# ‘Ripples’: A Message-Efficient, Distributed Clustering Algorithm for Wireless Sensor and Actor Networks


**Neeta Trivedi**
*Scientist*
*ADE/ DRDO*
*Bangalore – 560093*
*India*

**G. Elangovan**
*Director*
*ADE/ DRDO*
*Bangalore – 560093*
*India*

*** S. Sitharama Iyengar**
*Roy Paul Daniels Professor of Computer Science and Chairman*
*Department of Computer Science*
*Louisiana State University*
*Baton Rouge LA 70803, USA*

**N. Balakrishnan**
*Associate Director*
*And*
*Professor*
*Indian Institute of Science*
*Bangalore – 560012*
*India*

* Please correspond to Prof. Iyengar for details (iyengar@bit.csc.lsu.edu)



***Abstract:*** *Reducing sizes and increasing capabilities of wireless communication devices as well as electronic and electro-mechanical systems have made it possible to not only continuously monitor a physical phenomenon by staying very close to or even inside it, but also to quickly and autonomously perform the required action upon sensing an event. Distributed, large-scale networks that exhibit this capability are Wireless Sensor and Actor Networks (WSAN). Like wireless sensor networks (WSN), WSAN consist of dense deployment of large number of small, low-cost and low-power sensor nodes and additionally contain few powerful and resource-rich actor nodes. WSAN need to address issues pertaining to energy conservation and unattended, untethered operation present in WSN and additionally face challenges due to heterogeneity of nodes and real-time requirements of the applications.*

*Lifetime of sensor nodes determines lifetime of the network and is crucial for the sensing capability. Among the important techniques proposed for prolonging the network lifetime by exploiting redundant deployment is using hierarchical architecture or clustering. However, the primary drawback of hierarchical control is the control message overhead; it is essential that the overhead does not dominate the network operations cost. The clustering algorithm should also scale to the network sizes and the nodes that are awake at any point in time must preserve the desired sensing coverage of the entire network utilizing the redundant deployment. Also, it has been shown that in sensor networks, a fundamental tradeoff exists between energy and latency for data delivery [1]. If clustering should lead to efficient MAC and Routing protocols it could effectively address the real-time requirements posed by WSAN.*

*The principle contributions of this paper are as follows. We propose ‘Ripples’, an extremely lightweight scalable clustering algorithm for clustering in WSN. Inspired by the cellular infrastructure model, ‘Ripples’ produces clusters of bounded geographic size (given a certain node distribution density this guarantees a bound on the number of nodes in a cluster as well), handles perturbations in the network locally in space and in time. The novelty of the algorithm lies in its deterministic operation to optimally exploit the redundant deployment and produce balanced clusters while retaining the desired sensing coverage with minimum possible control overhead.*

*Extensive simulation has been conducted and results are presented. We compare the performance of ‘Ripples’ against ‘Rapid’ and ‘Persistent’ [2]. These two algorithms achieve order of magnitude saving in terms of messages exchanged compare to the classical ‘Expanding Ring’ algorithm [3]. Of the two, ‘Rapid’ achieves better message efficiency at the cost of producing worst-case cluster sizes smaller than required. ‘Persistent’ produces clusters of desired size but is costlier. We demonstrate that ‘Ripples’ uses as less as 50% of messages compared to ‘Rapid’ to produce clusters of desired sizes, like ‘Persistent’.*

*The clustering approach used in ‘Ripples’ is similar to the one proposed in ‘GS$^3$’ [4]. We compare the control overheads of ‘Ripples’ with that of ‘GS$^3$’, and prove that the overheads for ‘Ripples’ are as less as 25% of those for ‘GS$^3$’.*

*A sub-module ‘Bind’ has been outlined that associated the clusters to their nearest actor nodes; this is required for real-time response. ‘Bind’ works along with ‘Ripples’ and creates data tables that can be used by the data dissemination algorithm, medium access and routing protocols for efficient operation.*



*Key Words: Ripples, Clustering Algorithm, Actor Networks, Sensor Networks and Data Dissemination Algorithm*


*January 8, 2006*

# 1 Introduction

Reducing sizes and increasing capabilities of wireless communication devices as well as electronic and electro-mechanical systems have made it possible to not only continuously monitor a physical phenomenon by staying very close to or even inside it, but also to quickly and autonomously perform the required action upon sensing an event. Distributed, large-scale networks that exhibit this capability are Wireless Sensor and Actor Networks (WSAN). Like wireless sensor networks (WSN), WSAN consist of dense deployment of large number of small, low-cost and low-power sensor nodes that perform the sensing task and additionally contain few powerful and resource-rich actor nodes that perform action on the environment. Such networks can be an integral part of battlefield surveillance, microclimate control in buildings, nuclear, biological and chemical detection, habitat monitoring, home automation and many other surveillance and control applications. While seemingly similar to WSN, WSAN need to deal with different issues arising out of heterogeneity of nodes, requirement to act on the environment in real-time, mobility of actor nodes and other application-specific requirements [5].

## *1.1 Challenges and Opportunities*

Sensor nodes are mostly deployed in ad hoc manner. They need to form the network infrastructure to include connectivity, addressing and routing by themselves and subsequently handle network perturbations. Sensor nodes are small, battery-operated devices that would rather be replaced than recharged, and the network needs to operate unattended and un-tethered for long time possibly in dynamic or hostile environment. The lifetime of the sensor nodes determines the lifetime of the network, and is critical for maximum field coverage; it directly affects the network's sensing capacity. Among the important techniques proposed for prolonging the network lifetime are exploiting the trade-offs among energy, accuracy and latency, and using hierarchical architectures or clustering. While the trade-off depends largely on the application requirements, establishing a hierarchical control through clustering can be employed in the most general form to all of them. Nodes are grouped into a set of clusters and a leader is elected in each cluster to represent cluster at a higher level.

Replacement of failed sensor nodes with new ones is expensive and often infeasible. In most cases a large number of redundant nodes are deployed. Clustering helps optimize the energy consumption in more than one ways. Clustering can be used to put the redundant nodes to sleep for most part of the time thereby preserve their energy. By grouping the nodes into clusters and letting the cluster head nodes coordinate the network operations the number of nodes contending for channel access is reduced. The information and updates are summarized at the cluster heads thereby suppressing redundant information. Routing through an overlay among cluster heads, which has a relatively small network diameter, also helps in energy saving; while transmission using many smaller hops is known to be more energy-efficient than that using few longer hops, this analysis ignores energy consumed by the RF components of the intermediate nodes [6].

However, the primary drawback of hierarchical networks is the control message overhead. Changes in topology due to node join, leave and failure require constant update of network infrastructure, and a high overhead will render ineffective the savings obtained by clustering. It is highly desirable that each of the algorithms and protocols employed minimizes control overhead, so that the network maintenance and management operations do not dominate the energy usage. Algorithms need to be lightweight, scalable and tolerant to node or link failures. Additionally, all the nodes that are awake at a given point in time must ensure the desired sensing coverage.

If an efficient clustering algorithm requires costly maintenance phase, its advantages may easily be offset. In sensor networks where network dynamism is high and unpredictable, periodic maintenance requires as much careful planning as the initial configuration itself.

In WSAN, taking timely action on the environment is crucial. This leads to two significant qualities that are required in a WSAN. Actor nodes will, in most of the applications, be mobile. It is required that the sensor nodes stay updated on the actor nodes' position. Second, the sensed information must be communicated to the actor nodes in the least possible time. This requires efficient routing mechanism with guaranteed delays.

## *1.2 Contributions of the paper*

We propose 'Ripples', a novel distributed algorithm for clustering the sensor nodes in WSAN with extremely low overheads. 'Ripples' aims at producing clusters of bounded geographic size (given certain node distribution density this guarantees a bound on the number of nodes in a cluster as well) and on preserving the desired sensing coverage. Perturbations are handled locally for achieving scalability. We do not make any unreasonable assumption on the capabilities of the sensor nodes or on the network.

We also outline 'Bind', an actor association mechanism that establishes links from the cluster heads to the closest actor node(s) and continuously updates this link as the actor nodes move. Associating sensor nodes to the nearest actor is required for achieving lowest possible latency, which is crucial for real-time response. While we do not propose a routing mechanism, the process significantly reduces the routing complexity since majority of the information flow in WSAN is from sensors to the actors.Detailed simulation has been carried out and simulation results are presented in the paper.
The clustering mechanisms can be broadly classified into two categories. In a bottom-up approach, certain nodes decide to become cluster heads based on some criteria and other nodes join these clusters. Most of the existing work falls under this category. In top-down approach, some node initiates the clustering and hands over charge to 'subordinates' to carry on with the process.
Many methods have been proposed in the literature for clustering in WSN [1-2, 4, 8-16]; we select three of these that focus on bounding the cluster sizes or on reducing the communication overheads for clustering for comparing 'Ripples' with. We show that 'Ripples' produces balanced clusters with significantly less communication overheads.
To contrast our method with bottom-up approaches we compare the message efficiency of 'Ripples' with the algorithms 'Rapid' and 'Persistent' presented by Krishnan and Starobinski [2]. These two methods show significant improvement over the classical expanding ring approach. 'Rapid' uses less message exchanges whereas 'Persistent' achieves the desired cluster size in most cases. We compare message efficiency of 'Ripples' with Rapid and cluster size with both Rapid and Persistent.
Nearly all of the bottom-up approaches involve additional overhead to cater to network dynamism. Either separate modules need to be invoked or global reclustering is needed to accommodate the newly joined or damaged nodes. 'Ripples' caters to network dynamism with no additional overheads.
Our work is similar to the top-down approach presented by Zhang and Arora [4]; however, 'Ripples' requires much less overheads and much simpler system model. We also present a theoretical analysis to substantiate our claim.

