Attribute-Based Routing in Clustered SNETs

Wang Ke and Thomas D.C. Little
Department of Electrical and Computer Engineering
Boston University, Boston, Massachusetts
{*ke,tdcl*}*@bu.edu*

October 16, 2005

MCL Technical Report No. 10-16-2005

Abstract

We posit that SNETs are, like early embedded computing systems, hardwired using minimalist techniques to save energy and minimize system cost and thus lack flexibility and reconfigurability more typical of modern computer system design (hardware and software).

While the desire to economize energy consumption for system longevity will dictate spartan hardware resources, we believe there is an opportunity to improve flexibility in system operation, especially in data egress and dissemination within the SNET. We seek to enable flexibility and reconfigurability, conserve energy via limiting unnecessary SNET communications.

Our proposal relies on the use of attributed data sourced from individual sensor nodes and the use of attribute-based routing via a set of elected and rotated clusterheads. By understanding the mission requirements of the SNET application we can configure routing pathways that are matched to an energy-efficient topology and sleep-wake schedule for participating nodes. Finally, the attribute-based routing rules allow reconfigurabilty and convergence to energy conserving states of the SNET during changing SNET application missions.

In this paper we describe the core routing function achieved by our proposal and illustrate how it enables a rich set of capabilities not present in current SNET deployments.

1 Introduction

Current technological advances are enabling the deployment of wireless sensor networks [1, 2, 3] for many different applications. Such applications are also varied in scope and purpose, ranging from object tracking [4, 5], structural health monitoring [6], habitat monitoring [7] and monitoring of the environment and its resources [8, 9].

Sensor network applications have thus far been developed monolithically, i.e., sensors are programmed and deployed for a single task, with all communication paradigms set for one purpose. However, with the decreasing cost of the devices, and the increasing number of sensing capabilities a single device exhibits (i.e., a single mote [2] can sense light, relative humidity, temperature, pressure and has a 2-axis accelerometer, with the potential of attaching microphones and, with an expansion board [10], even videocameras), a deployed sensor network has resources that can fulfill multiple tasks within itself or collaborate with other co-located sensor networks to form a larger unforeseen sensor network application.

We envision in the future the appearance of wide area sensor networks that become a resource shared by multiple communities across diverse fields. Thus, different subsets of sensors within the large network would be the target of inquiries[1] initiated by different users in the community, each requesting different types of data. Also different applications may be executed at different spatial locations or at different time intervals in the network, each requiring different communication needs.

Routing paradigms that rely on flooding, such as Directed Diffusion [11], would generate too much redundant traffic in the scenario above, since inquiries may be initiated by different users targeting only subsets of deployed sensors. Geographic routing models such as GPSR [12] or GEAR [13] are actually application independent. They require applications to discover or know *a priori* the location of the data or the sink. Data-centric models built on top of such geographic models, such as GHT [14], DIM [15], DIFS [16] and DIMENSIONS [17] do not offer lower level control over communication patterns that different applications may require when transmitting data.

The goal of our work is then to provide an infrastructure that supports the different data inquiry mechanisms with increased lower level communication control.

In order to achieve this goal, we propose initially establishing an attribute-based hierarchical clustering in the sensor network. The hierarchy of attributes reflects physical containment relationships, with higher level clusters encompassing lower level clusters. The clusters established are attribute equivalent clusters. The attributes chosen are those that have an *a priori* high probability of being inquired. Clusterheads at different hierarchy levels maintain paths to one another, and are responsible for collecting attribute information of cluster member nodes. This information is used by the clusterheads to route inquiries to relevant parts of the sensor network, eliminating dissemination of redundant and energy consumptive traffic.

Once attribute equivalent regions have been established, clusterheads can coordinate intra- and inter-cluster data dissemination based on the application requirements. Thus part of the sensor network that is being tasked with an object tracking application may have different routing rules than another part which has been given the task of collecting soil humidity profile. Different performance expectations from the application may also result in different routing rules. Clusterheads in this context act as attribute-based routers, and can support different routing rules based on the application needs.

Thus our attribute-based routing scheme is data-centric (as opposed to host-centric models), in which

[1]*Inquiry* is a term we use to denote a generic way to task portions of the sensor network with requests for new types of data with different performance expectations.

the routers can forward data packets to nodes that satisfy *a priori* specified attributes (as opposed to disseminating the packets to all nodes in the network and have each node drop the packets whose attributes do not match with those of the sensor) and routers support different routing rules based on application level communication needs (as opposed to all sensors in the network processing packets in the same way). In this paper we specify data structures and algorithms that implement our attribute-based routing in wireless sensor networks.

We present in the next section some example scenarios that drive our research in attribute-based routing (Sec. 2). Then we present the related work in Sec. 3. In Sec. 4 we delineate our design choices and show the basic functionality specification, as well as data structure and algorithms related to the implementation of attribute-based routers. We illustrate our concepts with an example (Sec. 5) and present analysis of the memory requirements and routing efficiency in a subsection (Sec. 5.1). We conclude in Sec. 6. An appendix discusses some implementation issues in more details.

2 Scenario

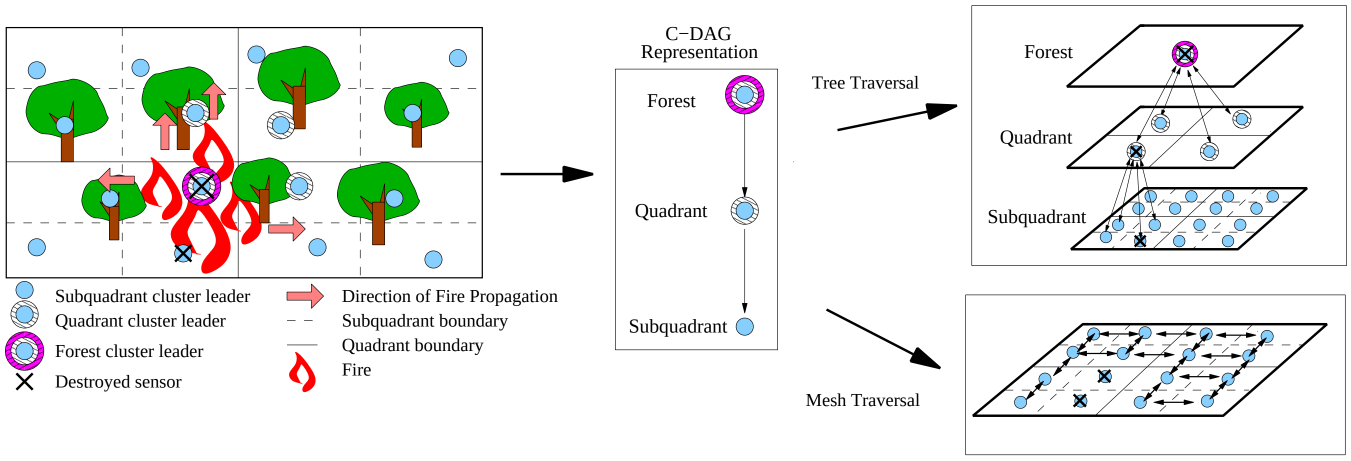

Figure 1: Sensors deployed in a forest under $\{Subquadrant \subset Quadrant \subset Forest\}$ C-DAG and two modes of cluster traversal: *Tree* or *Mesh*.

Enviromental monitoring and Fire Alarm Consider the following scenario: multi-modal sensors are deployed over an area for climate monitoring, and are collecting average values of temperature and humidity when suddenly fire is detected. One local application, designed to detect and track how the fire propagates, is awakened and immediately alerts neighbor sensors so that the fire front can be detected. This scenario is depicted in Fig. 1.

The communication needs of the sensor network while in the first stage of monitoring average temperature and humidity can be thought of as hierarchical. Data is slowly aggregated within each cluster by the cluster leader and sent to the base station. Thus sensors communicate using the "Tree traversal" mode found on the upper right side of Fig. 1. However, the communication needs of the fire detection application add a new component: the necessity for clusters to communicate with neighbor clusters, so that the fire propagation can be tracked over time. The way the fire propagates is also recorded and this information is spread to contiguous clusters, as in the event of a fire there is no guarantee that the top hierarchical leader has survived the fire. This situation is also depicted in Fig. 1, in which the sensor which plays the role of

Forest leader, as well as Quadrant SouthWest leader has been destroyed by the fire. If the tree traversal hierarchical mode is the only communication mode, then other quadrant leaders would not be able to detect the fire in time. However, by using the “Mesh traversal” mode (lower right side of Fig. 1) at the lowest level of the attribute hierarchy (Subquadrant clusters), sensors are able to spread the alarm and continue detecting the fire front.

The example above illustrates how different applications may require different communication patterns. We can think of other scenarios that may also require switching communication patterns. Consider the two more described below.

Toxic Chemical monitoring Suppose a sensor network is deployed to detect the presence of toxic chemical components. Sensors cluster together, and within each cluster aggregate data from cluster members. Cluster leaders exchange information with each other and jointly decide whether the chemical component is present in their neighborhood or not. Once the chemical is detected, the cluster leader sends a warning message up the hierarchy to the base station.

Habitat Monitoring and Wildlife tracking A sensor network spread across a region for habitat monitoring collect, aggregate and send data up the hierarchy back to the base station. But when an animal whose habits are being studied enters the network, a tracking application is launched which uses the mesh traversal capabilities to send warnings to adjacent clusters.

The next scenario illustrates how supporting different addressing systems concurrently may be beneficial for application deployment.

Tracking and Management Sensors are used to identify, track, log movements, and raise breach of security alarms (a person being where it ought not to be) within a high security building. The tracking part of the application requires communication from sensor clusters with their neighbor clusters, while the information logging benefits from a hierarchical approach. The alarm feature benefits from a logical, organizational type of addressing system (e.g., “Multimedia Communications Lab”) rather than a location based addressing system (“8 Saint Marys St, Room 445”). Now, suppose that a building maintenance application is also being deployed on the same network. For this application, knowledge of the organizational address (e.g., “ECE Dept”) would not be as helpful as knowing the correct location (“8 Saint Marys St, Rm 324”). Now, suppose that within that building a temperature and lighting control and maintenance application is also being deployed on the same network. This application is not concerned with security issues and is only interested in regulating temperature and lighting conditions when a person is detected within a room. Temperature and lighting controls are local, so information sharing must be localized (room 445 may have no one while room 446 is hosting a party - it is useless for room 445 to know of the conditions in room 446 and vice-versa despite their being right next to each other) but in the event of a mal-function (temperature rises above 90 F when no one is in the room) the alert must go to the building manager’s/maintenance office, which may not even be in the same building. In this case, a hierarchical clustering scheme may facilitate locating the building manager/maintenance office. For the maintenance team, knowledge of the organizational address (e.g., “ECE Dept”) would not be as helpful as knowing the correct location (“8 Saint Marys St, Rm 324”). We can see from this example that in addition to supporting different routing schemes, an additional desirable feature of a routing infrastructure for sensor networks is its capacity to support simultaneously multiple addressing schemes.

We present in the next section an overview of related work in the area.

3 Related Work

Past approaches such as diffusion [11, 18] flood inquiries to the network, and build gradients that collect data back. Such approach is limiting, for different applications may have different needs, and if sensor networks are shareable resources, then a single communication paradigm is not sufficient for fully utilizing the resource. Moreover, if the sensor network is shared, requests may arrive for all different forms and types of data, causing frequent floods that may be irrelevant to most of the nodes in the network and wasting energy.

In order to reduce the redundant transmission of packets, location information is explored in order to direct how data can be routed. GPSR (Greedy Perimeter Stateless Routing [12]) and GEAR (Geographical and Energy Aware Routing [13]) are two examples of geographical based routing. TBF (Trajectory Based Forwarding [19]) specifies trajectories that data can follow. TTDD (Two Tier Data Dissemination [20]) has data sources build uniform grids of data dissemination nodes. TTDD's main emphasis is in efficiently supporting sink mobility. CBM (Content Based Multicast [21]) has a similar approach to a hybrid model of Diffusion, in which data sinks pull data from a specified region of interest, and data sources push data to a specified region in which information is relevant (e.g., sensors detecting a target moving eastward may push alarm data further towards easter parts of the network). Rumor Routing [22] establishes paths to events by employing agents, which are packets with a high TTL field, that are propagated from node to node, leaving information on observed events. Queries are also propagated in the same way, and data are sent back when there is a rendezvous of two paths. While these schemes do not rely on network wide floodings, most [12, 13, 19, 20, 21] need the presence of location services to operate, and their addressing scheme is independent of the applications they support. In other words, a data sink must know *a priori* the region to which send the data request and vice-versa [12, 13, 19, 21]. TTDD is focused on supporting sink mobility, and does not support inquiries that requests data from same attribute regions. Rumour routing likewise does not offer direct support for queries that request data from regions of sensors. In the schemes above there is no exploitation of potential spatial correlation of sensor data to forward and request packets.

Attempts to exploit data correlation in routing can be found in [23, 24, 25, 26, 27]. In [23] sensors are clustered and the clusterhead queries cluster members regarding an observed event until the information it possesses satisfies a threshold value according to a utility metric. Requests for the event are forwarded based on the gradient levels established by the information utility metric. Work in [24] discusses ways to route data when data are spatially correlated. It uses a correlation index to determine the optimal one level cluster size to aggregate data. ACQUIRE [25] proposes a query propagation scheme in which the sensor receiving a query perform a d-hop look-ahead to see if there is information that can answer the query. If not the query is then propagated (through Random Walk or other mechanism). In our work, we explore data redundancy at multiple levels of a hierarchy and we route data through this hierarchy, instead of being guided by a utility metric function or by look-ahead steps.

