}_A \bigcap \mathcal{N}_B)$ is the maximum number of additional nodes that can be covered by any node which has received a broadcast from $node_A$ and is in the neighborhood of $node_B$. Greedy approach is in choosing the broadcasting node in line 5 of the algorithm, by favoring the node with maximum additional coverage. In Line 6, $\mathcal{S}$ is updated every time the node receives a broadcast of the packet from any

---

Algorithm 3. Cell Flood Algorithm

Require: $Node_B$ receives a broadcast packet from a neighbor $node_A$
Require: $\mathcal{N}_B$ be the set of 1-hop neighbors of $node_B$.
Require: $\mathcal{S}_B$ is a set of nodes in the 1-hop neighborhood which have not yet received the packet

1: **if** $Node_B$ has handled the same broadcast packet before **then**
2:     $Node_B$ *silently drops the packet*
3: **else**
4:     $Node_B$ *calculates   a   timer   proportional   to   the   ratio*
    $\left(\frac{maximum(|\mathcal{N}_X - \mathcal{N}_A| \, for \, X \in \mathcal{N}_A \cap \mathcal{N}_B)}{|\mathcal{N}_B - \mathcal{N}_A|} \right)$
5:     *Till timer expires, $node_B$ updates $\mathcal{S}_B$ when it hears any other neighbor broadcast, using
    the neighborhood information*
6:     **if** $\mathcal{S}_B = \emptyset$ **then**
7:         $Node_B$ *silently drops the packet*
8:     **else**
9:         $Node_B$ *rebroadcasts the packet with some jitter*
10:     **end if**
11: **end if**

---

neighboring node. Elements of $\mathcal{S}$ which are present in the $\mathcal{N}$ of the neighboring node are removed. If $\mathcal{S}$ becomes empty, the packet it dropped, else the packet is forwarded.

Proof of full coverage: Every node in the network checks it's neighborhood to determine whether all neighbors have received a packet, and forwards the packet if there is any uncovered node in the neighborhood. Hence, all the neighbors of any node in the network receives the packet. As the full network is an union of the neighborhood of all the nodes, hence all the nodes in the network are covered.

Optimality of coverage: The proposed greedy heuristic leads to a approximate MCDS for the graph. Analysis of the approximation bound gives $log(n)$ times the cardinality of the optimal MCDS solution, where n is the number of nodes in the graph. The proof is similar to the optimality proof for Multipoint forwarding [23] .

Optimal Selectors Implementation. *Selectors* filter can be specified in some of the communication interfaces. If the filter is *null*, then the communication operations are efficiently implemented as elaborated previously. If this filter is not *null* but has a *Selectors*, then this scope (termed *Selectors* scope) is a subset of the scope with *null* selector (termed *null* scope). The communication operation can be done assuming *null* selector, and filtered at each node. This is not most efficient, specially is the *Selector* scope is significantly less than *null* scope. We implement the scoping of the *Get* and *Put* operations using the following technique.

   If operations with any *Selectors* filter is invoked the first time, the communication has to be delivered in the *null* scope as there is no knowledge of the location of the nodes matching the *Selectors*. But, for frequently invoked *Selectors* filters, an optimized implementation is done to cover the matching nodes only. If any particular *Selectors* if invoked frequently at any particular node, a trigger is set. The first packet

being delivered after this trigger is set includes a *DoReinforment* flag. All the nodes matching this *Selectors* filter, sends a *Reinforced* message back to the source, specifying the *Selectors*. All the nodes through which this *Reinforced* message passes back are part of the forward set of nodes for the specified *Selectors*. All the subsequent packets with this *Selectors*, has a *OnlyReinforced* flag and is forwarded by the forward set of nodes only. This is remembered for a specific interval, greater than the period specified in the communication interface, if present. Periodically the *DoReinforcement* flag is set again and forwarded by all nodes to account for any change in topology or interest.

3.4   Multi-scale Application

To demonstrate the effectiveness of the network programming interfaces, we describe a distributed wavelet compression algorithm which can effectively use the interfaces to optimize the communication. Multi-resolution data analysis, processing and compression is useful for various sensor network applications. A lot of previous work on wavelet-based processing in sensor networks have assumed regularly-spaced data.

Recently, Wagner et. al. [24] have proposed a haar wavelet based multi-scale data analysis which enables irregular wavelet transform. In the bottom-up approach of that algorithm, all the sensors in the fundamental cell sends their sensor readings to the drum for that cell using *PutParent*. The locations and identifiers of the nodes are also available to the drum through the Discovery interface. The drum calculates the *scaling coefficient* describing the average reading of the cluster and the *wavelet coefficient* encoding the deviations from the average readings. These scaling coefficients are then passed up the hierarchy using the Parent interfaces, and similarly computed. In the top-down approach of the algorithm, querying is from the top-level and drilling down until the requisite resolution of the data is obtained. This is achieved using the GetPeer interfaces. Finest resolution is obtained if query goes down until the fundamental cell level.