### *1.3 Outline of the Paper*

In section 2 we describe a WSAN scenario, derive a system model and define the problem formally. Related work is discussed in section 3. In section 4 we describe and analyze the algorithms and also compare 'Ripples' with 'Rapid', 'Persistent' and 'GS$^3$'. In section 5 we discuss how the post-deployment phase handles network dynamism. 'Ripples' can be adapted to a pure WSN scenario. We discuss this and some more possible variations in Section 6. Finally we conclude in section 7.

## 2 Scenario, System Model and Problem Statement

In this section we first consider an ideal application for wireless sensor and actor networks. Then we derive a system model for this scenario, define the problem formally and define some terminology.

### *2.1 Scenario*

The army faces great challenges in maintaining the security of civilians and military personnel due to the threat of biological, chemical and radiological attacks. Situational awareness and reliable communications are critical to achieving today's mission of detecting and combating such attacks. Wireless Sensor Networks provide a vital key to improving situational awareness. Different types of sensors could be deployed in an ad hoc manner inside own territory and could also be sent across the border, say, in an artillery shell. Being low-cost and untethered, the sensor nodes can be utilized for coverage of very broad area. The nodes can communicate the events of interest via secure and reliable radio frequency waveforms for operation in hostile environments bypassing blocked paths, if any. Consider the deployment scenario as in Fig 1.

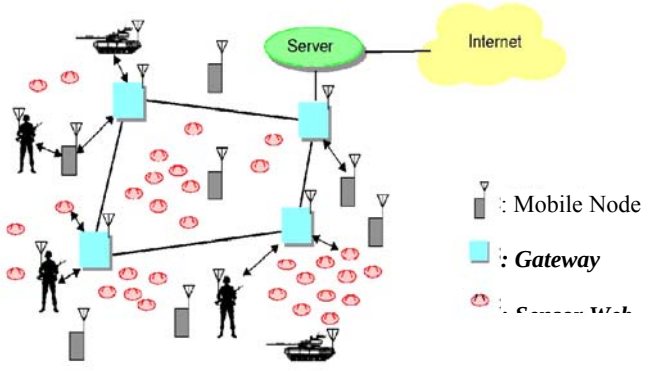



**Fig 1: Distributed Battleside/ Sensor Network**

Sensor nodes are deployed all over the field. In battle scenario, army sets up forward bases or quasi-permanent

communication stations. In peacetime there are command stations at key locations. These stations act as gateways to the operational headquarter, which could in turn be connected to the military headquarter, possibly on the Internet. Soldiers carry communication equipment on them, so do the vehicles. These stations, soldiers and military vehicles are the actor nodes. Sensor nodes upon detecting an event of interest pass on this information to one or more actor nodes, which perform the required actions.

## *2.2 System Model*

The scenario described above requires/ enforces the following system properties:

**Sensor Nodes:** Sensor nodes are stationary, deployed in large numbers in the battlefield. Failure rate of nodes in active battlefield is likely to be high and replenishment not always easy, requiring deployment denser than usual[1]. The entire battlefield needs to be covered by the sensor nodes and hence the deployment is nearly uniform. All nodes have equal significance; death of sensor nodes in a certain area could lead to loss of sensing capacity of the network in that area. This motivates the need for extending the lifetime of every node.

**Actor Nodes:** Some actor nodes such as the forward base are stationary while others like soldiers, tanks and other vehicles are mobile with varying speeds. These nodes can communicate and coordinate with each other at long distances, and will typically possess GPS-enabled devices. Depending on the action at the moment, stretches of a few square kilometers in the field could have just a couple of actor nodes (vehicles or observation camps) whereas in other areas several troops could be present at the same time. We make no assumption on the density or uniformity of deployment of the actor nodes.

**Localization:** While it is very critical to know the location where an event took place in the battlefield, use of GPS at each sensor node is impractical due to the size, power and cost factors. However, many lightweight location discovery mechanisms for WSN have been proposed in the literature [Ref]. In the scenario under consideration, actor nodes can possess GPS-enabled devices whereas sensor nodes can calculate their location – absolute or relative to some reference point – with the help of these actor nodes and through a simple location discovery mechanism.

**Wireless Transmission:** The need for precise alignment for line of sight in optical or infrared communication devices leaves their use impractical in the scenario described above; RF is preferred.

The size and power requirements of sensor nodes coupled with low frequencies of operation and mass scale deployment do not allow use of sophisticated antennae system at sensor nodes and most present-day sensor nodes use a very simple antenna model. We assume that the sensor nodes are equipped with simple and less powerful omni-directional antennae. However, a typical sensor node has a fixed number of transmission power levels[2] and we make use of this feature in 'Ripples'.

The actor nodes can use highly sophisticated technology including directional antennae with longer transmission ranges; however 'Ripples' does not need this.

**Perturbations:** Network dynamism (node failure and join, temporary or permanent) and node mobility are considered as perturbations. Both the sensor and actor nodes in the model are prone to damages, and more nodes can be added when required and when possible. Actor nodes will be on the move more often than not; static actor nodes are only special case of mobile ones.

**Characteristics of Self-Organization Algorithm:** The algorithm should be completely distributed and localized because a centralized solution is not scalable for WSAN. It should be simple, without excessive overheads because sensor nodes typically have limited energy sources. It should allow multi-hop routing to actor nodes, should allow most nodes to sleep most of the times, preferably in a deterministic manner. In any WSAN in general and for the scenario in particular, time to act is a very important parameter. It is highly desirable that target localization and message routing be deterministic and performed in the fastest possible manner.

## *2.3 Problem Statement*

In order to effectively and efficiently manage the scarce resources like the frequency spectrum, the bandwidth and power, the redundant deployment of sensor nodes is exploited and nodes are organized into clusters. The characteristics of the resultant

---

[1] Required node density is a function of the required sensing coverage, perceived failure rates and required network lifetime. See [7] for analysis on the critical density threshold for WSN. In our simulations, we use densities of 0.01 to 0.1 nodes/$m^2$.

[2] e.g. Berkeley motes support this feature. It's usually straightforward to set the transmission power level via standard system calls [6].

clusters as well as the algorithm used in forming them directly affects the energy efficiency of the network as well as the complexity of the medium access and routing protocols that can be employed.
We consider the sensor network as the graph GS = (VS, ES), where VS is the set of vertices or nodes and |VS| = n. ES is set of edges; an edge (u, v) $\in$ ES exists iff u and v can communicate with each other.
Each node $v_i$ has a residual energy level of $el_i$.
The actor network similarly forms another graph GA = (VA, EA).
The goal is to partition GS into subgraphs $GS_i = (VS_i, ES_i)$ where each subgraph represents a cluster that holds certain properties. Assuming that the abovementioned system model holds, it is required that:

1. $\forall\ v_j \in VS_i$, $v_j \notin VS_k\ \forall\ k, k \neq i$, i.e. each node belongs to one and only one cluster; $\forall\ GS_i$, $\exists$ unique $v_h$ called the cluster head[3].
2. The cluster heads form a connected network i.e. the graph represented by the cluster heads GCh = (VCh, ECh) is such that VCh $\subset$ V, ECh $\subset$ E, and GCh does not represent a disjoint graph. Communication takes place only through the cluster heads; this requires that the maximum distance between two neighboring cluster heads does not exceed the range that can be reached by a sensor node using its maximum transmission power.
3. A node is able to directly communicate with its cluster head (single hop). Since every node can become the cluster head at some point in time, this requirement means that $\exists$ edge (u, v) $\in ES_i$ between each u and v $\in VS_i\ \forall\ GS_i$. This way both the intra- and inter-cluster communications do not require non-clusterhead nodes to participate in routing or in route update procedures. Coupled with the requirement in serial 2 above, this places a bound on the maximum distance between nodes within a cluster, limiting the transmission energy required for intra-cluster communication.
4. In the event of some nodes failing or edges getting perturbed due to obstacles or noise, a new set of subgraphs $GS_i$ is obtained automatically that meets the abovementioned requirements. Additionally, if the disturbance is around a particular subgraph $GS_n$, only $GS_n$ and its neighboring subgraphs need to be altered. Also, the subsequent operations on the graph are not affected by this reorganization. In other words, the network is self-healing in the event of perturbations, and the self-healing operations are local in time and space in order to achieve stability and scalability.
5. In each $GS_i$, nodes take turn to become cluster head. Election of a cluster head $v_k = CH_i$ is such that $el_k \geq el_i\ \forall$ i that is $v_k$ has the maximum residual energy among all members. This balances load among nodes and prolongs the network lifetime while preserving the desired sensing coverage.
6. The clustering process requires minimal overheads and thus is efficient in terms of message complexity as well as processing complexity.
7. There is a mapping from each cluster $GS_i$ to one or more $va_i \in VA$ i.e. the cluster is associated with one or more actors. Distance between a cluster and an actor is calculated by number of intermediate clusters (number of hops through clusters). The associated actors are among the ones closest to the cluster[4] and the cluster has a way to reach these actor(s) through intermediate clusters[5].
8. The clustering must preserve the desired sensing coverage when only the cluster heads are awake.
9. Cluster characteristics must enable efficient design of network protocols.