Semantic Routing Trees (SRT) are proposed in TinyDB [26], in which tree structures are formed in the sensor network based on sensed values and queries are forwarded to children that have values within the range requested. Like SRT but with a more generalized filtering approach, CBCB (Combined Broadcast and Content Based routing [27]) adopts a two layer approach (one broadcast layer and one content-based layer) to place predicates (a set of constraints on the attributes) at the routers. Data that matches a predicate will be forwarded to the appropriate sinks. Our work differs from SRT and CBCB in that we do not attempt filtering at sensor level, but instead form attribute equivalent regions that help route traffic. We also support different communication needs in such regions.

Routing based on data content and semantics has also been studied by the Peer-to-Peer (P2P) network community [28, 29, 30]. A taxonomy for "content" network is described at [31]. Work in [29] proposes clustering nodes togther (i.e., adding logical edges) based on content similarity, while [28] suggests that nodes in the network should "learn about" the contents of other nodes in the network so that queries may be more efficiently forwarded. In particular, [30] offers an ontology-based solution to what content similarity may mean, by offering a matching process that involves a concept/content's "name," "attributes" and "relationships." As can be seen, overlay P2P networks have the flexibility of exploiting dynamic changes to the network topology by adding/removing logical links. The papers above assume a static knowledge representation system, i.e., a content categorization system, an ontology, etc., that is static and from this knowledge representation attempt to establish logical links between hosts of the P2P network that will make query routing more efficient. Our work can be seen as complementary: given sensors and their data, formulate attribute equivalencies in a hierarchy that reduces redundant traffic.

The advantages of being able to select the routing protocol at run-time have been pointed out by the active network community [32]. Work in [33] proposes encapsulating packets in SAPF (Simple Active Packet Format) headers, which carry indicators to an active node's FIB (Forwarding Information Base), guiding packet forwarding behavior at run-time. The routing example shown in [33] is tree based. In [34] the authors propose an overlay scheme that allows active nodes to coexist with passive nodes. The active nodes track communication paths to each other reactively. Our work shows how dynamic routing protocol selection can be implemented in attribute clustered WSNETs. We show the routing rules and the performance analysis for both the tree and the mesh traversal modes. Furthermore, we show how the changing density of active routers (in our case attribute based routers or cluster leaders) in the network, achieved through changing the number of levels in the attribute hierarchy, affects the expected performance of the two routing schemes.

In the next section we explain the routing related problems we address in the context of sensor networks, and specify the preliminary assumptions we make on the infrastructure of the sensor network.

4 Attribute-Based Routing

In the next subsections we describe our design decisions and functionality specifications of our routing scheme.

4.1 Design

Data Centric Routing In the past routing algorithms focused on reaching a specific host. This paradigm can be justified when the hosts are few when compared with existing data sets, that is, many data sets were mapped to a relatively few hosts. This is illustrated in the top part of Fig. 2, in which the packet sent by A wants to reach hosts B,C or D, which possess the data A desires. With the advent of sensor networks, however, there is a reversal on the numbers on each side. With the decreasing cost of sensors, it is envisioned that many physical phenomena will be monitored through network of sensors, that is, sensors sample the same phenomenon at different points in space and time (thus intrinsic to sensor applications is the notion of location and relative time). These sensors often collect highly correlated or even same data values. Under such circumstances, finding a specific host is not as essential as finding the desired data, for such data may be replicated in many hosts. This converse situation is illustrated in the bottom part of Fig. 2, in which the data A desires is stored in all but a few hosts.

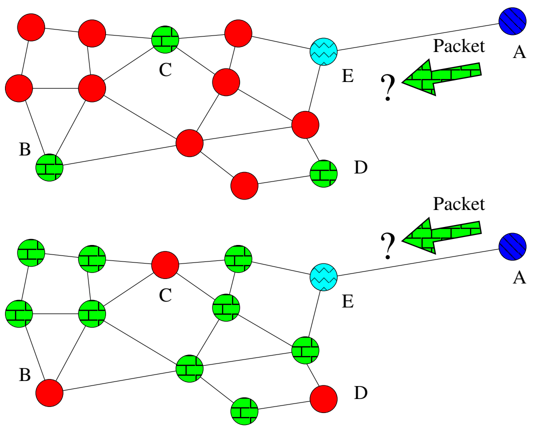

Figure 2: *Top*: data exists in a few hosts - packet sent by A needs to reach hosts B, C or D. *Bottom*: data replicated in almost all hosts - packet sent by A can retrieve data from any host except B, C, D or E.

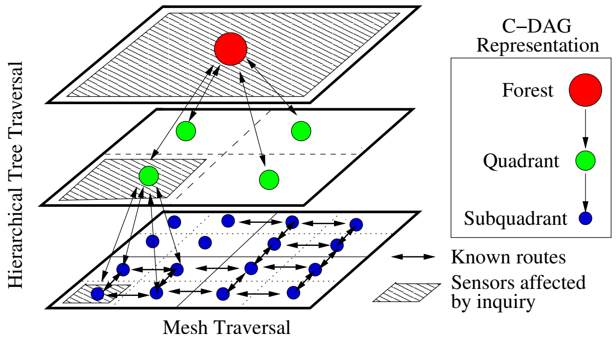

Figure 3: *Left*: Hierarchical view of the clusters in the network at different levels in the attribute hierarchy. Inquiries addressed to attributes at different levels affect different number of sensors. A logical hierarchical tree is formed by the paths that link the cluster leaders between hierarchy levels (*Forest* to *Quadrant* and *Quadrant* to *Subquadrant*), while a logical mesh is formed by paths that link clusters at *Subquadrant* attribute level. *Right*: C-DAG representation of the attribute hierarchy $\{$*Subquadrant* $\subset$ *Quadrant* $\subset$ *Forest*$\}$.

Hierarchical Location Based Attributes It is possible to implement data-centric approaches to routing that emphasizes finding the values of detected events (e.g. DIM [15], DIFS [16]). In such systems the location where the events occur is not as relevant as the fact that a specified event occurred. In our design we propose foundational support for inquiries of events in which the location of the event is not disassociated with the event itself (see the examples in Sec. 2). Such support enables easier addressing of only subsets of sensors for retasking or inquiry delivery, which in its turn can be used to reduce redundant traffic and to implement different communication patterns needed by different applications. In Fig. 3 inquiries addressed to the *Forest* attribute would be distributed to the whole network, while inquiries to a *Quadrant* would be delivered to only $1/4^{th}$ and inquiries to a *Subquadrant* would be delivered to only $1/16^{th}$ of the original network. It should be mentioned that it is possible to implement a value focused approach on top of our scheme, as well as it is possible to store location attributes in value focused routing paradigms.

The set of attributes used to address sensors are location based attributes that satisfy containment relationships. Such relationships form naturally a hierarchy, in which the higher level attributes contain the lower ones. The hierarchy can be represented via a directed acyclic graph (DAG), in which nodes represent attributes and edges the containment relationships. Because of this we call these DAGs Containment DAGs or C-DAGs for short (see the C-DAG in the right side of Fig. 3). Sensors that have the same attributes are clustered together, and such clustering happens in a hierarchical manner. Each cluster represents an attribute-equivalent region, and by controling which and how many attributes are part of the hierarchy we can control whether the propagation of inquiries is pure flooding (one level in the hierarchy), host centric (attributes have resolution that can pinpoint individual sensors uniquely), or a hybrid approach, in which we have attribute equivalent regions communicating with each other. Currently we posit that attributes should be selected based on an *a priori* assumption on the frequency such attributes will be called by users in their inquiries. We support dynamic modifications to the C-DAG after deployment to insert or remove nodes to more efficiently guide data propagation.

Attributes in our scheme must also be able to establish well-defined adjacency relationships between clusters. Such adjacency relationships are used to guide data forwarding. Clusters that are on the same hierarchy level but with different attribute values use the adjacency information to guide traffic forwarding.

Rules Based Routing Sensors process incoming packets based on routing rules that are grouped by sets. Different sets of routing rules define different communication patterns. The initial set of routing rules is either present in the sensors before deployment or is propagated at cluster formation time. These are application independent sets. Sensors may hold different application independent sets simultaneously. The initial set is the default one. If an application needs to invoke other set of routing rules for packet processing, it must indicate so by adding a routing rule set identifier in the packet header (see Sec. 4.1.4 and 4.1.5). If the requested routing rule set is not present in the sensor, then default routing behavior is adopted. Sensors may elect to accept changes to the application independent routing rules set, but this is not enforced. In Fig. 3 two communication patterns can be seen, the “Hierarchical Tree Traversal” mode, in which lower level cluster leaders communicate with higher level cluster leaders, routing in a hierarchical virtual tree, or the “Mesh Traversal” mode, in which cluster leaders at the lowest level in the C-DAG communicate with adjacent cluster leaders, routing on a logical mesh.

Applications may bring with them their own set of routing rules, though, and these may be changed dynamically by the applications. If applications do not require change of routing rules at all sensors, but only a small subset, then they may request forming a small cluster for this purpose. Sensors that become members of such cluster either must possess the same routing rule set or request the set from the cluster leader. Clusters formed with the purpose of changing routing rules are called application clusters. Sensors may change membership status of application clusters at will. Application clusters are established through modified (simpler) versions of the attribute based hierarchical clustering algorithms. Members of the application cluster are given “names,” that is, a string that identifies a particular set of attribute name-value pairs. Sensors matching the set assume the “name” given. Thus even in application clusters the identification of cluster members is attribute based. The purpose of these clusters is just to enable different communication patterns for a small subset of sensors, and thus no inherent support exists for managing high numbers of members. There is no limitation on the possible number of members, but a single cluster with many members will have significant performance degradation.

By setting routing as an interpreted process, we allow dynamic configuration of nodes to support different communication patterns and thus meet different communication needs from the various applications that share the network. It is true that mixing different routing rules may lead to inconsistencies and failures in the routing mechanism (cannot deliver packet even when there is a path between source and destination). We present default routing rules that mimic well known algorithms for routing in mesh and trees. We believe that supplying these basic routing algorithms and at the same time giving more lower level control of the routing functionality is the best approach for sensor network application development. Developers may come up with their own routing rule set and these may be re-used by other application developers. We present next the data structures and algorithms used in the routing process.

4.1.1 Naming

Address names in our routing scheme are composed of a sequence of attributes. Attributes possess name, type and value. Attribute names are specified as strings. The default type for all attributes is string, unless otherwise specified. Additional possible types are char, short, integer, float and double. If the type is an char, then the value is stored in one byte. Two bytes for short, 4 for int and float and 8 for double. String values are stored in an array of chars, with a special termination character like C strings.

We assume that sensors that are deployed are tagged with location based attributes that are relevant to the users of the sensor network. That is, users select these attributes in their inquiries. These location-based attributes can be as specific as GPS coordinates or can be as generic as *Quadrant*, *Subquadrant*, etc.

Attributes are only well-defined in the context of an attribute hierarchy. Attribute hierarchies are represented via a C-DAG specified through a file, and brings with it a list of all attribute names and their respective types, together with possible values for each attribute. In addition, containment relationships and adjacency relationships are clearly defined for attribute names and attribute values, respectively. This means that given two attribute names, we must be able to tell whether one is contained in the other (e.g., subquadrant $\subset$ quadrant, GPS X coordinate $\not\subset$ GPS Y coordinate). Likewise, given two attribute values (e.g., “NorthEast” and “NorthWest”), when queried, “NorthEast” *IsAdjacentTo* “NorthWest” returns true. See appendix .1 for more details.

A hash function (such as MD5 or SHA) generates a message digest for the file specifying the attribute hierarchy that is used as the identifier for this hierarchy. When sending information packets, sensors attach the hierarchy’s identifier together with the set of attributes that form the address. The order of appearance of the attributes in an address is relevant: most encompassing attributes (higher in the hierarchy) appear first.

It is assumed that sensors in the network will be clustered according to the attribute nodes defined in a C-DAG. Each node in the C-DAG represents an attribute, and sensors with the same attribute value will be clustered together. The clustering process happens across all attribute nodes present in the C-DAG, so that sensors will belong to at least one cluster (e.g., the one representing the root node). Thus sensors in a forest may be clustered together under one big *Forest* attribute cluster, while *Quadrant* and *Subquadrant* clusters will be formed according to the different values quadrant and subquadrant may have. That is, clusters with “NorthWest,” “NorthEast,” “SouthWest” and “SouthEast” quadrant attribute values will be formed, and likewise smaller clusters with subquadrant values of “NorthWest,” “NorthEast,” etc, will be formed.

4.1.2 Clustering

Within each cluster a leader will be elected. Sensors within the cluster know of a path to the cluster leader, and will forward relevant attributes and sensed data they possess to the leader. The specific data and attributes sent may vary in time depending on the inquiries users make to the sensor network. The cluster leader will aggregate such information into a catalog and will forward this catalog information to its higher level cluster leader, e.g., a *Subquadrant* cluster leader will forward information from its subquadrant

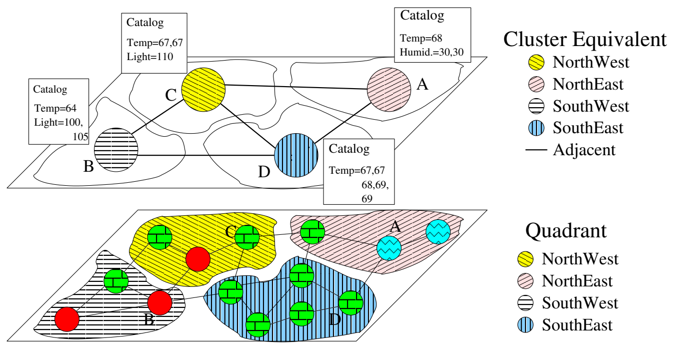

Figure 4: *Top*: *Quadrant* attribute equivalent clusters, in which cluster leaders (A, B, C and D) hold catalog information. Adjacency relationships are represented through the dark lines connecting cluster leaders. *Bottom*: Sensors that share the same quadrant attribute are grouped together to form the attribute equivalent clusters shown in the top part of the figure.

cluster sensors to the leader of the *Quadrant* cluster it belongs. In the upper figure of Fig. 4, catalog information is held in each quadrant by the cluster leaders (A, B, C and D) and only the "SouthWest" quadrant has temperatures lower than 65 F. Inquiries looking for sensors with such data need not be propagated within other clusters.