For any compression, maximum efficiency with minimal loss is achieved when many data points are similar enough to be represented with one data point. When compressing a field of sensor data, various regions in space might have similar readings. So, if there is one cluster for each region, the region might be efficiently represented by a single value. But, if one cluster encompasses two different regions with divergent values, compression is not so efficient. The reclustering technique is used here such that each cluster has similar values and hence efficient representation. Locally near the region boundaries clusters are aligned with the regions.

4   Initial Design Evaluation

We have performed an initial evaluation of the design of our architecture by simulating the adaptive overlay formation in the *ns-2* simulator. We are currently implementing the protocol and interfaces in TinyOS. A full evaluation of our architecture will then be possible by modifying existing and new sensor network applications to use this architecture.
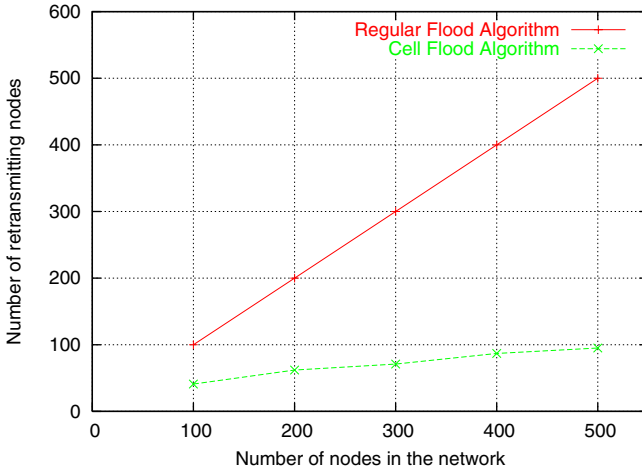
Fig. 4. Number of retransmitting nodes with increasing network size

4.1   Cluster Operations

In this section, we evaluate communication operations. In particular, we show the effectiveness of the Cell Flood Algorithm. To flood a particular cell with or without Selectors, this algorithm builds an approximate Minimum Connected Dominating Set. Thereby, the number of forwarding nodes is reduced without compromising the quality of the flood. Figure 4 shows the number of retransmitting nodes with increasing network size. The area of the network is kept constant while increasing the number of nodes, thereby increasing the node density. For a regular flooding algorithm, where each node forwards a packet exactly once, the number of forwarders is exactly equal to the number of nodes. In our Cell Flood Algorithm, the number of retransmitting nodes grows very slowly with increasing network size. The percentage of retransmitting nodes decreases with increasing network size, thereby showing good scalability.

4.2   Hierarchy Formation

In this section, we show the effectiveness of our hierarchy formation. Figure 5 shows the hierarchy formed for an example topology. 500 nodes are evenly distributed in an area of 3000 meters by 3000 meters. The radio range is taken to be 250 meters. Each part of Figure 5 shows the first and second level clusters formed, along with rays connecting each node to it's parent. Figure 5(a) shows the clustering for $\mathcal{D}_1$ equal to 1 hop, Figure 5(b) for $\mathcal{D}_1$ equal to 2 hops, and Figure 5(c) for $\mathcal{D}_1$ equal to 3 hops. As the number of hops allowed in the fundamental cell increases, the size of it increases along with decrease in the number of levels in the hierarchy.

   Figure 6 illustrates the number of drums at each level of the hierarchy as a percentage of the total number of nodes in the network. This is shown for increasing networks sizes, with the node density kept constant. As shown in the analysis in Equation 2, the number of drums in increasing levels decreases quadratically. Larger the network, lesser
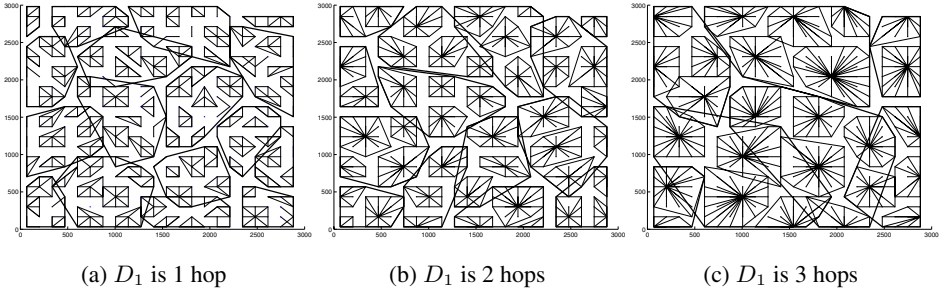
(a) D_1 is 1 hop   (b) D_1 is 2 hops   (c) D_1 is 3 hops