## 3 Related Work

The field of sensor networks has attracted many researchers and the issue of clustering has been addressed in many different ways, to address different system requirements. Some of the clustering methods for WSN have been examined by Ahmed, Shi and Shang [8] and by Akyildiz et al. [9].

---

[3] All nodes in a cluster take up the role of cluster head by rotation for energy balance, but at a given time there is only one cluster head.

[4] The distance can change, say, due to perturbations in the network when messages may have to take alternate, or longer, routes. Since these perturbations can't be predicted, we select the actor on the shortest available path.

[5] This is ensured by requirement 2.

Heinzelman, Chandrakasan and Balakrishnan [10] have proposed a clustering method called Low-Energy Adaptive Clustering Hierarchy (LEACH). LEACH makes two assumptions: i) there exists a unique and stationary Base Station (BS) with which all the sensors want to communicate. ii) all the sensor nodes are able to communicate directly with the BS. Each node elects itself to be a cluster head at the beginning of the setup phase starting at time 't' with probability $P_i(t)$. $P_i(t)$ is so chosen that the expected number of cluster heads for this round is 'k'. The parameter 'k' is a function of 'N', the number of nodes distributed through an MxM region of space. Nodes broadcast 'Hello' messages to reach certain number of hops; the number of Hello messages received is an estimate of the number of nodes. Cluster heads communicate with the BS directly whereas cluster members communicate to the BS through cluster heads. Load-balancing is achieved by allowing different nodes to become cluster heads at different times. Related to LEACH is PEGASIS [11], in which cluster heads communicate with the BS using multi-hop transmission to save energy.

The LEACH framework does not suit the scenario under consideration. There is no single Base Station with which the sensor nodes want to communicate and the actor nodes are assumed to be constantly on the move. Second, in LEACH, cluster membership of nodes can change in every operational frame. While this ensures lowest transmission power between cluster members and cluster head, it requires global reclustering at each iteration resulting in high overheads. Change in clustering requires constant updates of routes as well. Third, LEACH puts a limit on the number of cluster heads, and it is assumed that all other nodes are within transmission range of at least one of these nodes. This assumption may not be true given the spread of deployment. As such, LEACH either requires nodes to have global knowledge of system parameter or requires the base station to interact with every cluster head.

Liu and Lin [12] present a clustering algorithm. They also stress on rotation of cluster-heads for even load-balancing in a homogeneous sensor network. However, the authors assume a model is where each node can vary its transmission range to communicate directly with any other node in the network. This is only feasible for very small network sizes.

Bandyopadhyay and Coyle [13] discuss an algorithm for clustering in sensor networks. Nodes become cluster heads with certain probability and advertise themselves. This advertisement is forwarded to all nodes that are k hops away from this cluster head. Nodes that do not hear an advertisement for some time period *t* (enough for the advertisement to travel k hops) become forced cluster heads. Like many other methods, this method focuses on producing clusters with bounded logical size or maximum number of hops for intra-cluster communication.

HEED is proposed by Ossama Y and Sonia F [6]. The authors emphasize on terminating the clustering process within constant number of iterations and minimizing the control overhead. Additionally, they focus on producing well-distributed cluster heads and prolonging network lifetime by selecting cluster heads based on residual energy levels. A node calculates the probability of itself becoming the cluster head subject to certain minimum value; the probability is based on node's residual energy and number of nodes that can simultaneously be allowed to initiate clustering. Each node doubles its probability of becoming the cluster head after every iteration; once this probability becomes 1 it declares itself as a cluster head. HEED also repeats the entire clustering process periodically.

Like HEED, we propose using lower energy levels for intra-cluster communication and higher levels for inter-cluster communication. Like HEED, we insist on load balancing. However, unlike HEED, we execute reclustering only to cater to node failures and node joins, and reclustering is completely localized. During periodic rotation of cluster heads, new cluster head is chosen from among the existing cluster members; cluster membership is not disturbed. Also, unlike HEED, our method focuses on geographical spread of the clusters as well as more uniform cluster distribution.

Cerpa A and Estrin D present ASCENT [14]. Initially few nodes are in the active state and the others in passive state. Active nodes start transmitting. Another active node that hears the transmission but also senses some packet loss issues a 'help' message. A passive node that hears help message may decide to become active based on certain probability and acts as a router. The process ends once the network is stabilized. Given a dense network, ASCENT extends its lifetime by letting few nodes die before the passive nodes become active.

An access-based clustering protocol is presented by Hou and Tsai [15]. The protocol dedicates one channel for out-of-band signaling, which in most cases requires a separate radio, increasing the cost and complexity of the nodes.

While the abovementioned pieces of work deal with homogeneous set of nodes, in some other cases authors have assumed presence of two types of nodes: pure sensor nodes and nodes with additional features and/ or capabilities. These 'special' nodes have been referred to in the literature as 'initiators', 'big nodes', etc. We now discuss some of these.

Subramaniam and Katz [16] emphasize on building clusters with some desired number of nodes in each. The nodes first perform neighbor discovery where an 'initiator' node sends a solicitation message within radius 'r' and the nodes that hear the message respond to it. The initiator counts the number of replies and if it is less than the desired number, it broadcasts the message over radius 'kr' for k>1. Once neighbor discovery is done, the algorithm tries to build groups of 8 members each. A node must belong to exactly one group and maintains the next-hop destination for every other member in the group. This is followed by building a hierarchy of groups. The nodes effectively form a tree structure which is also used for broadcast in the network and for building and maintaining routing tables.

This method gives way to an important concept. However, the message complexity of this method is very high. There is an inherent assumption that routes and broadcast trees once built are not subject to significant changes, which makes this method unsuitable for the scenario under consideration.

Another work closely related to [16] is Expanding Ring Algorithm by Ramamoorthy, Bhide and Srivastava [3]. The initiators in this method increment hop count at each iteration and send to their immediate neighbors. The neighbors check the hop count and pass on the message to their neighbors if required. The process is repeated until the desired number of nodes is available in the cluster. Additional nodes obtained in the last layer are dropped and this information is conveyed to them through intermediate nodes. As discussed by Krishnan and Starobinski [2], this process is also highly expensive in terms of number of messages exchanged.

Krishnan and Starobinski [2] present improvement over the Expanding Ring algorithm. The organization process is initiated by 'initiator' nodes. The algorithms try to build clusters of given size by allocating a 'growth budget' to the neighbors. Of this budget, the neighbor accounts for itself and passes on the rest to its neighbors. A 'subtree' is thus built at each node and once the budget is exhausted or no further growth is possible, the subtree is sent to the parent. In the 'Persistent' algorithm, the initiator tries to reallocate the remaining budget, if any, to the neighbors who still seem to have scope for growth.

Rapid and Persistent achieve significant reduction in terms of number of messages exchanged compared to Expanding Ring. As we show later, 'Ripples' uses significantly less number of messages compared to 'Rapid' while giving the average cluster size same as 'Persistent' does.

Another energy-efficient Self-Organization approach has been proposed by Chakrabarti and Iyengar [17]. The algorithm, SCARE, presented by the authors focuses on preserving sensing coverage along with energy-efficiency. Preservation of sensing coverage is a very important aspect of sensor networks and our method also addresses this issue.

SCARE considers a Sensing Range (s) and a Transmission Range (r) for the sensor nodes; $r > s$. It lets some nodes become 'Coordinator' based on certain probability. The coordinator node issues a HELLO message. A non-coordinator node that receives this message and is within range 's' joins the coordinator; a non-coordinator that receives the message but is outside the sensing range is eligible to become a coordinator. Random wait is given in order to prevent many nodes becoming coordinators at once. In order to prevent network partitioning, nodes also have a timeout mechanism after which they elect themselves as coordinators.

Some more work related to preserving of sensing coverage has been proposed in the literature [18-19]; however, we select SCARE for comparison because the message-complexity of SCARE is the same as that of 'Ripples' whereas other methods are more complex.

Up to here we discussed the 'bottom-up' approaches. A common problem with these approaches is that the clusters are not well-distributed, the number of neighboring clusters in each case is not deterministic, and there could be large overlap between clusters. In almost all of the approaches presented except HEED and SCARE, intra-cluster communication is multi-hop. This requires the non-cluster head nodes also to participate in routing and therefore makes the design of MAC and Network layer protocols interdependent. In addition, in all of the methods proposed above, the post-deployment phase either requires global reclustering or involves different algorithms, increasing cost and complexity.