Due to the broadcast nature of the clustering formation protocol, sensors know whether they are "border" cluster sensors (that is, they are within range of a sensor that belongs to another cluster) or not. Border cluster sensors will overhear the broadcast of a neighbor that selects a different leader, if their neighbor belongs to the same attribute hierarchy, or the retransmission of the original cluster formation packet in which the sender specifically flags as not belonging to any cluster in the hierarchy being formed. In such transmission the sender usually also transmits information about the clusters of the hierarchy to which it belongs. Thus border cluster sensors are able to inform their cluster leaders of adjacent cluster's attributes. In Fig. 4 the adjacency relationships are represented by dark lines connecting the cluster leaders.

Sensors that reside on the path between border cluster sensors and the cluster leader learn a route to the attribute region represented by the adjacent cluster. Other routes that sensors may learn include paths to their cluster leaders (information obtained during cluster formation time), and occasionally paths to clusters a sensor is not a member of (this information is usually learnt when the sensor lies in the path that a lower level cluster leader used to send catalog information to an upper level cluster leader). Sensors store these path information in a routing table structure we describe next.

4.1.3 Routing Information Storage

The data structure we use to store routing information is better viewed as composed of three parts: the first part is composed of graph structures representing known attribute hierarchies and which is indexed by the hierarchy identifiers. The second part lists attributes which have been received (i.e., found in a packet) yet whose attribute hierarchy is unknown. The third part lists current membership clusters the sensor is part of, and routing information to cluster members. For simplicity we will refer this three-part structure as

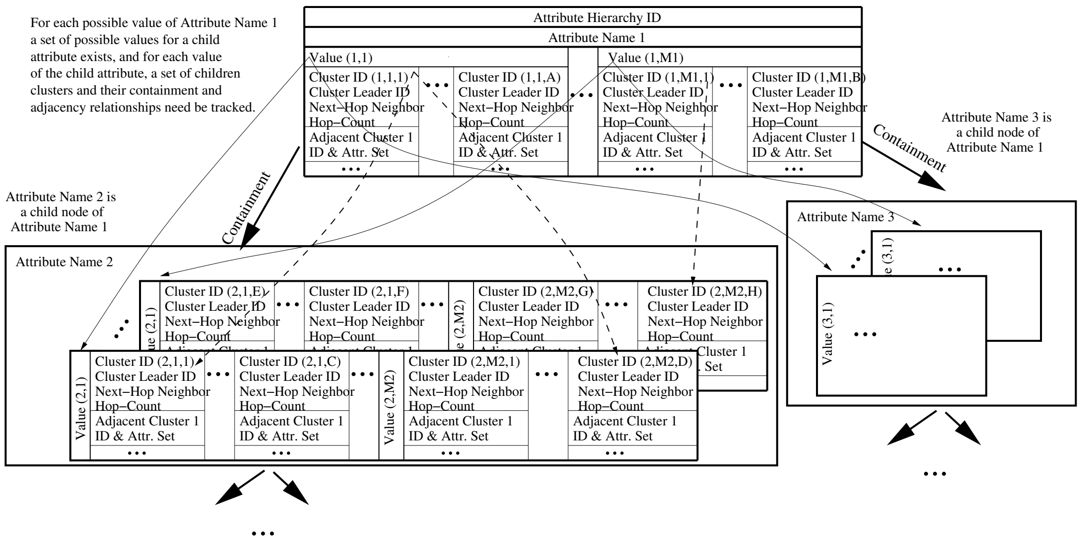

Figure 5: Graph Structure of an Attribute Hierarchy for Routing.

routing table, even though it is not technically a table.

The graph structures are DAGs, and each node represents an attribute in the hierarchy. Within the node we list all attribute values that have been seen by the current node, and track clusters that possess those values, listing their clusterheads, hop distance to the clusterhead, next-hop neighbor to reach that cluster, and reported adjacent clusters.

In Fig. 5 an attribute hierarchy, as tracked within the routing table, is represented. Each rectangular table represents a node in the hierarchy. The top rectangular table is the root node of the hierarchy, and it represents “Attribute Name 1.” Possible values for “Attribute Name 1” range from $Value(1, 1)$ to $Value(1, M1)$. For $Value(1, 1)$ there may exist A clusters in the network that match the attribute value. Each one is tracked, together with the cluster ID, cluster leader ID, next-hop neighbor to reach the cluster leader, hop-count to cluster leader, and any reported adjacent clusters. For each potential value of “Attribute Name 1” this information is also tracked. The “Containment” arrows link two nodes, so “Attribute Name 2” and “Attribute Name 3” are nodes in the attribute hierarchy that are contained by “Attribute Name 1.” This means that for every cluster with an attribute value that is in “Attribute Name 1,” there may exist clusters in it with values associated with “Attribute Name 2.” The way we track it in the routing table is to associate with each possible value of the parent node a set with all possible values of the child node. This is represented by the arrows linking $Value(1, 1)$ and $Value(1, M1)$ to their respective row of values in “Attribute Name 2” ($Value(2, 1)$ to $Value(2, M2)$) and “Attribute Name 3” (only the first value of the row in “Attribute Name 3” is represented). Individual clusters representing a parent node track those clusters of a child node individually. These are the dashed arrows that link “Cluster ID(1,1,1)” under $Value(1, 1)$ in “Attribute Name 1” to the clusters in the first row of values in “Attribute Name 2” and the arrow linking “Cluster ID(1,M1,1)” to “Cluster ID(2,M2,H).”

The second part of the routing table is composed of two more data structures. The first one brings with it the attribute name-value pairs found in a packet, the neighbor through which the packet was received, the hop-distance to the original sender and the time received. We can see the representation of this first structure in Fig. 6(a). The leftmost column indexes the number of packets with distinct list of attribute name-value pairs, and each element in the column tracks the next-hop neighbor from which the packet was received, the hop count to the original sender, time received and the list of attribute name-value pairs.

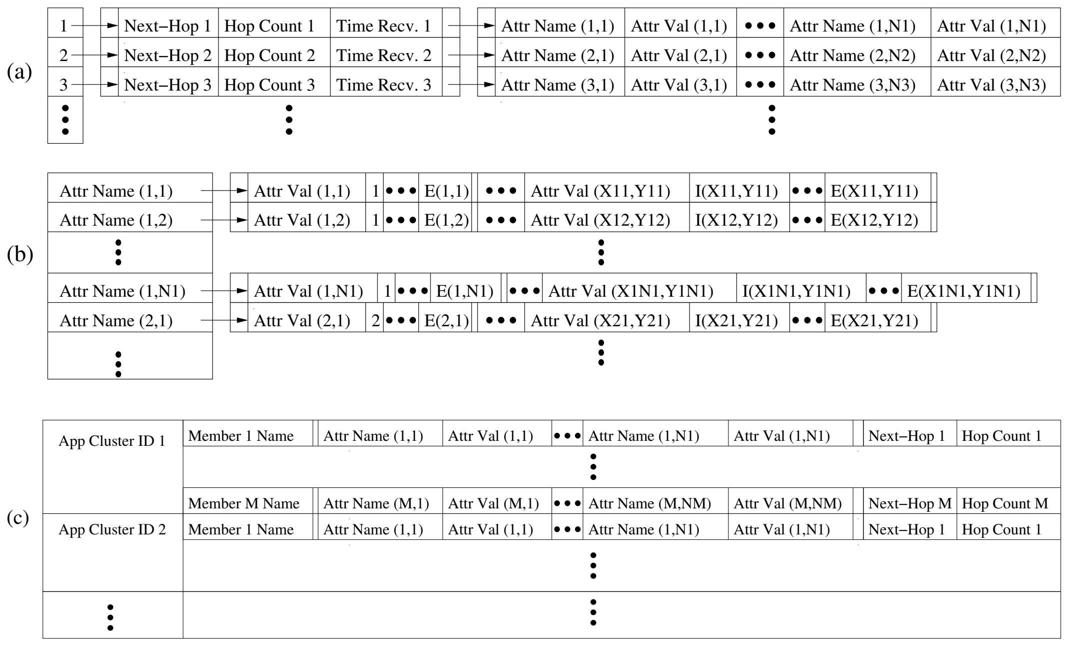
(a)
1
2
3
Next–Hop 1 | Hop Count 1 | Time Recv. 1
Next–Hop 2 | Hop Count 2 | Time Recv. 2
Next–Hop 3 | Hop Count 3 | Time Recv. 3
Attr Name (1,1) | Attr Val (1,1) | ••• | Attr Name (1,N1) | Attr Val (1,N1)
Attr Name (2,1) | Attr Val (2,1) | ••• | Attr Name (2,N2) | Attr Val (2,N2)
Attr Name (3,1) | Attr Val (3,1) | ••• | Attr Name (3,N3) | Attr Val (3,N3)
(b)
Attr Name (1,1)
Attr Name (1,2)
Attr Name (1,N1)
Attr Name (2,1)
Attr Val (1,1) | 1 ••• E(1,1) | ••• | Attr Val (X11,Y11) | I(X11,Y11) ••• E(X11,Y11)
Attr Val (1,2) | 1 ••• E(1,2) | ••• | Attr Val (X12,Y12) | I(X12,Y12) ••• E(X12,Y12)
Attr Val (1,N1) | 1 ••• E(1,N1) | ••• | Attr Val (X1N1,Y1N1) | I(X1N1,Y1N1) ••• E(X1N1,Y1N1)
Attr Val (2,1) | 2 ••• E(2,1) | ••• | Attr Val (X21,Y21) | I(X21,Y21) ••• E(X21,Y21)
(c)
App Cluster ID 1
App Cluster ID 2
Member 1 Name | Attr Name (1,1) | Attr Val (1,1) | ••• | Attr Name (1,N1) | Attr Val (1,N1) | Next–Hop 1 | Hop Count 1
Member M Name | Attr Name (M,1) | Attr Val (M,1) | ••• | Attr Name (M,NM) | Attr Val (M,NM) | Next–Hop M | Hop Count M
Member 1 Name | Attr Name (1,1) | Attr Val (1,1) | ••• | Attr Name (1,N1) | Attr Val (1,N1) | Next–Hop 1 | Hop Count 1

Figure 6: (a) *Top*: Structure that indexes packets received without attribute hierarchy; (b) *Middle*: Structure that indexes attribute names, followed by all possible values and the indices of packets in which each value was seen. (c) *Bottom*: Structure to track Application Cluster Routing Information.

The second structure stores the attributes seen in a packet individually, and indexes a series of entries from the first table from which specific values can be found. In Fig. 6(b) the leftmost column is a list of individual attribute names. Each attribute name tracks all the possible values seen (Attr Val(1,1) to Attr Val (X11,Y11)), together with the indices of packets in the first structure in which the value appeared (Attr Val(1,1) appeared in packets {1, ..., E(1,1)}, Attr Val (X11,Y11) appeared in packets {I(X11,Y11), ..., E(X11,Y11)}, etc.). Entries in these two tables are deleted after a time-out period. That is, sensors that do not identify which hierarchies they belong to are not assumed to have relevance in the long term deployment of the sensor network.

The third part of the routing table is used for tracking routing information within application clusters. It is indexed by the Application cluster ID, followed by its cluster members' names, a list of attribute name-value pairs that each member must match, the next hop neighbor and the hop count to reach the member. This can be seen in Fig. 6(c). Application clusters should be small in nature, and a flat table structure is reserved to track routing information. If a hierarchical approach is required, it should be implemented at the application level.

As we can see, sensors store only a next-hop value for an attribute value region. Every sensor in the network is essentially a distributed routing knowledge storage point. When senders transmit packets to a destination and include their own attribute hierarchy identifiers and attribute lists, they are essentially distributing hop-by-hop information on how to be reached to the sensors along the way. In the examples above either the sender is essentially announcing itself to nearby sensors and thus forming paths to itself (in the case of cluster formation), or the sender is already aware of paths to the destination (in the adjacency information update and the catalog update cases). We discuss in the next section issues in routing packets for which a path to the destination may not be known.

4.1.4 Rules-Based Routing

In our rules based routing, there are three different types of routing rules sets: (1) application independent, (2) application dependent and (3) application cluster routing rules. Application independent routing sets are stored in the sensors prior to deployment. Identifiers for these rules set may be pre-defined strings or numeric IDs. Application dependent rules are stored together with the application. The routing process allows the applications to supply a routing rules set file. The identifier for application dependent rules set is the message digest resulting from applying a hash function to the rules set file. The application cluster routing rules set are also stored together with the application. It may reside only in one host. Cluster members that do not possess the routing rules set request the routing rules. Identifiers for these rules set can be determined by the application.

We assume that the routing process will read from a configuration file and store the routing rules. Changes to the routing rules may be implemented as soon as the changes are made if the underlying host OS supports signaling. Otherwise the application must wait until the routing process becomes aware of the changes through its periodic checking of the file status.

Each "rule" in our rules based routing is composed of two parts: (1) a conditional statement and (2) an action statement. If the conditions specified are true, then the action is carried out. Otherwise, the following rule in the rule set is checked. If no conditional statement turns out true after going through all the rules, the packet is simply dropped. Our rules based approach essentially imposes a priority scheme over possible next-hop destinations. Each conditional statement defines a subset of all possible incoming packet states, and each action statement essentially defines a possible next-hop destination. Thus the order in which the rules are placed within the rule set reflects the priority assigned to each possible "state-destination" association. Ideally, the first rule in the rule set should reflect the most common applicable rule in the network. Because of this "condition-action" separation, the rule set can actually be described by a series of **if-then-else** statements.