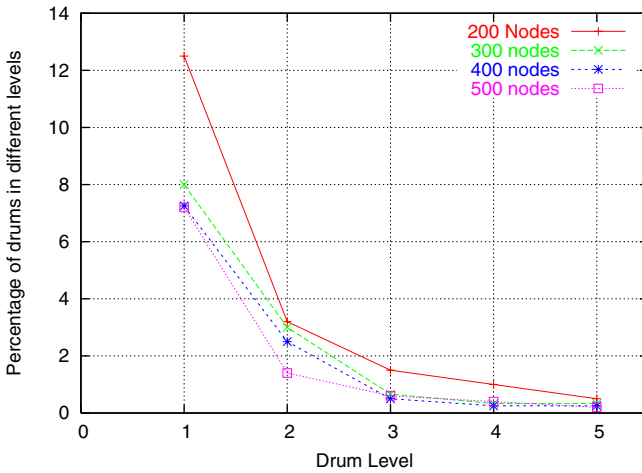**Fig. 5.** Example topology with different values of D_1

Fig. 6. Percentage of nodes which are drums are various levels

is the percentage of drums at each level. In this scenario, the value of α, which is the ratio between the beacon hop limits for consecutive drum levels, is taken as 2.

Figure 7 shows the latency for cluster formation with increasing network size. Here again, in all the networks sizes, the density is kept constant. The higher the level of drum, the longer it takes to stabilize. This is because the higher level drums are further spread apart and have larger beaconing intervals, which makes any change in higher levels propagate more slowly. The startup latency increases slowly with increasing network size. For the scenario sizes experimented with, the drum level 3 for all network sizes get stable at the same time. The cost of cluster formation arises from the beaconing during this phase, and hence is directly proportional to the length of time it takes to stabilize.

This also illustrates the effect of local change for adapting the hierarchy to communication requirements. The local clustering changes take place at a lower latency as shown in the figure. Therefore adaptivity of the clustering can be achieved with low latency, and hence also with low cost.
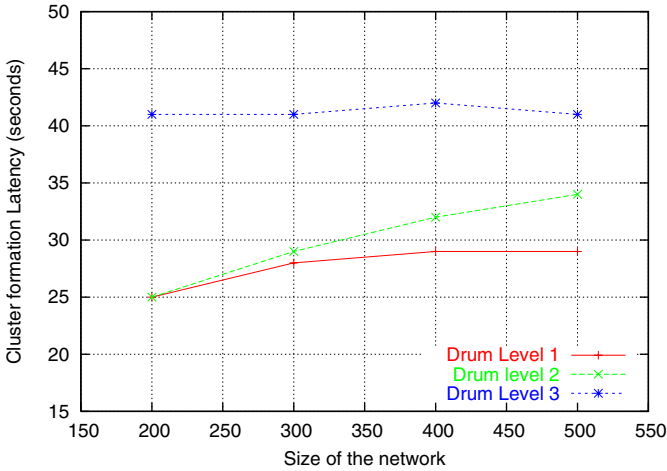
Fig. 7. Cluster formation latency with increasing size of network

5   Conclusion and Future Work

In this paper, we have motivated the need for a multi-scale architecture for sensor networks. Apart from enabling multi-resolution collaboration, a clustering hierarchy allows the network to scale to a very large number of sensors. Our architecture design adapts to the communication and collaboration requirements of the application, reducing communication energy and bandwidth usage. Our architecture also provides an abstraction to its low-level networking aspects, thereby simplifying application design.

We are currently implementing the architecture in TinyOS. We are using the fibers as blocking threads to implement Abstract Regions. This will enable us to deploy real applications and quantify the utility of our abstractions and adaptation interface. Currently, our evaluation is limited by the constraints of the *ns-2* simulator.

In this paper we have not addressed sensor network reliability or QoS requirements. These are important aspects that deserve attention. We plan to develop abstractions with tunable parameters through which the application can control the trade-off between resource usage and accuracy/reliability. Abstract Regions currently provides a tuning interface, but it entails the application to specify low-level parameters such as number of retransmissions. We intend to build tunable parameters at a higher level, which will enable the application to set goals, which will be automatically translated by the networking layer into the low-level parameters.