'$GS^3$', presented by Zhang and Arora [4], applies top-down approach to clustering. '$GS^3$' is a self-configuring, self-healing, scalable, geography-aware clustering mechanism that divides the given network into non-overlapping hexagonal 'cells' of roughly equal size. One or more 'big' nodes act as initiators and are rooted at the cluster-heads hierarchy. The center of each cluster is the ideal location for its cluster head, and the node closest to this location is elected as the head.

Melodia T, Pompili D, Gungor CV and Akyildiz IF discuss a Distributed Coordination Framework for WSAN [20]. The authors discuss an event-driven clustering scenario and propose sensor-actor as well as actor-actor coordination mechanism. Upon occurrence of an event, the sensor nodes send exploratory messages to the actor nodes. A packet is considered expired if it is received after a delay and tolerance period. Actor computes the ratio of expired packets to the reliable packets and periodically broadcasts its value. If this value is less than some threshold there is a need to reduce the number of hops (perhaps by increasing the Tx range and sending to a far off node), if it is above an upper threshold the reliability can be traded off for energy saving, the node then sends data to a node close by. The path thus established is a tradeoff between energy and latency. Our work differs from this work in many ways. One, the authors consider the classes of applications where actor nodes are not mobile. In our scenario actors will almost always be mobile. Second, nodes that belong to a cluster in this work are those who need to send data to one actor in order to meet the real-time requirement of the application whereas 'Ripples' forms clusters so that redundant nodes in an area can be put off to sleep. Since actor nodes in our scenario are most likely mobile, they send periodic control messages (periodicity can be a function of rate of mobility) that helps set up the path for routing without the sensors sending any exploratory data. If the rate of event occurrences varies, the rate of cluster maintenance can be reduced without involving any additional overheads thus saving energy. However, the authors present an actor-actor coordination mechanism that can be used in conjunction with 'Ripples' for actors to take effective action.

More related work is discussed in [17] and [21].

### 3.1 The Missing Link

While most of the existing methods discussed in the previous section highlight important aspects of the self-organization problem in sensor networks, none addresses the unique requirements of WSAN especially in the scenario described earlier, as we shall see below. As such, the clustering methods have not been designed to enable the design of efficient networking protocols.

'$GS^3$' addresses some of the issues but assumes the Sink or the big node to be part of the network; movement of the big node requires changes to existing clustering, involving additional overheads. The application demands that the 'Sensor Webs' (see fig 1) be in close touch with the actor nodes yet be independent of them. Changes in actor nodes' topology should not require sensor nodes to recluster.

'Ripples' resembles '$GS^3$'; however, are significant differences. While 'Ripples' is motivated by hexagonal cell structure, it does not attempt to build rigid hexagons. This provides increased flexibility and message efficiency. Also, the information 'sinks' i.e. actor nodes are not part of the sensor network. Actors never act as cluster heads, separating their movement from cluster maintenance process.

'$GS^3$' assumes availability of directional antenna at each node, which is not feasible for most applications in the near future given the size, power and cost constraints. Also, in $GS^3$, cluster heads once elected do not change until they fail or move; energy balance is achieved by letting these nodes die before some other nodes take on as cluster heads. When all the nodes close to the ideal center of the cell exhaust their energy, entire cell structure is shifted. This requires global reclustering and also may affect the sensing coverage. Cluster head rotation does not consume substantial amount of additional energy since in any case the periodic cluster maintenance needs to be to cater to perturbations in the network.

The authors have not discussed the message efficiency of '$GS^3$'. We present a theoretical analysis of the message efficiency of 'Ripples' and '$GS^3$' to show that 'Ripples' performs better.

The clustering should make full use of redundant deployment yet preserve the desired sensing coverage. No method Other than 'SCARE' [Ref] emphasizes on preserving sensing coverage.

'Ripples' preserves the desired sensing coverage. Additionally, it produces clusters that can form the basis for the design of efficient networking protocols.

## 4 'Ripples' and 'Bind'

We now describe and analyze the 'Ripples' algorithm. First we define the key terms used in the descriptions. Next, we present a functional overview of the system. We then describe the 'Ripples' algorithms in detail and discuss its complexity. Finally we present 'Bind' and discuss its correctness.

### 4.1 Definitions

We define the following terms that are used subsequently in this paper.

**Node Density:** Density of the sensor nodes in the network ($\lambda$), in number of nodes per meter2.

**Cluster Layer:** 'Ripples' starts from the actor node and we call the innermost cluster as layer 0. It then grows outwards in circular 'ripples' and each ripple is called a 'Cluster Layer'.

**Cluster Diameter:** Clusters are approximated as circles and cluster diameter (DC) refers to the diameter of that circle. Typically the distance that can be reached by sensor nodes using minimum power level.

**Cluster Hops:** Number of intermediate clusters on the path to a given destination. 'Cluster hops' ($H_C$) is the number of clusters (cluster heads, essentially) that participate in forwarding the message, and determines the time taken to reach the destination.

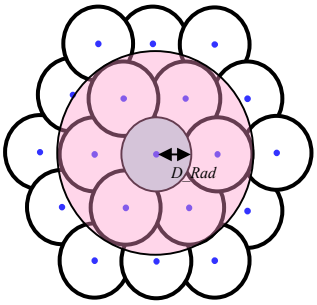
#### 4.1.1 Timer Values

'Ripples' uses the following timer values.

1. **Network Operation Cycle ($T_{NO}$):** One frame duration or duration between two successive beacons.

2. **Reclustering Frequency (FR):** Measured in terms of number of TNO, it is the frequency at which cluster heads announce themselves. No global reclustering is performed; cluster members use this parameter to decide if the cluster head is operational; a newly joined node uses this to join a cluster.

3. **Cluster Head Timer (TCH):** Measured in terms of number of TNO, it determines how long the cluster members should wait to hear from the cluster head before concluding that the cluster head has died and electing another cluster head.
4. **Cluster Head Rotation Frequency (FCHR):** Measured in terms of number of TNO, it is the frequency at which a new cluster head with maximum residual energy is elected.
5. **Actor Association Frequency (FAA):** Measured in terms of number of TNO, it is the frequency at which actor nodes announce themselves. The announcement is used by the clusters to update the association information.
6. **Actor Association Timer (TAA):** Measured in terms of number of TNO, it determines how long a cluster head should wait before concluding that the actor node is no longer available.




***D_Rad: Desired Cluster Radius***

*Fig 3: Typical clusters produced by 'Ripples'*

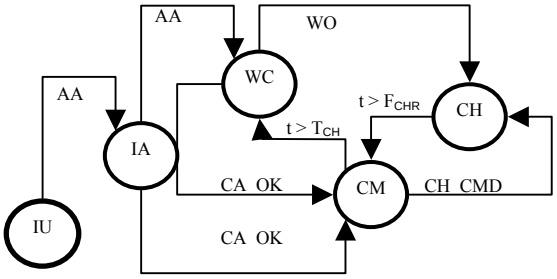
FAA is chosen to be greater than FR for reasons that are discussed later.
These frequencies and timer values will depend on the real-time nature, dynamism and criticality level of the application. For example, if the actor nodes move fast and events also take place more frequently, actors should announce themselves more often so that sensing information could be forwarded to the desired destination in least possible time. If nodes are expected to fail more often or new nodes join more often, clustering should be more frequent.
These values can be different at different times for the same application. For example, during intense battle, actors may move fast, sensors may fail more often and events could be more frequent. Once the battle is paused, all of these could be less frequent. The abovementioned parameters can be preprogrammed into the sensor nodes. Subsequently, their values can be altered by the actor nodes.
The cluster head and actor announcements are essentially very short broadcast packets. There is no periodic global reclustering. This makes the algorithm extremely lightweight with absolute minimum overheads.

### *4.2 Functional Overview of 'Ripples'*

We provide a functional overview of 'Ripples' through a state-transition diagram (Figure-2). A node can be in one of the following states: Initial Unaware (IU), Initial Aware (IA), Waiting To Cluster (WC), Cluster Head (CH), Cluster Member (CM). Each node maintains certain application specific timers. These timers are activated upon sensing some events and their activation as well as expiry changes a node's state.

### *4.3 'Ripples' Operation*

Metaphorically speaking, the actor node throws a 'pebble in the lake' and clusters get formed in concentric circles growing outwards. The actor sends out an 'Actor Association Announcement'. Sensor nodes that hear this announcement and are within the desired range form cluster and elect a leader. This leader advances the announcement to the next bigger circle. Sensor nodes in this layer form suitable clusters based on their locations and then advance the announcement. The process continues until there are no more clusters formed. If there are multiple actor nodes, they all throw their own pebbles. The individual ripples proceed until they meet each other.




AAA: Actor Association Announcement
***CA_OK: Cluster Join Announcement from Desired Cluster Center***
***WO: Wait Over, CH_CMD: Cluster Head Command, t: Time Elapsed***

**Fig 2: State Transition Diagram for Ripples**

Salhieh A, Weinmann J, Kochhal M and Schwiebert L discuss Power-efficient topologies for WSN [22]. They make two important observations. One, an increase in the number of neighbors increases the total power consumption, because it means increase in number of receivers. This means there is a fundamental tradeoff between reducing number of transmissions and increasing number of receptions. Second, increasing number of nodes increases the degree of routing freedom.
This coupled with the known advantages of a cellular communication infrastructure inspires 'Ripples' to have six neighbors for each cluster. While we do not attempt to make tight hexagons, the structure is derived from the hexagonal structure. Fig 3 shows layout of typical clusters produced by 'Ripples.