We show here two sets of routing rules as example of application independent routing rules set. The first is to route packets within the same attribute hierarchy and the second to route between different attribute hierarchies.

Within the Attribute Hierarchy When sending packets within the same hierarchy, sensors may follow an algorithm like Algorithm 1. A sensor receiving a packet initially checks whether the destination address matches a known routing entry (Lines 9 and 10 - in this paragraph all Line references are with respect to Alg. 1). If the sensor itself belongs to the region satisfying the attributes sought, then the packet is flooded (Line 12). If there is a routing entry E matching the destination address and the packet was not received from the neighbor to which the packet need be sent to reach E, then the packet can be forwarded to that neighbor. Otherwise, the information stored in the sensor's routing entry probably is outdated and the destination address should be treated as unknown (after Line 15. If the sensor is a cluster leader, and the packet with an unknown destination address came from the parent cluster leader (Lines 16 and 17), then the packet is forwarded to any children clusters that have at least partially matched attributes, that is, there is no known attribute in the child cluster that has a different value than the values specified in the destination address (Line 19). If no such child cluster exist, then the packet is dropped. If the packet did not come from a parent cluster leader then the packet may be (1) forwarded to a higher level leader (Line 24) if there are attributes further up in the hierarchy that needs be resolved; (2) sent back to children clusters that have fuller matches with the destination attributes, assuming all the attributes from the root node to the current leader level are matched (Line 27) or (3) dropped, if neither of the two prior conditions can be satisfied (Line 29). Condition (2) above is correct because at cluster formation time all cluster leaders under the same parent instance know of each other. Thus, if a packet is destined to “Building=PHO, Floor=3,” then if a packet reaches a cluster leader for “Building=PHO, Floor=3”, this packet can be forwarded to all “Floor=3” clusters (Line 26). In this way any attribute that need be resolved under “Building, Floor” can be resolved at lower level clusters.

Full knowledge of how to route packets based on the attributes specified is only possible in the presence of an attribute hierarchy. The attribute hierarchy brings information on all possible attribute names and values, as well as containment and adjacency relationships. Therefore with the full knowledge of the attribute hierarchy a sensor not only knows *whether* the attributes sought can be satisfied, but also how to forward a packet to the appropriate regions to find suitable sensors. The root node of the attribute hierarchy must have full knowledge of the entire attribute hierarchy.

Between Attribute Hierarchies Routing packets between two sensor network applications may happen in two ways, (A) the two applications share the same geographic space, that is, either two sensor networks have been deployed at the same location, or two applications are sharing the same sensors, or (B) the two applications are separated by one or more sensor network in-between.

In case (A) above, since the two sensor networks are in the same geographic region, any cluster formation packet or new leader packet from one application will be stored by the sensor and the information shared by the other. Applications become thus mutually aware of each other’s attribute hierarchies and can route packets between them.

However, to have *a priori* knowledge of the attribute hierarchy is not always feasible, especially in the case (B) above, when we are connecting two sensor networks that are far apart geographically and are not aware of each other’s presence. This may happen when the sending network is probing the space around it to find networks with sensors satisfying certain attributes. That is, the sender specifies attributes that it believes a desired destination sensor must possess. Since this involves a very subjective evaluation of possible attributes, we propose a prioritized approach to the attribute matching process. The sender flags that there is no attribute hierarchy ID attached to the packet, and will flag either a single status for all attributes listed, or that each attribute will have its own status. The possible status are:

- *Required* - the packet must be delivered in the end to sensors that matches all name-value specifications of the “required” attributes. If no known sensor matches all the attributes then the packet is dropped.

- *Preferred* - the packet must be delivered to sensors that match the most number of name-value specifications of preferred attributes. In case two or more groups of sensors satisfy different sets of attributes but the sets have the same number of elements, the packet will be forwarded to all the groups. “Required” attributes have precedence over “preferred” attributes. If “preferred” attributes co-exist with “required” attributes in the same packet, the packet will be sent to the sensors that satisfy all the “required” attributes and the most number of “preferred” attributes. The packet will not be delivered even if one “required” attribute is not satisfied, independently of how many “preferred” attributes are matched.

- *Exploring* - “exploring” attributes are only relevant when there are no “required” attributes in the packet, and when no “preferred” attributes are matched. In this case, the packet will be forwarded first to the sensors that match the most number of name-value specifications of “exploring” attributes, then in the absence of any name-value match, to the sensors that match the most number of attribute names.

There is no provision in the status specification to flood the sensor network. This is achieved by a special flag in the packet header. See Sec. 4.1.5 for the packet specification.

Given the different status of the attributes, a sensor receiving a packet which has no attribute hierarchy attached follows the steps delineated in Algorithm 2. Essentially the sensor forwards the packet to a known destination (e.g. Lines 12 and 23 of Alg. 2 - in this paragraph, all line references are with respect to Alg. 2 and the references are by no means exhaustive), or attempt to contact a leader in the hierarchy (Lines 15 and 26). If nodes are within the attribute regions sought, they may simply flood the packet (Lines 17 and 28). The way packets are forwarded is dependent on whether the sensor is a cluster member or a cluster leader. In the former, often unresolved packets are forwarded to the cluster leader (Lines 40 and 57) while in the latter case, packets may be forwarded to a cluster leader, either a child cluster (Line 38) or an ancestor cluster (Line 59). If there are no known attributes among all specified, a flood throughout the network is performed (Line 61).

Algorithms 1 and 2 are expressed in higher level terms. In the specification of the routing rules lower level directives are used. Some examples of which are described below:

- **Communication directives**
  - *sendTo* - this takes as an argument a set of attribute value pairs and a formatted outgoing packet. It will attempt unicast tranmission between the host and the attribute region it wants to reach;
  - *floodIn* - this floods the outgoing packet. It will take as argument a node in the attribute hierarchy. This is the region within which to flood the packet. If the node selected is not one of the sensor's ancestor nodes, the packet is dropped.
- **Operators**
  - *isAdjacent* and *isContained* ($\subset$) - these are specified in the attribute hierarchy specification file (see appendix .1).
  - relational operators ($<$,$\leq$,$>$,$\geq$) - the "order" of string values is determined by the order of their appearance in the attribute hierarchy specification file.
  - equality operators ($==$, $!=$) - evaluated by string comparison or numeric comparison.
- **Flow Control** - the *if-else-if* expression is also supported for flow control. There can be an arbitrary number of "else-if"s that follow an "if."
- **Application Cluster control** commands:
  - *AppFormCluster* - this command forms the application clusters. The specification of the cluster obeys the following format: the cluster name, followed by "{", and then a series of "{ <member-name> : (<attribute name 1>, <attribute value 1>, ..., (<attribute name N>, <attribute value N>) }," separated by commas and followed by "}";
  - *AppClusterSendTo* - this commands takes as argument the application cluster name, the member name the packet must be sent to, and the formatted outgoing packet;
  - *AppClusterFloodTo* - this commands takes as argument the application cluster name and the formatted outgoing packet. The packet is flooded to all cluster members.

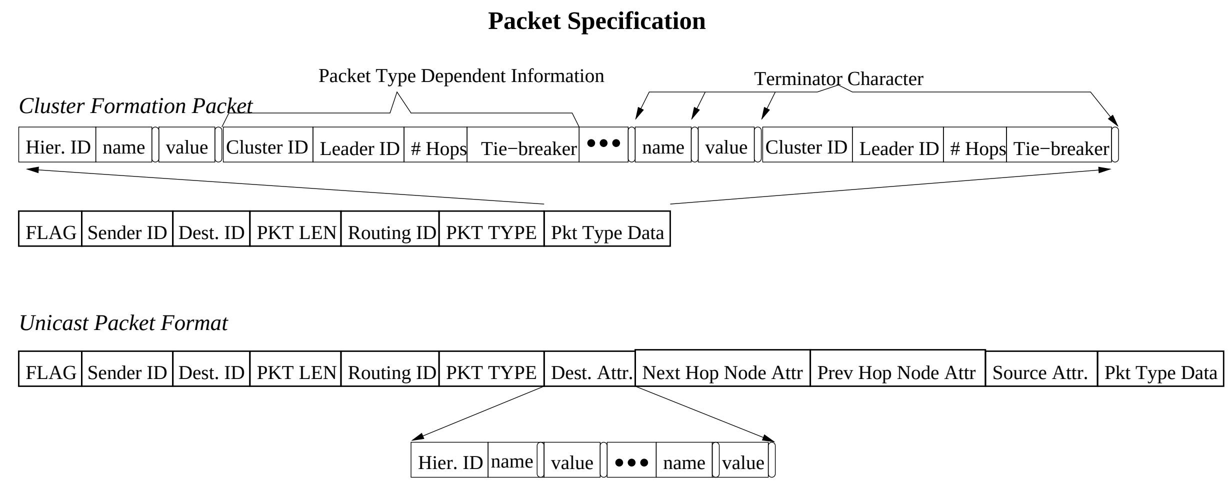

Figure 7: Packet specification format for cluster formation packet and unicast packets

- **Routing Data access** commands:
  - *NumberAttrHierarchy* - returns the number of hierarchies the routing process is aware of;
  - *AttrHierarchyAt* - takes a number as argument and returns the corresponding name of the attribute hierarchy stored by the routing process;
  - *NodesInAttrHierarchy* - returns the number of nodes in an attribute hierarchy
  - *NodeAtAttrHierarchy* - takes an attribute hierarchy name and a number as arguments and returns the corresponding node. The order by which nodes are listed is in a breadth-first-search manner as stored in the routing table. Alternatively, instead of the number, it takes a string composed of: $\{$attribute name 1, attribute name 2,...,attribute name $N\}$, in which attribute name 1 is a root node in the hierarchy, attribute name $i+1$ is a child of attribute name i ($1 \leq i < N$) and attribute name N is the attribute name of the node sought. The returned value is a string composed of; $\{$ i, (attribute name 1, ..., attribute name N), (attribute value N_1, ..., attribute value N_M) $\}$, that is, the order of the node in the node list, the sequence of attribute names from the root node to the node itself, and a sequence of possible M values the node has that has been seen by the routing process.
  - *NodeInstanceAtAttrHierarchy* - like *NodeAtAttrHierarchy* but looking for a specific instance of a node. Takes in as arguments the attribute hierarchy and a string composed of: $\{$(<attribute name 1>, <attribute value 1>), ..., (<attribute name N>, <attribute value N>)$\}$. Returns two numbers (i,j) in which i indicates the order of the node in the node list, and j the order of the node value given the node. Negative numbers indicate that the sought element was not found.
  - *ChildrenNodesOf* - this command returns the children of a node. This node may be specified either through $\{$<attribute hierarchy name>,(<attribute name 1>, ..., <attribute name N>)$\}$, or through attribute hierarchy name and a number (see *NodeAtAttrHierarchy* for how to interpret the number and the sequence of attributes). The list returned is a string composed of $\{$ $\{$(attribute name 1, i_1, (attribute value 1_1),...,(attribute value 1_{N_1})$\}$, ..., $\{$attribute name M),

i_M, (attribute value M_1, ..., attribute value $M_{N_M}\}$ $\}$. That is, a list composed of the M children nodes the specified node possesses, their indices in the node list, together with the known values associated with each child node.

- *ChildNodeAt* - this command takes as an argument a node in the hierarchy and two numbers (i,j). See item *ChildrenNodesOf* for how to specify the node in the hierarchy. The first number is the i^{th} child node while the second number is the j^{th} possible value that that specific child node possesses. The order of the child node (specified by i), as well as the order of the possible value (specified by j) are as in the string returned by *ChildrenNodesOf*.
- *ParentNodeOf* - this command takes as an argument: (1) an attribute hierarchy and (2) either a node specification or a number (see *ChildrenNodesOf* for an explanation of the number and how to specify a node). It returns the specified node's parent in the following format: $\{$ (attribute name 1, ..., attribute name N), i, (attribute value N_1, ..., attribute value N_M) $\}$, that is, the sequence of attribute names that goes from a root node (attribute name 1) through children nodes (attribute name $i+1$ is child node of attribute name i, $1 \leq i < N$) all the way to the parent node (attribute name N), and then all the M possible values of the parent node that the routing process has seen.
- *ClustersOf* - this command takes in as an argument (1) an attribute hierarchy and (2) either two numbers (i,j) or a node instance specification (see *NodeInstanceAtAttrHierarchy* for an explanation of what the two numbers mean and how to specify a node instance). It returns a set composed of all known cluster IDs of the node instance.
- *AdjacentClusterOf* - this command takes in as an argument (1) an attribute hierarchy, (2) either a node instance specification or a pair of numbers (see *NodeInstanceAtAttrHierarchy* for an explanation), and (3) a cluster ID as returned by *ClustersOf*. It returns a set of cluster IDs of adjacent clusters.
- *ParentClusterOf* - this command takes in as an argument (1) an attribute hierarchy, (2) either a node instance specification or a pair of numbers (see *NodeInstanceAtAttrHierarchy* for an explanation), and (3) a cluster ID as returned by *ClustersOf*. It returns the parent cluster's ID.
- *ChildrenClusterOf* - this command takes in as an argument (1) an attribute hierarchy, (2) either a node instance specification or a pair of numbers (see *NodeInstanceAtAttrHierarchy* for an explanation), and (3) a cluster ID as returned by *ClustersOf*. It returns the following string: $\{$ (attribute name 1, (attribute value 1_1, (cluster ID $1_{1,1}$, ..., cluster ID $C_{1,1}$)), ..., (attribute value M_1, (cluster ID $1_{1,M}$, ..., cluster ID $C_{1,M}$))), ..., (attribute name N, (attribute value 1_N, (cluster ID $1_{N,1}$, ..., cluster ID $C_{N,1}$)), ..., (attribute value M_N, (cluster ID $1_{N,M}$, ..., cluster ID $C_{N,M}$))) $\}$. That is, a list composed of children nodes' names, together with all possible values each name possesses, and seen clusters of each child instance.