The Selectors implementation is currently fairly straightforward. The need to share attributes between hardware, application, and networking layer remains. This has been tackled previously in various approaches towards cross-layer design. However, there is ample scope for formalizing these cross-layer interactions and implementing them in an efficient way. Very complicated non-scalar Selectors can be supported by a fallback function provided by the application or by a loadable kernel module written by the application writer.

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: A survey. Computer Networks **38** (2002) 393–422

2. Welsh, M., Mainland, G.: Programming sensor networks using abstract regions. In: NSDI. (2004) 29–42

3. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: A scalable and robust communication paradigm for sensor networks. In: Proceedings, Sixth Annual Int. Conf. on Mobile Computing and Networking (MobiCOM '00), Boston, Massachussetts, USA (2000) 56–67

4. Heidemann, J., Silva, F., Estrin, D.: Matching data dissemination algorithms to application requirements. In: SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems, ACM Press (2003) 218–229

5. Park, S.J., Vedantham, R., Sivakumar, R., Akyildiz, I.F.: A scalable approach for reliable downstream data delivery in wireless sensor networks. In: MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing, ACM Press (2004) 78–89

6. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L.: Fractionally cascaded information in a sensor network. In: IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks, ACM Press (2004) 311–319

7. Ganesan, D., Greenstein, B., Perelyubskiy, D., Estrin, D., Heidemann, J.: An evaluation of multi-resolution storage for sensor networks. In: SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems, ACM Press (2003) 89–102

8. Yalagandula, P., Dahlin, M.: A scalable distributed information management system. In: SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, ACM Press (2004) 379–390

9. Whitehouse, K., Sharp, C., Brewer, E., Culler, D.: Hood: a neighborhood abstraction for sensor networks. In: MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services, ACM Press (2004) 99–110

10. Du, S., Khan, M., PalChaudhuri, S., Post, A., Saha, A., Druschel, P., Johnson, D.B., Riedi, R.: Self-organizing hierarchical routing for scalable ad hoc networking. Technical report, Rice (2004)

11. Chen, B., Morris, R.: L+:scalable landmark routing and address lookup for multi-hop wireless network. Technical Report MIT-LCS-TR-837, Laboratory for Computer Science Massechusetts Institute for Technology (2002)

12. PalChaudhuri, S., Du, S., Saha, A., Johnson, D.: Treecast: A stateless addressing and routing architecture for sensor networks. In: Proceedings of the 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN). (2004)

13. Culler, D., Shenker, S., Stoica, I.: Creating an architecture for wireless sensor networks. In: http://today.cs.berkeley.edu/SNA/. (2004)

14. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. In: ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems, ACM Press (2000) 93–104

15. Ni, S.Y., Tseng, Y.C., Chen, Y.S., Sheu, J.P.: The broadcast storm problem in a mobile ad hoc network. In: MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, ACM Press (1999) 151–162

16. Williams, B., Camp, T.: Comparison of broadcasting techniques for mobile ad hoc networks. In: MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, ACM Press (2002) 194–205

17. Blum, J., Ding, M., Thaeler, A., Cheng, X.: Connected Dominating Set in Sensor Networs and MANETs. In: Handbook of Combinatorial Optimization (Editors D.-Z. Du and P. Pardalos), Kluwer Academic Publisher (2004) 329–369

18. Clausen, T., (editors), P.J., Adjih, C., Laouiti, A., Minet, P., Muhlethaler, P., Qayyum, A., L.Viennot: Optimized link state routing protocol (olsr). RFC 3626 (2003) Network Working Group.

19. Wu, J., Dai, F.: Broadcasting in ad hoc networks based on self-pruning. In: Proceedings of Infocom '03. (2003)

20. Bao, L., Garcia-Luna-Aceves, J.J.: A new approach to channel access scheduling for ad hoc networks. In: MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking, ACM Press (2001) 210–221

21. Rajendran, V., Obraczka, K., Garcia-Luna-Aceves, J.J.: Energy-efficient collision-free medium access control for wireless sensor networks. In: Sensys '03: Proceedings of the first international conference on Embedded networked sensor systems, ACM Press (2003) 181–192

22. Peng, W., Lu, X.C.: On the reduction of broadcast redundancy in mobile ad hoc networks. In: Poster at MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing, IEEE Press (2000) 129–130

23. Qayyum, A., Viennot, L., Laouiti, A.: Multipoint relaying for flooding broadcast messages in mobile wireless networks. In: HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9, IEEE Computer Society (2002) 298

24. Wagner, R., Sarvotham, S., Baraniuk, R.: A multiscale data representation for distributed sensor networks. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Philadelphia (2005)