A typical sensor node can select the transmit power level from one of the available six or eight discrete power levels. We group these power levels into three; we call these power levels as $Pt_1$, $Pt_2$ and $Pt_3$ with associated transmission radii as $R_1$, $R_2$ and $R_3$ respectively. We assume that $R_2 \geq 2^* R_1$ and $R_3 \geq 2^* R_2$.

Actor Node makes an 'Actor Association Announcement' (AAA). This is a short broadcast packet that contains: Actor_ID (actor node ID), Center (center location of the cluster, in this case location of the actor node), D_Rad (desired cluster radius[6]) and Depth (distance from actor node in terms of number of intermediate clusters, in this case 0)[7]. This announcement is made with power level set to $Pt_1$.

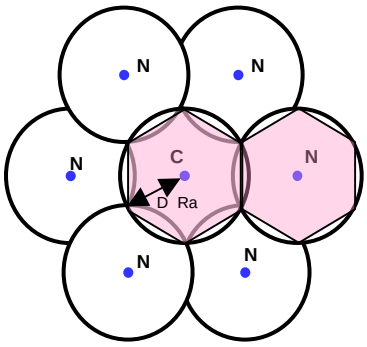
When AAA is received from the actor node (Depth=0), every node in IU state that listens to it (i.e. is within radius $R_1$[8]), computes its distance δ from 'Center', in this case the location of the actor node. If δ <= 'D_Rad', the node enters IA state; upon hearing the second AAA[9], node in the IA state switches to WC state and waits for a time period α*δ, where α is an application-dependent constant decided by the node density. It also sets its associated actor ID as the one received through the AAA and Depth as 1.

The node closest to 'Center' completes its wait first, enters CH state (ties broken by node ID) and issues a 'Cluster Join Announcement' (CJA) with power level set to $Pt_2$[10]. The nodes in WC state join this cluster, enter CM state and send acknowledgement to the cluster head. For announcements originating from sensor nodes (Depth > 0), the algorithm works as follows. The newly elected cluster head makes an AAA with power level set to $Pt_3$[11]. The announcement is the same short broadcast packet that includes 'Center' (center location of the cluster), 'Actor_ID' (actor node ID) and 'Depth'.

Sensor nodes within radius $R_3$ hear this announcement; of these, nodes that are at most 3*D_Rad distance away from the center of the originating cluster and are in IU state enter IA state; upon hearing the second announcement the nodes in IA state enter WC state. They set their Actor ID as the one received in the announcement and Depth to one greater than that received in the announcement.

Each of these waiting nodes computes the center locations of the six possible neighbors of the originating cluster (Fig 4) and computes its distance from all of these locations. Of these, the node selects the shortest distance δ (ties broken by random choice); the corresponding center location is the desired cluster center. The node waits for a time period α*δ. The node closest to the center completes its wait first and takes over as the cluster head and announces itself. Nodes that were in the WC state and had the same 'Center' as the announcing head join this cluster.

Note that only the neighbors that were not previously clustered will respond back. The new




N1.x = C.x + 2*D_Rad*COS 30, N1.y = C.y

N2.x = C.x + D_Rad*COS 30

N2.y = C.y + D_Rad + D_Rad*SIN 30

N3.x = C.x - D_Rad*COS 30, N3.y = N2.y

N4.x = C.x - 2*D_Rad*COS 30, N1.y = C.y

N5.x = N3.x

N5.y = C.y – D_Rad - D_Rad*SIN 30

N6.x = N2.x, N6.y = N5.y

**Fig 4: Desired Center Locations of Neighboring Clusters**

---

[6] D_Rad could be so chosen that the worst-case distance between two neighboring cluster heads is equal to sensing range. This ensures desired sensing coverage. Choosing this radius to be less than sensing range provides sensing redundancy.

[7] It sends another parameter called 'On Move', which is discussed in the next subsection on 'Bind'.

[8] It is possible to select this distance to be less than the sensor nodes communication range.

[9] Ignoring first announcement caters to post-deployment phase without any changes in the algorithm,.

[10] Cluster head may be away from the actor; if it transmits with the same power level, nodes within D_Rad from actor may not be covered. More importantly, typical sensing range of sensor nodes is half of their communication range. By selecting cluster radius as half of the communication range ensures proper sensing coverage even when only the cluster heads are awake.

[11] Given $R_3 \geq 2^*R_2$, this ensures the desired set of sensors is covered even if the selected cluster head is away from center of the cluster.

cluster heads then extend the ripple[12].
The pseudo code of the algorithm is presented in Appendix-A.

### *4.4 'Ripples' Analysis*

We now analyze the key features of 'Ripples'. We also discuss the algorithm against some of the requirements discussed in section 2.3 n this and the next subsections. Some other requirements dealing with network dynamism and actor association are discussed in later sections.

#### 4.4.1 Clusters Produced

**Lemma 1:**

*The number of clusters produced is at most (1+3*L*(L+1)) where L = (Network Diameter[13]/ Desired Cluster Diameter)*
**Proof:** Except the innermost layer where number of clusters is 1, each $i^{th}$ layer consists of at most (6*i) clusters. Each cluster has a diameter of 2*D_Rad or the Desired Cluster Diameter. The number of cluster layers is L = (Network Diameter/ Desired Cluster Diameter). Therefore, the total number of clusters is bounded by

$$C = 1 + \sum_{i=1}^{L} (6*i) = 1 + 6*L*(L+1)/2 = 1 + 3*L*(L+1)$$

**Observation 1:**

*Clustering by 'Ripples' allows simple protocols to be employed for MAC and routing (Req 9).*
The cluster size corresponds roughly with the direct communication range of the nodes. This allows simple protocols to be used for routing, medium access and broadcasting within a cluster. The clusters have fixed geographical size and regular layout. This facilitates the same time- or frequency- division multiplexing to be reused [23]. Since every cluster has a fixed number and layout of neighbors, this also simplifies routing. 'Ripples', coupled with 'Bind' (discussed later) effectively sets up gradients for use by the routing protocol.
**Observation 2:**

*Clustering by 'Ripples' extends the overall network lifetime.*
In the scenario under consideration in this paper, if the number of clusters is independent of the node density, the cluster radius can be chosen such that the cluster head graph provides the desired sensing coverage. Such a system would maximally utilize redundant deployment. Excess number of nodes can be deployed; however, only a constant number of nodes will be fully operational at any point in time, prolonging the overall network life time.
**Observation 3:**

*The number of messages exchanged in 'Ripples' is independent of the network topology.*
The number of clusters, C, produced depends only on the area of deployment and desired cluster radius. Given the total number of nodes N in the network, a 'Clique' representing a sparse network and a 'Ring' representing dense topology both require C broadcast messages from cluster heads for CJA, N-C messages from cluster members to their cluster head and C broadcast messages for AAA. The total number of messages exchanged therefore does not depend on the network topology.
**Observation 4:**

*'Ripples' is sensing coverage aware (Req 8).*

---

[12] To avoid collision, Ripple is extended in alternate cycles–1, 3, 5 announce in one cycle and 2, 4, 6 in the next one. Comparison with neighboring locations on inner layer can be avoided; however, this could lead a chain of unclustered nodes when a 'Cluster Gap' exists.

[13] We refer to the network diameter as the largest Euclidean distance between any two nodes.

Limiting the worst-case distance between two cluster head nodes to the sensing range of the nodes and keeping cluster heads awake ensures the desired sensing coverage at all times. By reducing this distance, desired sensing redundancy can be provided.

### 4.4.2 Correctness

**Lemma 2:**

*At the time ($4*\alpha*D\_Rad*Depth*T_{NO}$), each sensor node 'Depth' hop count away from the initiator actor node is either a cluster head or a cluster member that belongs to one and only one cluster (Req 1).*

**Proof:** We prove this also by induction. When Depth=0, the only node 'Depth' hop count away from the actor node is the actor node itself, no sensor node is in this category. The lemma holds good in this case.

At depth 'i', the node that listened to the AAA

a) could already have been clustered. The lemma holds good in this case.
b) was not yet clustered. It waits for time $\alpha*\delta$ (max of $\alpha*$D_Rad). After $\alpha*$D_Rad cycles, each node would have stopped waiting; the node first to complete its wait becomes cluster head. Other waiting nodes join this cluster. Therefore, each node at this depth would either have become a cluster head or have joined a cluster.
   Odd and even numbered cluster heads in this layer further the announcement in two separate cycles. Also, the nodes ignore the first AAA. Therefore, at the end of $4*\alpha*D\_Rad*Depth*T_{NO}$, every node at 'Depth' hop count from the initiating actor node would either have become a cluster head or joined a cluster.

**Corollary:**

*The time for the entire network to get clustered is $\theta$(Max_Separation) where Max_Separation is the maximum distance between a sensor node and an actor node.*

**Lemma 3:**

*The cluster heads form a connected network (Req 2).*

**Proof:** We assumed that $R_3 \geq 2*R_2$ and $R_2 \geq 2*R_1$. We also assumed that $R_1 \geq$ D_Rad, the desired cluster radius. From these equations we get $R_3 \geq 4*$D_Rad. The maximum separation between two cluster head nodes is 4*D_Rad. This ensures that even in the worst case two neighboring cluster heads can reach each other using the power level $Pt_3$.