- **Handlers**
  - *Self* - gives a handle for the application to refer to the sensor it belongs to.
  - *IncomingPacket* - accesses the current incoming packet being processed.
  - *OutgoingPacket* - handle through which the application may format an outgoing packet in the way it desires.

We show next our packet format specification.

4.1.5 Packet Formats

Packets in our routing scheme are composed of multiple fields. Fig. 7 brings the specification for the cluster formation type of packet, and packets for unicast communications. Cluster formation packets are flooded to the whole network, and brings with them information regarding the clusters that are being formed, while unicast packets bring specification of the destination, source and in-between node addresses.

1. *[Flag]* – the first field is used for flagging. We specify one byte, and the bits have the following meaning:

   (a) bit 1 – existence of destination hierarchy ID (see appendix .1).

   (b) bit 2 – set if intra-hierarchy routing.

   (c) bit 3,4 – specify which routing rules to use. If bits are

      00 use the default application independent routing rules set;

      01 indicates an application independent routing rules set but one other than the default;

      10 indicates an application dependent routing rules set

      11 indicates an application formed intra-cluster routing rules set

   (d) bit 5 – existence of source attribute based address

   (e) bit 6 to 8 – unused.

2. *[Sender ID]* – specifies the link layer's sender's hardware address.

3. *[Dest ID]* – specifies the link layer's destination's hardware address. If set to a specific sensor, then it is the "unicast" option, otherwise, all sensors within range receive the packet ("broadcast").

4. *[Pkt Len]* – specifies the length of the packet, in bytes.

5. *[Routing ID]* – this field brings the ID for the routing rules set used. The IDs for application independent routing rules set are integer numbers, while IDs for application dependent routing rules set must bring with it the sensor's hardward ID and an application specified identifier (name or process number).

6. *[Pkt Type]* – the various types of packets exchanged in our attribute based routing scheme. It contains all the clustering formation packet types, catalog exchange/building, plus data exchange packet types.

7. *[Dest Attr]* – this field specifies the attributes of the intended destination. It may initially bring with it the Attribute Hierarchy ID of which the attributes are part of (in which case bit 1 of the flag byte will be set). A special null byte is the terminator character that separates the Hierarchy ID and the name and the value fields of each attribute (see Fig. 7).

8. *[Next Hop Node Attr]* – this field has the attributes of the neighbor node in the attribute hierarchy to which the packet is intended. Its format is the same as *[Dest Attr]*. When bit 2 is set, the field does not have the attribute hierarchy subfield and assumes the same hierarchy as the one found under *[Dest Attr]*.

9. *[Prev Hop Node Attr]* – this field has the attributes of the neighbor node in the attribute hierarchy from which the packet came. Its format is the same as *[Next Hop Node Attr]*.

10. *[Source Attr]* – this field's presence in the packet is indicated by having bit 5 of the flag byte set. Its format is the same as *[Next Hop Node Attr]*.

11. *[Pkt Type Data]* – this field varies according to the type of the packet. Fig. 7 shows the contents for a `CLUST_FORMATION` packet, with fields for Attribute Hierarchy ID, attribute name-value pair, cluster leader, hop count, and tie-breaker information for leader election mechanism.

In our next section we present an example of a sensor network deployed for two applications. We then present an analysis of communication and storage costs associated with our packet and routing specifications for the example in Sec. 5.1.

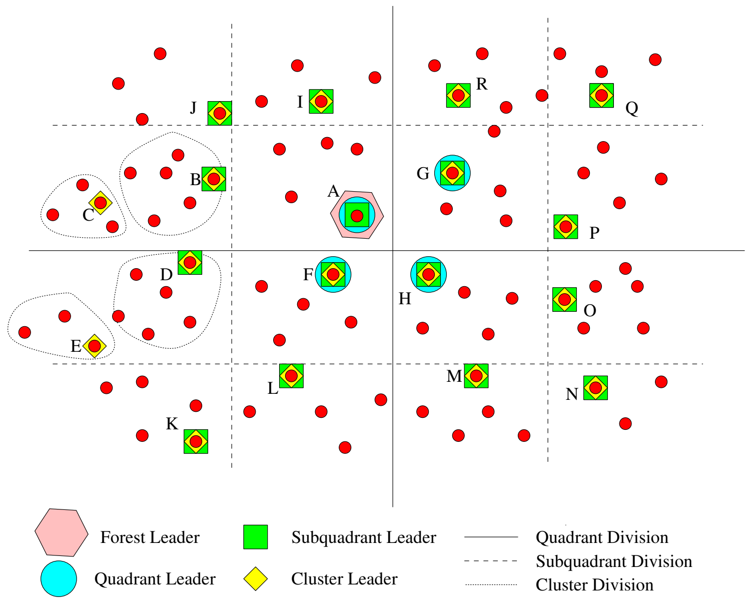

Figure 8: Example

5 Example

The example we will study is the joint climate monitoring and fire warning applications depicted in Sec. 2. The deployed sensor network example we will study is illustrated by Fig. 8 while the representative CDAG is the 3 level single child "line" structure shown in Fig. 1. We discuss next all the aspects necessary to support these two communication patterns simultaneously and present some performance analysis.

Sensors in the field need initially to be tagged with appropriate attributes, including location oriented ones. For outdoor applications, a GPS capable device can be used to communicate the correct geographical coordinates to a sensor before it is deployed, while in indoor applications, a similar device with pre-assigned human readable location attributes may be used, that is, the device would imprint "Quadrant=NE" or "Subquadrant=NE" tags onto the sensors. These attributes that are being tagged prior to their deployment are considered core attributes. We assume that once deployed, it will not be necessary (nor feasible) to update these core attributes. Note that derived attributes may still be added after deployment, e.g., sensors with “Quadrant=SW,Subquadrant=SW” and “Quadrant=SW,Subquadrant=SE” are assigned the “Lake=Walden” attribute. We call these derived attributes dynamic attributes.

We discuss in the appendix practical considerations of attribute tagging. For now, we assume sensors are tagged with core attributes, and all sensors know the different relationships among the attributes (e.g., containment and adjacency relationships). Some sensors also have the full name-value information of all possible core attributes, while other sensors only know the name-value information of attributes with which it had been tagged (these sensors may not become cluster leaders).

Thus in the climate monitoring and fire tracking example considered above sensors deployed have the following attributes:

- Name: X, Values: $x_{min} \leq X \leq x_{max}$
- Name: Y, Values: $y_{min} \leq Y \leq y_{max}$
- Name: $Forest$, Values:“Lorien”
- Name: $Quadrant$, Values: NE, NW, SE, SW
- Name: $Subquadrant$, Values: NE, NW, SE, SW

The C-DAG is represented by the three last attributes and form a line: $Subquadrant \subset Quadrant \subset Forest$. From the C-DAG the containment relationships follow, which are:

- Subquadrant $\subset$ Quadrant
- Quadrant $\subset$ Forest

Adjacency relationships are defined separately (see appendix .1) and can be expressed as:

- Quadrant
  - NW adjacent NE, NW adjacent SW, NE adjacent SE, SW adjacent SE
- Subquadrant (the adjacency relationships below concern two subquadrants S1 and S2 - subquadrant adjacency relationships are conditional on the adjacency of the quadrants):
  - NE adjacent NW, SE adjacent SW, S1 Quadrant $\in$ NW, S2 Quadrant $\in$ NE
  - SE adjacent NE, SW adjacent NW, S1 Quadrant $\in$ NW, S2 Quadrant $\in$ SW
  - SE adjacent NE, SW adjacent NW, S1 Quadrant $\in$ NE, S2 Quadrant $\in$ SE
  - NE adjacent NW, SE adjacent SW, S1 Quadrant $\in$ SW, S2 Quadrant $\in$ SE

The relationships above assume that the sensor knows that adjacency rules are commutative (i.e., if Q_1 adjacent to Q_2, then Q_2 is adjacent to Q_1).

Communication among the sensors follow two patterns:

1. The “tree” like pattern, in which lower level sensors communicate their data to their cluster leaders, and these in turn forward the collected information to their upper level leaders. This communication pattern is used by the climate monitoring application and;

2. The “mesh” like pattern, in which lower level clusters send packets to their adjacent (same level) clusters. This communication pattern is used by the fire detection/warning application.

The two rules set can be described by Algorithms 1 (Sec. 4) and 3.

The Mesh traversal algorithm, unlike the Tree traversal (Alg. 1) one, drops packets that have been seen before (Line 8 of Alg. 3). In the tree traversal, unknown destination packets may be sent to higher level cluster leaders (Line 24 of Alg. 1), and these may eventually forward the packets back (Line 27 of Alg. 1). The Mesh traversal algorithm forwards packets of unresolved attributtes to neighbor clusters (Line 21 of Alg. 3). Notice that the difference in routing rule differs only on the resolution of unknown attribute based addresses. While in the tree case the packets are forwarded up the hierarchy level, in the mesh the packets are simply spread towards other adjacent clusters. These two resolution modes also characterise the intrinsic communication pattern each rules set supports. Sensor networks that are deployed for different applications will benefit from being able to support switching between the two modes, as we will show in the next section.

5.1 Performance

In this section we will show through theoretical analysis the advantages of having support for multiple routing schemes. Consider the C-DAG shown in the center of Fig. 1. It represents a “line” attribute hierarchy. This hierarchy can be used by applications to send data through the network in a “tree” traversal mode, by going “up” and “down” the hierarchy through cluster leaders at different levels, or to send data in a “mesh” traversal mode, by going only to adjacent clusters at the same hierarchical level. Both possibilities are illustrated in the right side of Fig. 1.

We will study the performance of schemes that use a hierarchical clustered approach and route data as if on a “tree” or on a “mesh,” and of schemes that rely on “flooding” for data propagation, as well as schemes that have full knowledge of all sensors in the network. The network is consisted of N sensors spread uniformly over a square region of area L^2 and there are l_h levels in the line attribute hierarchy. Therefore, since the C-DAG representation of the attribute hierarchy is a line, there are l_h nodes in the C-DAG. The root node (at level 1) in the C-DAG covers the whole region, while subsequent nodes (at levels l_i, $i \in \{2, \dots, l_h\}$) have four possible values each (a quadtree format), with each value covering a square region of side $L/2^{(i-1)}$. In the right of Fig. 1 a three level C-DAG is shown.

The metrics we will be studying for each scheme include: (1) total memory requirement from all nodes for implementation; (2) the estimated number of transmissions taken when routing one packet from a source to an unknown destination in the worst case (considering that the sensors are deployed over a square region, the worst case is when source and destination lie at opposite corners across a diagonal) and (3) the estimated number of transmissions that separates source from destination. The difference between (2) and (3) is that the former takes into account all transmissions triggered by routing the packet, while the latter counts only the estimated number of transmission taken specifically to deliver the packet from source to destination. In a sense, (3) is an estimate of the hop distance a packet traveled from a source before it reaches the intended destination. We assume that the average maximum delay is proportional to the estimates in (3) in the absence of concurrent traffic.

When estimating the number of transmissions triggered or taken to deliver the packet, i.e., items (2) and (3) above, for non-flooding type of schemes, we consider that the path the packet takes is composed of consecutive straight line segments. One estimate of the number of transmissions is the product of the length of the segment by the linear node density. The node density is given by $\rho = N/L^2$, thus one estimate of the number of neighbors that lie on a line segment within transmission radius R is $R\sqrt{\rho}$. On the average, assuming the sensors are uniformly distributed and the whole network connected, the number of transmissions should not be greater than this value, for this value reflects the number nodes that lie in the segment[2]. If this value is $\gg 1$, then we are overestimating the number of transmissions needed. Estimates made in this way can still be used for comparison between different routing schemes, though, since the overestimation comes from the high node density value and will be reflected by all routing schemes.

An estimate that is closer to the minimum number of transmissions needed to cover the path between source and destination is obtained by dividing the path length by the transmission range R. However, when the "line" C-DAG has a very high number of nodes (i.e., high l_h), the leaf node's covered region may be smaller than the transmission range ($L/2^{(i-1)} \ll R$, when $i \gg 1$). Because our hierarchical routing scheme stores routing information based on attribute regions, and routes according to containment and adjacency relationships, the lower bound in the number of transmissions is the number of attribute regions traversed.

The results of our performance comparison are summarized in Table 1.

Table 1: Performance Metrics for different Routing Schemes

		Flooding	Full	Tree (One level information)	Tree (Full cluster information)	Mesh
Memory		0	$E\,N(N-1)$	$E\,\frac{4}{3}\,(4^{(l_h-1)}-1)+E\,N\,l_h$	$E\,2\,N\,l_h$	$E\,(4^{l_h}+N+2(2^{(l_h-1)}-1)\sqrt{N})$
Num Tx	Max	N	$\sqrt{2N}$	$\sqrt{N}\,(2^{(l_h-1)}-1)(\frac{2\sqrt{2}}{2^{(l_h-1)}}+\frac{3\sqrt{2}}{2}+\sqrt{5})+\frac{N}{2^{(2l_h-2)}}$	$4\sqrt{2N}(1-\frac{1}{2^{(l_h-1)}})+\frac{N}{2^{(2l_h-2)}}$	$2\sqrt{2N}(2^{l_h}-\frac{2}{2^{(l_h-1)}})+\sqrt{N}(\frac{8-4\sqrt{2}}{2^{(l_h-1)}})+\frac{N}{2^{(2l_h-2)}}$
	Min	N	$L\sqrt{2}/R$	$\max(\frac{L}{R}\,(2^{(l_h-1)}-1)(\frac{2\sqrt{2}}{2^{(l_h-1)}}+\frac{3\sqrt{2}}{2}+\sqrt{5}), \sum_{i=2}^{l_h}(4^{(i-2)}\lceil\frac{L\sqrt{2}}{R2^{(i-1)}}\rceil+4^{(i-2)}\,2\lceil\frac{L\sqrt{5}}{R2^{(i-1)}}\rceil+(4^{(i-2)}+1)\lceil\frac{L\sqrt{2}}{R2^{(i-2)}}\rceil))+\frac{N}{2^{(2l_h-2)}}$	$\max(\frac{4L\sqrt{2}}{R}(1-\frac{1}{2^{l_h-1}}), \sum_{i=2}^{l_h}2\lceil\frac{L\sqrt{2}}{R2^{(i-2)}}\rceil)+\frac{N}{2^{(2l_h-2)}}$	$\max(\frac{L2\sqrt{2}}{R}(2^{l_h}-\frac{2}{2^{(l_h-1)}})+\frac{L(8-4\sqrt{2})}{R\,2^{(l_h-1)}}, 2^{l_h}\,(2^{(l_h-1)}-1))+\frac{N}{2^{(2l_h-2)}}$
Num Hops	Max	$\sqrt{2N}$	$\sqrt{2N}$	$4\sqrt{2N}(1-\frac{1}{2^{(l_h-1)}})$		$2\sqrt{2N}(1-\frac{1}{2^{(l_h-1)}})+\frac{\sqrt{N}(4-2\sqrt{2})}{2^{(l_h-1)}}$
	Min	$L\sqrt{2}/R$	$L\sqrt{2}/R$	$\max(\frac{4L\sqrt{2}}{R}(1-\frac{1}{2^{l_h-1}}), \sum_{i=2}^{l_h}2\lceil\frac{L\sqrt{2}}{R2^{(i-2)}}\rceil)$		$\max(\frac{L}{R}(2\sqrt{2}(1-\frac{1}{2^{(l_h-1)}})+\frac{4-2\sqrt{2}}{2^{(l_h-1)}}), 2\,(2^{(l_h-1)}-1))$

Flooding A flooding based routing scheme does not need to store any routing information about the network. Every packet is flooded to the whole network. Consequently, the memory requirement is zero[3] and it takes N transmissions to deliver the packet. The farthest any two sensors may be from each other is if they lie at opposite corners across a diagonal. Thus, transmission across the diagonal will take a minimum of $L\sqrt{2}/R$ and if the node density of $\rho = N/L^2$, then an estimate of the number of nodes lying

[2]We are assuming the routing scheme will not present as a rule a sharp zigzag pattern while routing packets, but instead will attempt to route packets around the segment.

[3]Diffusion schemes are not purely flooding schemes, since Diffusion remembers paths to published source/sink.

in the diagonal is $L\sqrt{2\rho} = \sqrt{2N}$, and this is, on the average, the maximum number of transmissions needed to send the packet from source to destination.

Full Knowledge A routing scheme that stores next hop routing information of all nodes in the network has a huge memory requirement. In fact, each node needs to store information about $N-1$ other nodes in the network. Considering that each routing entry requires E bytes, the total memory requirement in the network is $E\,N(N-1)$. However, because of the complete knowledge, the number of transmissions triggered and the number of transmissions needed to send the packet are equal. These are equal to the estimated maximum and minimum number of hops in the flooding case.

Cluster Flooding In both *Flooding* and *Full Knowledge* schemes destination sensors are sure to be reached. In Tree or Mesh schemes below, however, packets reaching the intended leaf cluster(s) still need to reach the sensors. Assuming the intended destination address resolves into one leaf cluster, to flood that cluster the number of additional transmissions is equal to $\rho\,(L/2^{(l_h-1)})^2 = N/2^{(2l_h-2)}$ is needed. This term appears in all "NumTx" entries in Table 1.

Tree (One level information) In our clustered hierarchical scheme, each node that is not cluster leader tracks leaders of the cluster it belongs across all hierarchy levels ($E\,N\,l_h$). Assuming one cluster per attribute value, we have one cluster for the root node, four clusters for the node at the second level, 16 for the node at the third level, etc. Each cluster leader tracks the routing information of its four children clusters. Leaf cluster leaders track information about their cluster members. Since leaf clusters cover the whole network, it requires N entries. Thus it is $E4(1+4+4^2+...+4^{l_h-2})+EN = E4(4^{l_h-1}-1)/3+EN$. The sum of the two factors shown in this paragraph is the memory requirement equation for "Tree (One level information)" in Table 1.

When a packet with an unknown destination is received, it is sent to the cluster leaders through the hierarchy all the way up to the root node if no matching attributes are found. The longest segment that separates the root to a second level cluster leader is $L\sqrt{2}$, while the longest segment that separates the second level cluster leader to a third level child cluster is $L\sqrt{2}/2$. Thus the sum of the segment lengths is at most $U_L = L\sqrt{2}(1 + 1/2 + \cdots + 1/2^{(l_h-2)}) = L2\sqrt{2}(1 - 1/2^{(l_h-1)})$.

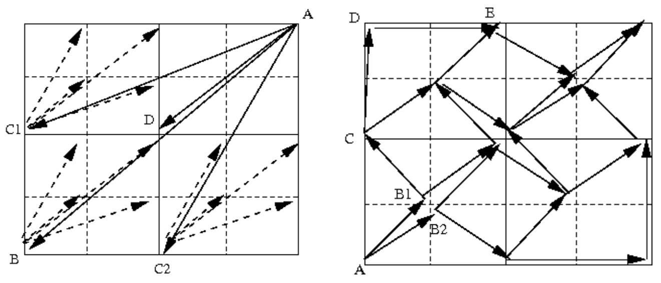

Figure 9: (a) *Left:* Propagation path for *Tree* traversal when resolving unknown destination address (b) *Right:* Propagation path for *Mesh* mode when resolving unknown destination address

Also, assuming the packet reaches the root node, the root node must send down the packet to all its child clusters, which may in turn pass it down again, all the way to the leaf cluster leaders, at which point the cluster leader that satisfies the destination address attributes floods the packet to its cluster. In this process of forwarding the packet down the C-DAG, from the root node to the second level cluster leaders four segments (covering the longest distance possible) are needed: the longest will be $L\sqrt{2}$ (segment AB in Fig. 9(a)), while there will be two segments of $L\sqrt{5}/2$ (segments $AC1$ and $AC2$ in Fig. 9(a)), and one segment of $L\sqrt{2}/2$ (segment AD in Fig. 9(a)). From the second level cluster leader to the third level clusters the same process will be repeated: the packet is sent to four clusters, with the longest segment being half of the longest segment of the previous level, two segments which are half of the two analogous segments of the previous level, and the shortest segment being half of the shortest in the previous level (shown as dotted lines starting from $B, C1, C2$ - omitted for D for clarity's sake). This process repeats itself all the way down to the clusters at level $l_h - 1$.

Thus, assuming $S = (L\sqrt{2} + 2L\sqrt{5}/2 + L\sqrt{2}/2)$, then the total segment length the packet may need to traverse when going down is $D_L = S + 4S/2 + 16S/4 + \cdots + 4^{(l_h-2)}S/2^{(l_h-2)} = S + 2S + \cdots + 2^{(l_h-2)}S = S(2^{(l_h-1)} - 1) = L(2^{(l_h-1)} - 1)(3\sqrt{2}/2 + \sqrt{5})$. The total length is then $T_L = U_L + D_L = L(2^{(l_h-1)} - 1)(2\sqrt{2}/2^{(l_h-1)} + 3\sqrt{2}/2 + \sqrt{5}$.

Of the four packets that are sent from a higher level leader to a lower level leader only one eventually reaches the destination. So that we will estimate the number of hop that can cover the worst case (i.e., sender and receiver are farthest apart), we take the longest segment at each level, and thus $T_L = 2U_L = L4\sqrt{2}(1 - 1/2^{(l_h-1)})$.

The higher estimate on the number of transmissions is obtained by multiplying the length obtained by $\sqrt{N}/L$ (see "NumTxMax" and "NumHopsMax" equations in Table 1), while an estimate on the minimum number of transmissions is obtained by dividing the length by R (the first term of the max function for "NumTxMin" and "NumHopsMin" in Table 1).

However, in the case in which the transmission range is much higher than the leaf attribute region side, the number of transmissions is lower bounded by the number of attribute regions the packet crosses. Given that there are four different segments that the root node needs to send to reach the level 2 leaders, each of the two segments of equal length ($AC1$ and $AC2$ in Fig. 9(a)) will generate $4^{(i-2)}$ segments of length $L\sqrt{5}/(R2^{(i-1)})$ at level $i \geq 2$. In the same way we count $4^{(i-2)}$ segments for the shortest segment (AD in Fig. 9(a)) and $4^{(i-2)} + 1$ for the longest. The "+1" is because we must also count the transmission costs incurred when the packet was coming up the hierarchy towards the root node. Each segment counts at least once (i.e., we round up the cost) no matter how small its length is with respect to the transmission radius R, because it represents one distinct attribute region. When we sum up for all levels in the hierarchy we obtain the second term in the max function for "NumTxMin" and "NumHopsMin" in Table 1.

When we consider the number of hops that can separate source from destination, the worst case is if the source and destination eventually are resolved by going through the longest segment across all levels of the hierarchy. This is represented by the corresponding equation in Table 1.

Tree (Full cluster information) For a tree scheme in which cluster leaders track all information from its cluster members the following memory requirement is necessary for a cluster leader at level i: $\rho\,(L/2^{(i-1)})^2 = N/2^{(2i-2)}$. Since this is a quadtree format, there are exactly $4^{(i-1)}$ children cluster leaders at level i, thus the memory requirement for cluster leaders to track cluster member information is exactly $l_h N$. Adding this to the requirement of N nodes tracking their l_h cluster leaders, the total memory requirement is $2l_h N$, as seen in Table 1. For the Tree traversal mode with full cluster information, it is not necessary for the root node to forward the packet down to all of its children clusters. Since it has information of all the sensors in the network, it can forward the packet to the child cluster that contains the desired destination attributes. Thus the number of transmissions and the number of hops in this case is the same, and it corresponds to the case in which the longest segment is taken both when the packet is coming up the hierarchy and going down the hierarchy to the destination leaf cluster.

Mesh We study the performance of a routing scheme in a mesh like topology at only one attribute hierarchy level (say l_h). In a mesh like routing scheme, we assume each cluster leader tracks only its (at most) four neighbor clusters, resulting in a memory requirement of $E\,4\,4^{(l_h-1)}$. Also all sensors track their cluster leader ($E\,N$ of memory), and sensors that lie at the attribute border will track the two clusters for which it is the border. At level l_h, the total length of the border is $2(2^{(l_h-1)} - 1)L$, which, when multiplied by $\sqrt{N}/L$ and summed with the other terms, results in the memory requirement equation seen in Table 1.

We assume that when a packet with an unknown destination is received it will be transmitted to the neighbor clusters other than the ones from which the packet arrived. Thus if a packet is sent from the lower left cluster leader, with a destination that is unknown to the cluster leader, but whose final sink is in the top right cluster, then the packet will be propagated across all attribute regions. The total length traversed as the packet is distributed in the network is longer if the cluster leaders are located close to opposite corners across the diagonal, in the zigzag pattern shown in Fig. 9(b). In this figure we show the traversal taken when there are three nodes in the line C-DAG. The cluster leader of the lower left attribute region (A) sends the packet to its immediate neighbor cluster leaders ($B1$ and $B2$). As these are located close to the corner across the diagonal the length traversed is $2L\sqrt{2}/2^{(l_h-1)}$. To increase the length traversed, as the packet gets closer to the top left and bottom right corners, we assume the cluster leaders are located at the corners of their respective attribute regions. In this way we force comparison of the worst case in a mesh approach with the worst case of the tree based scheme analyzed previously. Notice that essentially the packet traverse the diagonals of squares with side length $2jL/2^{(i-1)}, j \in \{1, 2, 3, ..., 2^{(i-1)}\}$ in a regular fashion, discounting the borders and the top left and bottom right corners. The total length traversed, and the corresponding expected number of transmissions (both maximum and minimum) are given by the corresponding expressions in Table 10.

When the transmission radius $R \gg L/2^{(i-1)}$, then it takes at least one transmission to cross one attribute region, and assuming each attribute region will transmit to two of its immediate neighbors (with top and right border attribute regions transmitting only once), the total number of transmissions will be $2(2^{(l_h-1)} - 1) + 2(2^{(l_h-1)} - 1)^2 = 2^{l_h}(2^{(l_h-1)} - 1)$, as seen in the table.

The shortest path that separates the source from the destination must traverse $2(2^{(l_h-1)} - 1) + 1$ attribute regions (the $+1$ is because the source attribute region also must be traversed). However, if the packet goes through only the diagonals, only $2(2^{(l_h-1)} - 1)$ diagonals need be crossed. One of the attribute region leaders will receive the packet from the left and can immediately forward to the upper region, without needing to traverse itself. Thus the worst case scenario is actually when the source is at the top left corner while the destination is at the bottom right corner (or vice-versa). In this case there are additional four traversals across the border of the attribute region ($4(L/2^{(l_h-1)})$) and two less diagonal traversals. This explains the second term in the “NumHopMax” and the second term in the first argument to the $\max$ function in “NumHopMin.” When we are considering the minimum number of hops, this must be lower bounded by the number of attribute regions that need be crossed ($2(2^{(l_h-1)} - 1)$), since in principle the cluster leader only tracks the four adjacent clusters. We show some plots of the equations of Table 1 in Fig. 10.