**Lemma 4:**

*A node can directly communicate with its cluster head (single hop) (Req 3).*

**Proof:** We assumed that $R_2 \geq 2*R_1$ and $R_1 \geq$ D_Rad. The maximum separation between two members in the same cluster is 2*D_Rad, which can be reached using the power level $Pt_2$.

**Lemma 5:**

*The algorithm terminates in finite number of steps (no cyclic execution).*

**Proof:** The algorithm terminates when there are no further AAA made. This happens when no more cluster heads are selected. Nodes that are in CH or CM state do not attempt clustering again, only the nodes in IU or IA states do. This ensures that the ripples only grow outwards. Once all the nodes are in CH or CM states (at the time $(4*\alpha*R_1*Depth*T_{NO})$ as per Lemma1, the algorithm terminates.

### 4.4.3 Message Efficiency of 'Ripples'

We now analyze the message efficiency of 'Ripples' (Req 6). The number of messages is inclusive of the acknowledgements sent by the cluster members to the cluster heads.

**Lemma 6:**

*The message complexity of 'Ripples' for clustering the entire network is O(N); the total number of messages transmitted in 'Ripples' is bounded by (N + 3*(1+3*L*(L+1))) where L = (Network Diameter[14]/ Desired Cluster Diameter) and N is the total number of nodes in the network.*

[14] Network diameter is the largest distance between any two nodes.

**Proof:** In each cluster, one member (cluster head) sends an announcement for forming cluster. Only the nodes that have chosen the same cluster center respond. This ensures that each non-cluster-head node sends acknowledgement to one and only one cluster head.
The cluster formation process therefore involves transmission of N messages where N is the number of sensor nodes in the network.
Each cluster head sends out one broadcast message as AAA; total message transmissions in this case is (1 + 3*L*(L+1)) (see Lemma 1).
Each cluster head except the centermost performs a handshake with its predecessor and receives command to further the AAA. These messages tally to (2*(1+3*L*(L+1))).
Therefore, the total number of messages exchanged are (N + 3*(1+3*L*(L+1))).
Assuming N >> L, message complexity of 'Ripples' to cluster the entire network is O(N).

### 4.4.4 Comparing 'Ripples' with 'Rapid' and 'Persistent'

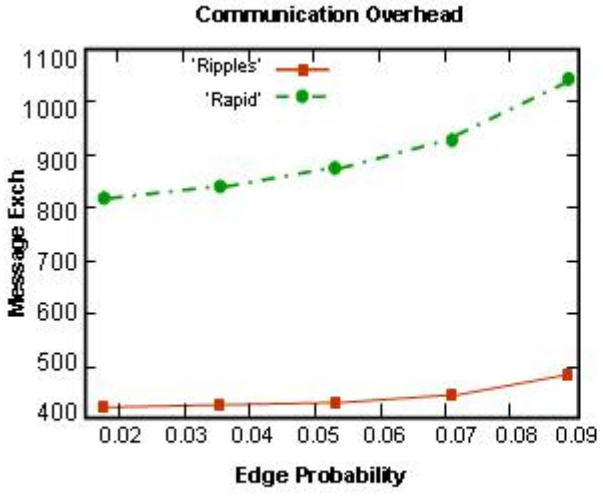
In this section we compare the message efficiency of 'Ripples' with that of 'Rapid' and average cluster size with 'Persistent'. Simulations were carried out using NS-2.27 [24]. CMU's 'setdest' utility (part of NS-2.27 package) was used for generating the random topologies. In the first set, we assume that two nodes have a link if and only if they are within distance $d_r$ units with each other (distance that can be reached with minimum power level); $d_r$ is chosen to be 1. This also fixes the cluster diameter for 'Ripples' as 1. Edge probability is approximated to be $\pi d_r^2/l^2$. In order to obtain a justifiable comparison, we first run 'Ripples' (with single initiator in the center, although this symmetry is not required) on the topology to generate clusters. We then choose the cluster heads selected by 'Ripples' to be the 'Initiators' for 'Rapid' and 'Persistent', keeping 'Budget' as the average cluster size produced by 'Ripples'. With this, the number of clusters produced by these three algorithms remains the same with cluster heads at the same locations. For 'Rapid' and 'Persistent', only one initiator is made active at a time, after forming the cluster it is made to hand over to the next initiator. We compare the number of messages exchanged only with 'Rapid', since 'Persistent' is known to use more messages. The average cluster size formed is compared both with 'Rapid' and 'Persistent'.
In the second case, we selected areas of 50mx50m, 50mx100m and 100mx100m and varied the node density from 0.01 to 0.1 per meter$^2$ in each case. Cluster radius for 'Ripples' is chosen to be 12m; the neighbor reachability for 'Rapid' and 'Persistent' is calculated accordingly. Simulations were run in a manner similar to the one described above.
In each of the two cases, 50 random topologies were generated for each scenario and simulation runs carried out on each. Results were averaged for comparison. Neighbor lists were generated from the scenario file prior to running simulations.

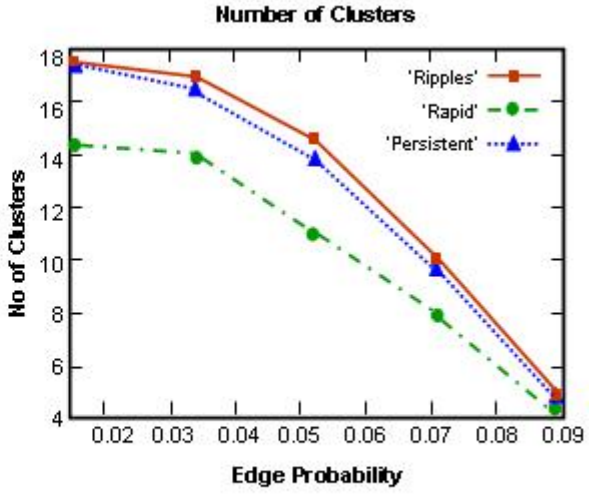
#### *4.4.4.1 Messages Exchanged*

The graphs in Fig 5A and 6A depict the number of messages exchanged in case of the two scenarios described above. 'Ripples' uses as less as 50% of the messages used by 'Rapid'. Note that 'Ripples' does not require additional messages for the post-deployment phase.

#### *4.4.4.2 Cluster Size*

The graph in Fig 5B shows a comparison between number of clusters produced by 'Ripples' as against those in 'Rapid' and 'Persistent' for increasing edge probability with same number of nodes. The graphs in Fig 6B show a comparison between



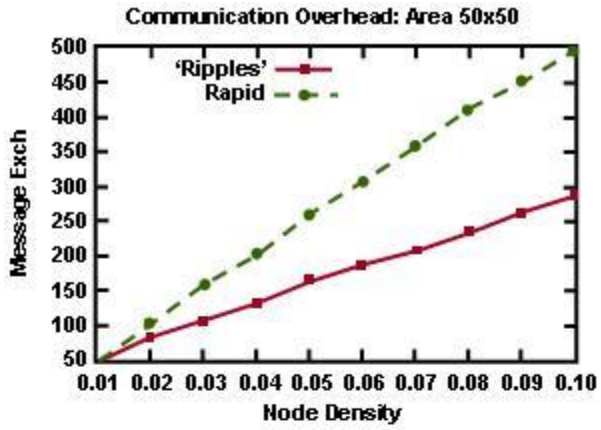



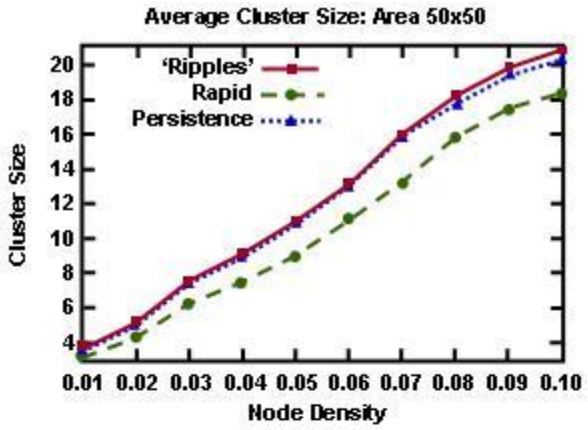

1/7/06

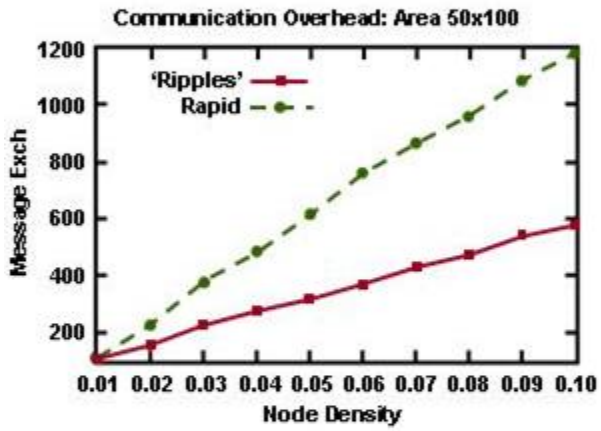
Fig 5A: Communication Overheads - 400 nodes  Fig 5B: Number of Clusters - 400 nodes