We can see from Fig. 10(a) and Fig. 10(b) that the expected number of transmissions to resolve an unknown address in the worst case is higher for the Mesh traversal mode than for the Tree cases. In fact, when cluster leaders track full cluster information, the performance dramatically improves. This is because the root node need not propagate the packet with unknown address down to all of its children clusters. We can see that the high number of levels in the attribute hierarchy contributes to the inefficiency of the process (Figs. 10(b) and 10(e)). With the increase in the number of hierarchies, the packet with unknown destination address need essentially be distributed to the whole network in the Mesh and Tree (with one level information) schemes at increasing levels of granularity (i.e., covering more of the network), contributing to their performance degradation.

A high number of levels will involve transmission costs to cross adjacent clusters in the Mesh case and costs to resolve all the way to the leaf cluster in the Tree (one level info) case. These costs surpass those of the mere flooding schemes and should be avoided. The cost for resolving an unknown address in the

Tree (full cluster info) case remains constant. However, the memory requirements are high (Figs. 10(c) and 10(f)).

When we consider the number of hops metric, we find that Mesh schemes are able to find shorter paths between source and destination (Figs. 10(g) and 10(h)). The only drawback is that Mesh schemes currently cross only spatially adjacent attribute regions. Thus when the number of levels in the hierarchy increases, there is a corresponding increase in the hop distance (Fig. 10(h)).

From the graphs in Fig. 10 we can see that if the network is composed of heterogeneous nodes, in which some nodes have higher capacity, then a Tree (full cluster info) scheme will be the most economical in transmission costs related to address resolution issues. Sensor networks that have a high inquiry arrival, especially from a large user base, will benefit from the increased savings in Tree based address resolution schemes, while applications that require fast response can invoke Mesh traversal mode for their data packets.

6 Conclusion

In this technical report we presented examples of different applications being tasked to the same sensor network simultaneously. Due to the different objectives of the applications, their underlying data communication and dissemination patterns favor different routing schemes. We envision that sensor networks will be widespread in the future. and to tap into the full potential of such sensorsphere, the underlying routing infrastructure must support dynamic routing scheme selection. With this feature, tasked applications can request routing support from a scheme whose packet forwarding rules match their data communication requirements, thus maximizing their performance.

In order to enable dynamic routing scheme selection, we propose using sets of routing rules that forward data in pre-defined ways as the elements to be selected at runtime. We assume that the underlying sensor network has been clustered according to a hierarchy of attributes, and that containment and adjacency relationships between the clusters (or the attributes) are clearly defined. We present in this paper two routing rules set for applications deployed in a sensor network with the above mentioned logical structure. One rules set implements Tree traversal mode while the other Mesh traversal mode. We show analytical performance results of the two traversal modes and show that Mesh traversal mode favors applications that need fast response, while Tree traversal mode has less transmission cost when resolving a previously unknown destination address.

Acknowledgment

This work is supported by the NSF under grant No. CNS-0435353. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] G. J. Pottie and W. J. Kaiser. Wireless Integrated Network Sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[2] J. Hill and D.E. Culler. MICA: A Wireless Platform For Deeply Embedded Networks. *IEEE Micro*, 22(6):12–24, Nov/Dec 2002.

[3] Rex Min, Manish Bhardwaj, Seong-Hwan Cho, Nathan Ickes, Eugene Shih, Amit Sinha, Alice Wang, and Anantha Chandrakasan. Energy-Centric Enabling Technologies for Wireless Sensor Networks. *IEEE Wireless Communications (formerly IEEE Personal Communications)*, 9(4):28–39, Aug 2002.

[4] Dan Li, Kerry Wong, Yu Hen Hu, and Akbar Sayeed. Detection, Classification and Tracking of Targets. *IEEE Signal Processing Magazine*, 19(2), March 2002.

[5] R. R. Brooks, P. Ramanathan, and A. M. Sayeed. Distributed Target Classification and Tracking in Sensor Networks. *Proc. of the IEEE*, 91(8), August 2003.

[6] Jeongyeup Paek, Nupur Kothari, Krishna Chintalapudi, Sumit Rangwala, Ning Xu, John Caffrey, Ramesh Govindan, Sami Masri, John Wallace, and Daniel Whang. The Performance of a Wireless Sensor Network for Structural Health Monitoring. In *2nd European Workshop on Wireless Sensor Networks*, Istanbul, Turkey, Jan 31 – Feb 2 2005.

[7] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proc. 1st ACM Intl. Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, pages 88–97, 2002.

[8] Rachel Cardell-Oliver, Keith Smettem, Mark Kranz, and Kevin Mayer. Field Testing a Wireless Sensor Network for Reactive Environmental Monitoring. In *International Conference on Intelligent Sensors, Sensor Networks and Information Processing ISSNIP-04*, December 2004.

[9] J. Lundquist, D. Cayan, and M. Dettinger. Meteorology and Hydrology in Yosemite National Park: A Sensor Network Application. In *Information Processing in Sensor Networks (IPSN)*, April 2003.

[10] Stargate Gateway. URL. `http://xbow.com/Products/productsdetails.aspx?sid=85`.

[11] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proc. 5th ACM MobiCom Conference*, Seattle, WA, August 1999.

[12] B. Karp and H. T. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of the 6th ACM MobiCom Conference*, pages 243–254, Boston, MA, August 2000.

[13] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks. Technical report, UCLA Computer Science Department Technical Report UCLA/CSD-TR-01-0023, May 2001.

[14] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash Table for Data-Centric Storage. In *Proc. 1st ACM Intl. Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.

[15] Xin Li, Young Jin Kim, Ramesh Govindan, and Wei Hong. Multi-dimensional Range Queries in Sensor Networks. In *Proc. 1st ACM Intl. Conference on Embedded Networked Sensor Systems (Sensys'03)*, Los Angeles, CA, USA, November 2003.

[16] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A Distributed Index for Features in Sensor Networks. In *Proc. 1st IEEE Intl. Workshop on Sensor Network Protocols and Applications (SNPA'03)*, 2003.

[17] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new Data Handling architecture for Sensor Networks? In *Proc. 1st Workshop on Hot Topics In Networks (HotNets-I)*, Princeton, NJ, October 2002.

[18] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proc. International Conference on Mobile Computing and Networking (MobiCom)*, Boston, MA, August 2000.

[19] D. Niculescu and B. Nath. Trajectory Based Forwarding and Its Applications. In *Proc. ACM MobiCom 2003*, San Diego, CA, USA, September 2003.

[20] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang. TTDD: A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks. In *Proc. International Conference on Mobile Computing and Networking (MobiCom)*, Atlanta, GA, September 2002.

[21] Hu Zhou and Suresh Singh. Content Based Multicast (CBM) in Ad Hoc Networks. In *Proc. 1st ACM International Symposium on Mobile Ad-hoc Networking & Computing (MobiHoc)*, 2000.

[22] David Branginsky and Deborah Estrin. Rumor Routing Algorithm for Sensor Netowrks. In *Proceedings of WSNA02*, 2002.

[23] M. Chu, H. Haussecker, and F. Zhao. Scalable Information-Driven Sensor Querying and Routing for ad hoc Heterogeneous Sensor Networks. *Intl. J. High Performance Computing Applications*, 16(3), 2002.

[24] Sundeep Pattem, Bhaskar Krishnamachari, and Ramesh Govindan. The Impact of Spatial Correlation on Routing with Compression in Wireless Sensor Networks. In *Symposium on Information Processing in Sensor Networks (IPSN)*, April 2004.

[25] Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy. The ACQUIRE Mechanism for Efficient Querying in Sensor Networks. In *IEEE International Workshop on Sensor Network Protocols and Applications (in conjunction with ICC 2003)*, May 2003.

[26] S. Madden, M Franklin, J. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor for Sensor Networks. In *Proc. ACM SIGMOD*, San Diego, CA, June. 2003.

[27] Antonio Carzaniga, Matthew J. Rutherford, and Alexander L. Wolf. A Routing Scheme for Content-Based Networking. In *Proc. IEEE INFOCOM'04*, Hong Kong, China, March 2004.

[28] Sam Joseph. NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks. In *Proceedings of the International Workshop on Peer-to-Peer Computing*, 2002. `http://www.neurogrid.net/NeuroGridSimulations_mod_b.pdf`.

[29] Arturo Crespo and Hector Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Computer Science Department, Stanford University. `http://www-db.stanford.edu/~crespo/publications/op2p.pdf`.

[30] S. Castano, A. Ferrara, S. Montanelli, E. Pagani, and G. P. Rossi. Ontology-Addressable Contents in P2P Networks. In *Proc. 1st Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID'03)*, pages 55–68, Budapest, Hungary, May 2003.

[31] H. T. Kung and C. H. Wu. *Content Networks: Taxonomy and New Approaches*. Santa Fe Institute series. Oxford University Press, 2002.

[32] Christian Prehofer and Qing Wei. Active Networks for 4G Mobile Communication: Motivation, Architecture, and Application Scenarios. In *Proc. IFIP-TC6 International Working Conference (IWAN'02)*, London, UK, 2002.

[33] Christian Tschudin, Henrik Gulbrandsen, and Henrik Lundgren. Active routing for ad-hoc networks. *IEEE Communications Magazine, Special issue on Active and Programmable Networks*, April 2000.

[34] S. Calomme and G. Leduc. Performance Study of an Overlay Approach to Active Routing in Ad Hoc Networks. In *Proc. 3rd Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2004)*, Bodrum, Turkey, Jun 2004.

.1 Attribute Tagging and Representation

Attributes are assigned to a sensor in a specific human language (e.g., English). While this is human readable, an attribute that is spelt "temperature" and has a string representation would need 11 bytes for storage and transmission. If we desire to limit the energy spent in transmission more than the cost of adding storage, one way to reduce the number of bytes required in transmission is to index the set of attribute names and their respective values.

The way the attributes are indexed is as follows: two files, one called "Attribute Name Value Index" (ANV-IDX) and the other called "Attribute Name Value Instance" (ANV-IST) will be used. ANV-IDX contains only the indices while ANV-IST contains the representation in a specific human language of the attribute's name and possible values. The first row of ANV-IDX has the index range (from 1 to N). The first column of ANV-IDX has the indices used for attribute names. The second column contains its type (integer, double, string), while subsequent numbers in each row represent the indices used for all possible attribute values associated. The file ANV-IST will contain in correspondent locations (i.e., at correspondent column, row positions) the specific representation of an attribute's name and possible values in a human language. Relational symbols ($<$, $\leq$, $>$, $\geq$, $=$, $\neq$) are used to represent a range for numeric attributes. If the type of attribute is string, the row containing its possible values is assumed ordered (i.e., the order of the possible values is the order of appearance in the row, not its lexicographical value).

A third file, called "Attribute Relationship Rules" (ARR) lists the dependency relationships between different attributes names (e.g. containment) or attribute values (e.g.,adjacency). Essentially ARR should contain (1) the C-DAG used to represent the attribute hierarchy and (2) other relationships that concern attribute values (e.g., adjacency relationships). These rules do not refer to any attribute in its full name, but only to the indices found in ANV-IDX. In this way the ARR rules do not depend on the specific language in which the attribute is specified. Also, if two attribute hierarchies share the same structure and same adjacency properties, the same ANV and ARR files can be used.

C-DAGs are described by multiple rows in the ARR, each row containing first the parent node, followed by the symbol $\supset$ and then the child node, followed by the symbol $\supset$ and then by the grandchild node, and so forth as decided by the writer of ARR, or until a leaf node is reached. For single root C-DAGs, the number of rows used to describe the C-DAG will be at least the same as the number of leaf nodes.

After the C-DAG is described, a single element line containing the index representing a node in the C-DAG is stored. The lines that follow this single element line describe the adjacency relationships of the attribute represented, until a new single element line is encountered, or the file terminates.

In the lines that describe adjacency relationships, each line is possibly split into two parts. The first part include pairs containing the indices of two attribute values that are adjacent. The two elements that compose the pair are separated by whitespace, while the pairs themselves are separated by commas. The second part of the line starts after the last pair, and it includes the values that attributes higher in the hierarchy must have in order for the adjacency conditions to be true.

In this second part, the keywords "S1" indicates the sensor that represents the first element in the pair, while "S2" indicates the sensor that represents the second element. The second part is composed of "sentences" split again by commas. Each sentence starts with "S1" (or "S2"), which is followed then by the index of a higher level hierarchy, ahd then by the symbol $\in$ (or $\notin$), and finally by the set of values the attribute indicated must have. If there are no conditions then the second part is blank.

All sensors deployed will have ARR stored. Together with the ARR file it will also be stored the hash code (e.g., applying MD5 algorithm to ARR) of the ARR file. If sensors being tagged with the core attributes are also storing the ANV files, then instead of storing the full name and value of each attribute, they will store only the indices of the attribute names and values as found in the ANV files. If, however, the sensors are not storing the ANV files, then the full name and value of the attributes, together with the indices used for them in the ANV files will be stored in the sensor. Also to be stored with the sensor is the hash number of the ANV-IDX file.

When sensors transmit packets, they have two options: use the full string representation of the attributes' name and values stored, or use the encoded format. Essentially the encoded format consists of sending initially the hashcodes of ARR and ANV-IDX files, followed then by the indices of the attributes descriptive of the destination. The hashcodes for ARR and ANV-IDX guarantee that nodes that have the same hashcodes for ARR and ANV-IDX will act in a consistently equal manner.

By not using the hashcode for ANV-IST we are allowing sensors that have the same attributes but using different language representations to communicate with one another. If we adopt a "default" language for specifying ARR files, then we can also support ARR files that are written in different languages. Without agreement on what the "default" language should be, ARR files that are written in different languages will yield different hashcodes and sensors cannot exchange packets, even though both ARR files describe the same set of relationships.