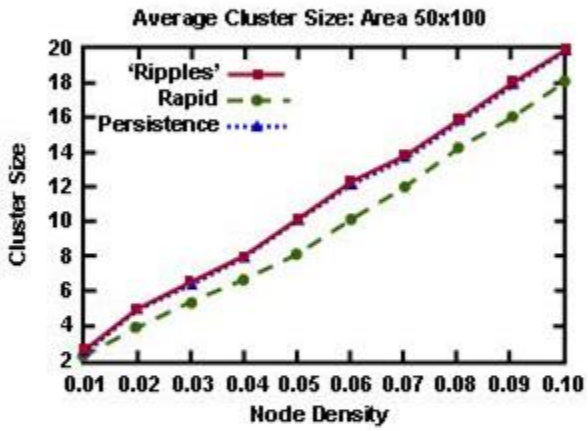
the average cluster size produced in terms of number of nodes in the cases of 'Rapid', 'Persistent' and 'Ripples' for the second scenario discussed above. We provide this comparison assuming that the selection of initiators and bound on cluster size in 'Rapid' and 'Persistent' and 'Desired Cluster Radius' for given density and spread of nodes in 'Ripples' can be matched to produce clusters of equivalent sizes. The comparison effectively shows the number of nodes covered as against the desired ones.

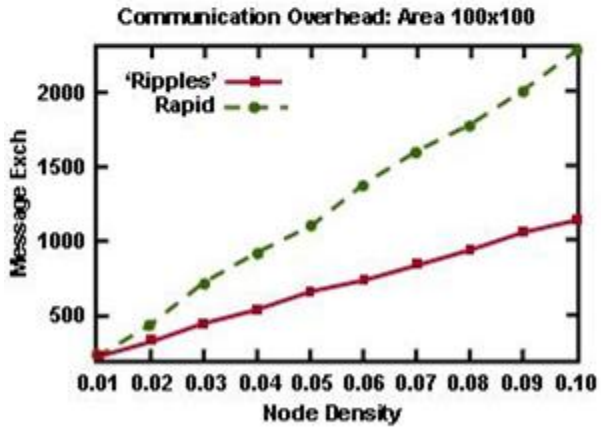
As the graphs show, 'Ripples' produces the clusters with required number of nodes, equal to or better that 'Persistent' and better than 'Rapid'. Since the communication cost of 'Ripples' is much less than that of 'Rapid', this shows that 'Ripples' produces clusters of required sizes with much less overhead, apart from providing the added advantage of preserving sensing coverage.



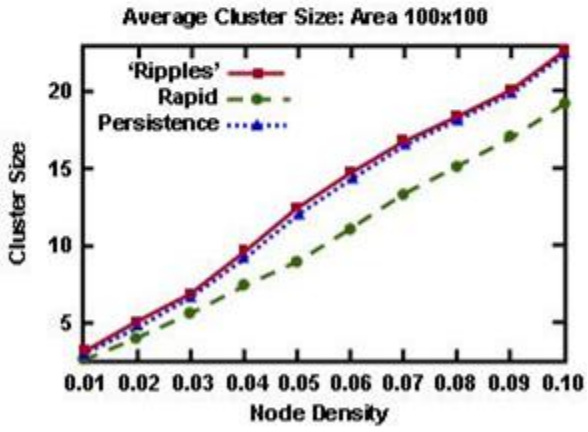




Fig 6A: Communication Overheads for Fixed Area With Varying Node Densities

Fig 6B: Average Cluster Size for Fixed Area With Varying Node Densities

### 4.4.5 Comparing 'Ripples' with 'GS$^3$'

In this section we discuss the main differences between 'Ripples' and 'GS$^3$'. To the best of our knowledge this is the only clustering algorithm proposed with top-down approach. 'Ripples' and GS$^3$ are similar in their approaches (both start from a head node or an Actor and grow outwards); however, both the approach to clustering and the assumptions made are different in these two cases.

- In the case of GS$^3$, the 'Big Node' acts as a cluster head. When it moves, another node replaces this head and if this node reaches an Ideal Location for another cluster head, it replaces the cluster head there. In 'Ripples', actor nodes are never part of the sensor network. All the cluster heads are ordinary sensor nodes. Movement of actor nodes does not alter the cluster head graph, and is completely separated from the clustering and reclustering process.
- Unlike GS$^3$, a cluster head in 'Ripples' only advances the clustering process further; it does not select the neighboring cluster heads. Cluster formation is completely independent. This results in significant saving in the number of messages exchanges while achieving the same goal.
- GS$^3$ assumes that the sensor nodes possess sophisticated, directional antenna system. Given the size, power and cost constraints this is not feasible in the near future. The sensor nodes in our model possess much simpler antenna systems.
- GS$^3$ focuses on retaining the cellular structure throughout the network lifetime with cluster heads very close to the ideal center of the cells. Since energy in cluster heads gets depleted faster compared to other nodes, this eventually results in no nodes being present close to the ideal location. In this case, a 'Cell Shift' is performed, which involves considerable overheads. Additionally, since nodes in the given areas are all dead, this leads to non-uniform sensing coverage over the network.

'Ripples' does preserve the desired cell center location as well as the coverage area of individual clusters, but cluster members take turn to become cluster heads to ensures uniform energy depletion in the network. Any new node within $R_1$ radius from the cluster center becomes members of that cluster.

We now present a comparison between the message efficiency of 'Ripples' and GS$^3$. We consider two scenarios. First, when the sensor nodes do not possess directional antennae so GS$^3$ needs to determine whether a node is within the search region only after it knows its location. This effectively means the search region for each head is (0°, 360°). We call this case GS$^3$-O. In the second case, we take the model proposed in GS$^3$ i.e. each node having a directional antenna. We call this case GS$^3$-D. As we show, even with this complex model, message complexity of GS$^3$ is higher than that of 'Ripples'.

#### *4.4.5.1 Case 1 – GS$^3$-O*

Since nodes do not possess directional antenna, each head needs to get location information of all the nodes within radius $\sqrt{3}\ R + 2\ R_t$. R here is equivalent to $R_1$ in our case. Number of nodes within the circular area, given node density $\lambda$, is
$\lambda * \pi\ (\sqrt{3}\ R + 2\ R_t)^2 \approx \lambda * \pi * 3\ R_1^2$. (Assuming $R_t << R_1$)

We assume that the command to seek the location information was broadcast and, therefore, the only transmissions counted are replies from the nodes.

Each head will be selected by exactly one 'Parent Head'. Also, each node will send acknowledgement to at least one cluster head to join the cluster. Assuming cluster radius of $R_1$, the number of nodes sending acknowledgement per cluster will be $\lambda * \pi * R_1^2$.

Therefore, total messages transmitted by a cluster head and the cluster members are, ignoring small numbers,
$\lambda * \pi * R_1^2 + \lambda * \pi * 3\ R_1^2 = \lambda * \pi * 4\ R_1^2$.

In the case of 'Ripples', this number is $\lambda * \pi * R_1^2$. Therefore, in the simplest case[15], number of messages transmitted in the case of GS$^3$-O is four times that of 'Ripples'.

#### *4.4.5.2 Case 2 – GS$^3$-D*

In this case we assume that the sensor nodes in GS$^3$ possess directional antenna, while those in 'Ripples' have simple omni-directional antennae.

---

[15] In GS$^3$, a new node may turn out to be a better head in which case this node will be nominated as the head. This will involve additional message exchanges. We also ignore here the acknowledgement sent by existing cluster heads.

In $GS^3$, each cluster head except the Big Node receives location information of all the nodes within radius $\sqrt{3}\ R + 2\ R_t$ but only in the direction $(-60°-\alpha, +60°+\alpha)$. This is $(1/3)^{rd}$ of the full circle. Number of nodes in this area are
$(\lambda * \pi * 3\ R_1^2)/3 = \lambda * \pi * R_1^2$.
In case 1, each head will be selected by exactly one 'Parent Head'. Also, each node will send acknowledgement to at least one cluster head to join the cluster. Assuming cluster radius of $R_1$, the number of nodes sending acknowledgement per cluster will be
$\lambda * \pi * R_1^2$.
Therefore, total messages transmitted by a cluster head and the cluster members are, ignoring small numbers, $\lambda * \pi * R_1^2 + \lambda * \pi * R_1^2 = \lambda * \pi * R_1^2$.
In the case of 'Ripples', this number is $\lambda*\pi*R_1^2$. Therefore, in the simplest case, number of messages transmitted in $GS^3$-D is twice that of 'Ripples', when the nodes in 'Ripples' are much simpler.
Given large number of nodes, this is significant saving in terms of messages exchanged.