Sensors that do not have full ANV files but which receive a packet with unknown attributes will forward the packet to their cluster leader. Sensors that have full ARR and ANV-IDX and ANV-IST files that receive packets with unknown attributes or unmatched hashcodes will simply rebroadcast the packet when first received, and dropped if heard before.

Foreign sensors which do not share the same ANV and ARR files will simply list the desired attributes (names and values) in the full string representation and broadcast. If no response is received after a threshold number of requests, the sensor may request to receive any new ANV-IDX and ARR files from neighboring nodes.

When new dynamic attributes are introduced, with their names and range of values, the node initiating the update assigns indices to the new names and values. Nodes receiving the new attributes store them in "Attribute Name Value Index Dynamic" (ANV-IDX-Dyn) and "Attribute Name Value Instance Dynamic" (ANV-IST-Dyn) files. The order in which the attributes are stored is based on the indices assigned (lower indices stored first). The hash function must allow for concatenation, and the original hashcode for ANV-IDX is the starting point for the hash function applied to ANV-IDX-Dyn. Subsequent packets will bring this resultant hashcode in their headers. Since all sensors have ARR files, any updates sent will be received by all sensors and a new hashcode for ARR is produced.

Algorithm 1 Tree Traversal within the same attribute Hierarchy.

```
1: $CDAG \leftarrow \{Subquadrant \subset Quadrant \subset Forest\}$;
2: $RoutingTable \leftarrow$ Routing table used by current application;
3: $SensorAttributes \leftarrow$ Attributes current sensor possesses;
4: $SensorClusters \leftarrow$ Set of clusters the current sensor belongs to;
5: $SensorClusterLeader \leftarrow$ Set of clusters the current sensor is leader of;
6: $\mathcal{N}(X, Y)$ = function that returns the number of consecutively matched attributes between $X$ and $Y$, starting
   from the first attribute in both $X$ and $Y$;
7: Received packet P;
8: $DestAttrList \leftarrow$ list of attribute name-value pairs of the destination in P;
9: Find $E \in RoutingTable \mid (\mathcal{N}(DestAttrList, E)$ is maximized) ;
10: if ($E$ = $DestAttrList$) then
11:   if ($E \in SensorClusters$) then
12:     Flood P in $E$; Return;
13:   else if (P.PrevHop $\notin$ {path between current sensor $\wedge$ $E$}) then
14:     Send P to $E$; Return;
15:
16: if ($\exists$ $L \in SensorClusterLeader \mid (L$ = P.NextHop)) then
17:   if (P.PrevHop is parent node in $CDAG$) $\vee$ (sensor is root leader) then
18:     if ($\exists$ children node $\mid$ known attributes of children node match $DestAttrList$) then
19:       Send P to children node in $CDAG$;
20:     else
21:       Drop packet P;
22:   else
23:     if ($\exists$ unmatched attribute at level $L$ or higher between the sensor and $DestAttrList$) then
24:       Send P to parent of $L$;
25:     else if (all attributes from root to level $L$ match between the sensor and $DestAttrList$ $\wedge$ $\exists$ child cluster with
        increased attribute match) then
26:       Send P to sibling clusters;
27:       Send P to child cluster;
28:     else
29:       Drop P;
30: else
31:   Send P to leader of P.NextHop;
```

Algorithm 2 Handling Packets With No Attribute Hierarchy.

```
RoutingTable ⟵ Routing table used by current application; SelfAttrList ⟵ current sensor's attribute list;
Received packet P, no attribute hierarchy specified;
AttrList ⟵ attribute list of P;
(RequiredAttr, PreferredAttr, ExploringAttr) ⟵ (Required, Preferred, Exploring) attributes from AttrList;
if ($\exists$ attribute r $\in$ SelfAttrList that matches attribute in RequiredAttr) then
    if (SelfAttrList matches all attributes in RequiredAttr) then
        if (matching attributes between SelfAttrList and RequiredAttr belong to the same CDAG) then
            if ($\exists$ attribute p $\in$ CDAG | (p matches attributes in PreferredAttr) $\wedge$ (p is at lower level in CDAG than any r)) then
                if ({set of matching attributes p} (named matchp) that are at levels lower than r form a line in CDAG) then
                    if (attribute plowest $\in$ matchp) $\wedge$ (plowest at the lowest level in CDAG) $\wedge$ (plowest $\notin$ SelfAttrList) then
                        if ($\exists$ path to cluster c with attribute matching {RequiredAttr $\cup$ matchp} $\in$ CDAG) then
                            Send P to c;
                        else
                            LC ⟵ lowest common ancestor node in CDAG between SelfAttrList and {RequiredAttr $\cup$ matchp}
                            Send P to cluster leader in LC;
                    else
                        Flood P in plowest;
                else
                    LN ⟵ leaf nodes of matchp that form the longest branches;
                    for all leaf nodes $L$ $\in$ LN do
                        if (sensor does not belong to any leaf node cluster $c \in L$) then
                            if ($\exists$ path to any leaf node cluster $c$) then
                                Send P to $c$;
                            else
                                LC ⟵ lowest common ancestor node in CDAG between SelfAttrList and {RequiredAttr $\cup$ matchp}
                                Send P to cluster leader in LC;
                        else
                            Flood P in $c$;
            else
                RPseqs ⟵ {sequences of attributes from RequiredAttr $\cup$ matching attributes from PreferredAttr | (all attributes from RequiredAttr are present) $\wedge$ (the resultant sequence form a "line"
                in CDAG) $\wedge$ (as many matching attributes from PreferredAttr as possible are included)}
                for all longest sequences RP $\in$ RPSeqs do
                    if (SelfAttrList matches all attributes in RP) then
                        Flood in cluster at lowest attribute level in RP;
                    else if ($\exists$ path to entry $E$ $\in$ RoutingTable | $E$ matches at least all attributes in RP) then
                        Send P to $E$;
                    else if (SelfAttrList matches top $T$ attributes in RP) then
                        if (sensor is cluster leader at level $C$ in any of the $T$ attributes) then
                            Send P to child cluster of $C$ in RP;
                        else
                            Send P to cluster leader of $T^{th}$ attributes;
        else
            for all CDAG with matching attribute do
                for all lowest attribute level node $L$ $\in$ { CDAG $\cap$ RequiredAttr $\cap$ SelfAttrList} do
                    if (sensor is not cluster leader in $L$) then
                        Send P to cluster leader in $L$
            Flood P with RequiredAttr
    else
        for all CDAG with matching attribute do
            for all lowest attribute level node $L$ $\in$ { CDAG $\cap$ RequiredAttr $\cap$ SelfAttrList } do
                if (sensor is not cluster leader in $L$) then
                    Send P to cluster leader in $L$
        Flood P with (SelfAttrList $\cap$ RequiredAttr)
else
    if (RequiredAttr have not been seen) then
        for all CDAG do
            if (sensor is not cluster leader at any level in CDAG) then
                Send P to lowest attribute level cluster leader;
            else if (sensor is not root in CDAG) then
                Send P to parent cluster of the highest level for which sensor is cluster leader;
            else
                Flood P in CDAG;
    else
        Drop P;
```

Algorithm 3 Mesh Traversal within the same attribute Hierarchy.

```
1: $\mathit{CDAG} \leftarrow \{Subquadrant \subset Quadrant \subset Forest\}$;
2: $\mathit{RoutingTable} \leftarrow$ Routing table used by current application;
3: $\mathit{SensorClusters} \leftarrow$ Set of clusters the current sensor belongs to;
4: $\mathit{SensorClusterLeader} \leftarrow$ Set of clusters the current sensor is leader of;
5: $\mathcal{N}(X,Y)$ = function that returns the number of consecutively matched attributes between $X$ and $Y$, starting
   from the first attribute in both $X$ and $Y$;
6: Received packet P;
7: **if** (P was received before) **then**
8:     Return;
9: $\mathit{DestAttrList} \leftarrow$ list of attribute name-value pairs of the destination in P;
10: Find $E \in \mathit{RoutingTable} \mid (\mathcal{N}(\mathit{DestAttrList}, E)$ is maximized) ;
11: **if** ($E$ = $\mathit{DestAttrList}$) **then**
12:     **if** ($E \in \mathit{SensorClusters}$) **then**
13:         Flood P in $E$; Return;
14:     **else if** (P.PrevHop $\notin$ {path between current sensor $\wedge$ $E$}) **then**
15:         Send P to $E$; Return;
16:
17: **if** ($\exists\ L \in \mathit{SensorClusterLeader} \mid (L$ = P.NextHop)) **then**
18:     **if** ($\exists$ children node | known attributes of children node match $\mathit{DestAttrList}$) **then**
19:         Send P to children node in $\mathit{CDAG}$;
20:     **else if** ($\exists$ adjacent cluster $C$ at same level of $L$ with matching attribute $\wedge$ no copy of P came from $C$) **then**
21:         Forward P to all such $C$;
22:     **else**
23:         Drop P;
24: **else**
25:     Send P to leader of P.NextHop;
```

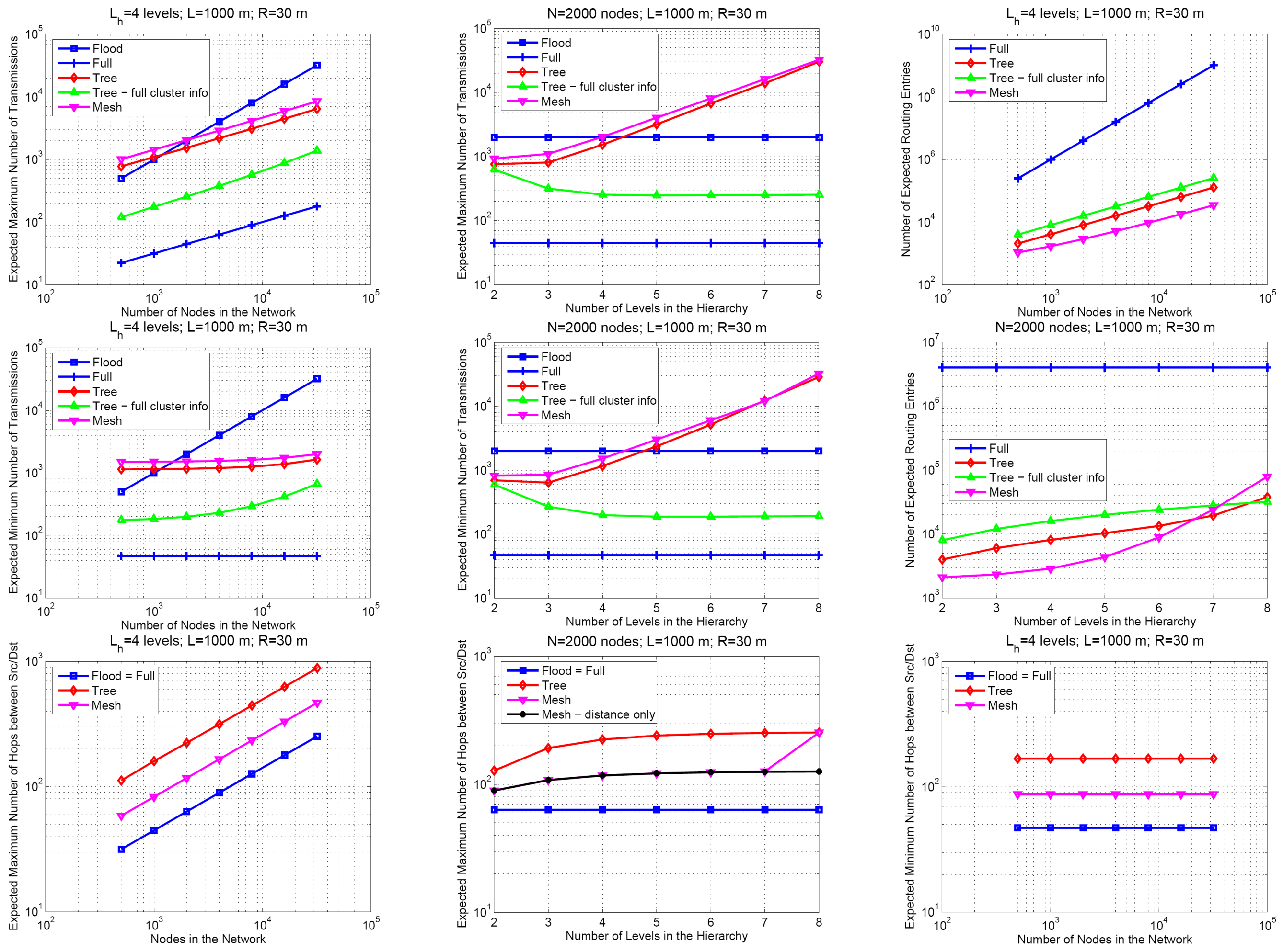

Figure 10: (a) *Top,Left:* Graph of expected maximum number of transmissions ($NumTxMax$) triggered by a packet with unknown destination (b) *Top,Center:* Graph of how $NumTxMax$ varies according to the number of levels in the hierarchy (c) *Top,Right:* Graph of Memory requirements with increasing number of nodes in the network (d) *Center,Left:* Graph of expected minimum number of transmissions ($NumTxMin$) triggered by a packet with unknown destination (e) *Center,Center:* Graph of how $NumTxMin$ varies according to the number of levels in the hierarchy (f) *Center,Right:* Graph of how memory requirements varies according to the number of levels in the hierarchy (g) *Bottom,Left:* Graph of expected maximum number of hops ($NumHopMax$) that separates source from destination as the number of nodes in the network increases (h) *Bottom,Center:* Graph of how $NumHopMax$ varies according to the number levels in the hierarchy (i) *Bottom,Right:* Graph of expected minimum number of hops ($NumHopMin$) that separates source from destination as the number of levels in the hierarchy changes