## *4.5 Sensing Coverage*

In this section, we compare the sensing coverage of 'Ripples' with that of 'SCARE'. 'SCARE' achieves sensing coverage close to 100% when node density is reasonable, with very less control overhead and very less fraction of nodes selected to be 'Coordinators'. 'Ripples' performs comparably in terms of sensing coverage as well as the number of nodes selected as cluster heads. Number of control messages in both 'SCARE' and 'Ripples' is little over N where N is the total number of nodes in the network. However, 'Ripples' additionally enables the design of efficient networking protocols.
Figures 7(a) and 7(b) show the obtained sensing coverage and fraction of nodes selected as cluster heads respectively for 'Ripples' and 'SCARE'. 'Ripples' was run with cluster radius of 12, 14 and 16 meters, shown in figures as Ripples-12, Ripples-14 and Ripples-16. Sensing radius in all the cases is 12.5m.
It is evident from the figures that 'Ripples' achieves the same (or more) sensing coverage as SCARE for much less number of nodes selected as coordinators. The number of coordinators can be reduced further by compromising on the sensing coverage

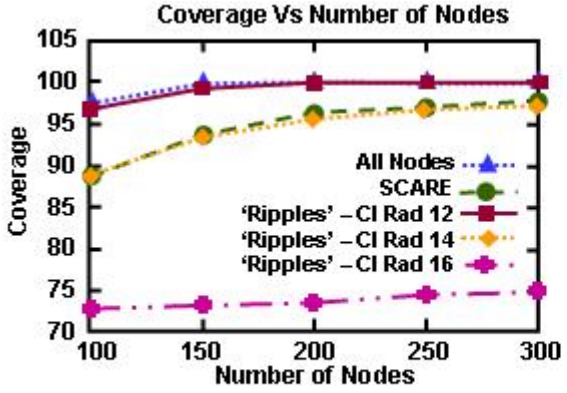



Fig 7(a) Sensing Coverage

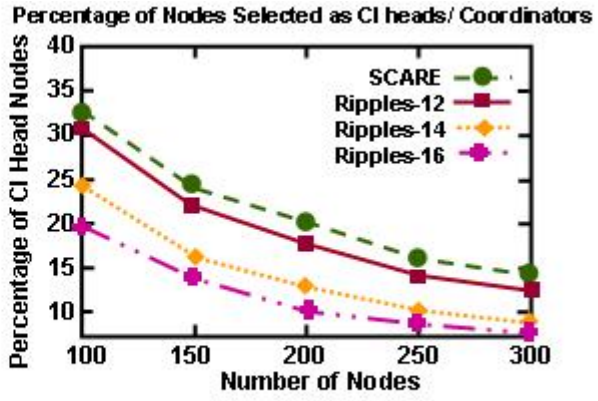



Fig 7(b) Percentage of Cluster Head Nodes

and increasing the cluster radius. This can alternatively be achieved by keeping the cluster radius small and letting cluster heads sleep periodically.

### 4.5.1 Summary of Comparison

The table given below summarizes the comparison between 'Ripples' (our method), 'Rapid', 'Persistent' and '$GS^3$'.

| | **Rapid** | **Persistent** | **$GS^3$** | **Ripples (our method)** | **SCARE** |
|---|---|---|---|---|---|
| Achieves Desired Cluster Size in terms of Number of Nodes | Not always | Mostly | Mostly if geographical size is chosen appropriately | Mostly if geographical size is chosen appropriately | Possible |
| Achieves Desired Geographical Cluster Size | No | No | Yes | Yes | No |
| Antenna Model | Omni | Omni | Directional | Omni | Omni |
| Routing | Non cluster head nodes need to participate in routing | Non cluster head nodes need to participate in routing | Multi-hop routing possible through cluster head overlay | Multi-hop routing possible through cluster head overlay | Multi-hop routing possible through cluster head overlay |
| Handling Perturbations | Post-deployment mechanism different from initial clustering | Post-deployment mechanism different from initial clustering | May call for global re-clustering after certain time period | No separate overheads for post-deployment phase | Post-deployment phase different from initial configuration |
| Preserves Sensing-coverage | Not likely if only cluster heads are awake | Not likely if only cluster heads are awake | Yes if cluster size is chosen appropriately | Yes if cluster size is chosen appropriately | Yes |
| Movement of Actor Nodes/ Sink | NA | NA | Calls for reclustering/ setting up proxies etc | Actor node movement completely separated from clustering process | NA |

## *4.6 'Bind' Operation*

'Bind' is a submodule of 'Ripples' that associates clusters to their closest actor node(s).
We assume that the actor nodes have a way of communicating among themselves and therefore it is sufficient to send the required data to the closest actor of a certain type. For example, armored vehicles may be interested in a particular type of event that may be of no use to foot soldiers. This information can therefore be sent to the closest available armored vehicle. However, multiple entries for the same type of actors can be maintained for the purpose of redundancy.

Every cluster head maintains association with one or more actor node types, as required by the application. It also maintains for each actor, the number of hops to the actor, the attribute of interest to the actor, the duration of interest and the rate of interest.
Each actor node announces itself with the frequency $F_{AA}$. On receiving this announcement:
- If it is a new actor of its type, add the entry to the table.
- If the actor is not of new type and its ID does not match the one in the association list but is closer than the stored one and its timestamp is newer, replace the entry by the new entry.
- If the actor ID matches one of the IDs stored in the list but the timestamp is newer, update the entry.

### *4.7 'Bind' Analysis*

In this section we discuss the properties of 'Bind' (Req 7).

**Lemma 5:**

*The sensor nodes are always associated with the nearest actor node of a certain type found thus far.*

**Proof:** We assume, for simplicity, that sensor nodes store association only with one actor node of one type. Case of association with multiple actors is similar.
We prove the lemma by induction. When Depth=0, the only node 'Depth' hop count away from the actor node is the actor node itself, no sensor node is in this category. The lemma holds good in this case.
At depth 'i', the sensor node that listened to AAA
a) could already have been associated with another actor of similar type. If the new actor is closer than the previous one, the sensor node updates its actor association, else leaves it unchanged. Therefore the sensor node is associated with the closest actor of this type found so far.
b) was not yet associated. It updates its actor association to be that of the only actor of this type found so far.

**Observation 5:**

*'Bind' assists in obtaining real-time response.*
Each cluster associates itself to the closest actor node of a given type. This results in the minimum possible latency in sending data to actors, assisting in getting real-time response. Also, since each cluster has a fixed number of neighbors, an upper bound on the number of hops to destination can be guaranteed. The $F_{AA}$ can be altered based on the real-time response requirements of the application and mobility of the actors.

**Observation 6:**

*'Bind' helps reduce routing complexity.*
The association through 'Bind' effectively forms gradients for information flow. Since majority of the traffic flow in WSAN is from the sensor to the actor nodes, the routing protocol can make use of these gradients to greatly reduce the routing complexity.

## 5 The Post-Deployment Phase

The primary requirements handled in this phase are network dynamism and load balancing. The distinction between the infrastructure establishment and maintenance is only conceptual. The nodes never need to know the network phase; they only need to know the state they themselves are in. We demonstrate how the node join and death are easily accommodated in 'Ripples' without the need for additional overheads. Like LEACH and HEED, 'Ripples' works in the 'Setup Phase' every few network cycles, and leaves the 'Steady State Phase' for other network operations. However, unlike these two, no global reclustering is performed.

### *5.1 Load Balancing*

Load balancing (Req 5) is achieved through periodic rotation of cluster heads. Handshake messages are exchanged as follows.

i) Cluster head selects the member with maximum residual energy and issues a 'Cluster head takeover request' (CH_CMD) to it.
ii) The member acknowledges the request with 'Cluster head accept acknowledge' (CH_ACK) message; updates own state to CH. If the designated cluster member fails to respond, the next best candidate is chosen.
iii) Old cluster head updates its state to CM.
iv) New cluster head issues a broadcast CJA. Cluster members update the Cluster_Head_ID.

## *5.2 Handling Dynamism*

In this section we show how 'Ripples' caters to network dynamism i.e. failure and join of sensor nodes (Req 4) as well as failure, join and movement of actor nodes.

### 5.2.1 Sensor Node Failure

If a cluster head fails, the $T_{CH}$ timer of the cluster members will eventually expire; the nodes will enter WC state with the wait period proportional to their distance from the center of the cluster, a cluster head will be chosen and cluster will be formed as in the IA state.
The worst-case time taken for the cluster to reform in this case is therefore $T_{CH} + \alpha * R_1$ or O(1).
If a cluster member fails, cluster head will not take any action until the failed node's turn arrives to become cluster head. The original cluster head will hand over charge to the next best candidate. There is no change in the network state in this case.
The failure of a sensor node therefore affects only the nodes in the cluster and, if the failed node is a cluster head, the immediate neighbors (the cluster head may be en route to an actor node). Since the cluster gets reformed in a finite amount of time, the effects are local in time as well.

### 5.2.2 Sensor Node Join

The new node will be in IU state. If it hears AAA first, it computes its desired cluster radius and enters the IA state. Since $F_R < F_{AA}$, it will receive a CJA before receiving the next AAA. When the node hears a CJA from the desired cluster, it joins the cluster. If the node hears AAA again before any CJA (there is no cluster in the desired area), it enters the WC state and eventually forms the cluster.
The worst-case time taken by the sensor node to join the right cluster is therefore $2*F_{AA}$ or O(1).

### 5.2.3 Actor Node Failure

If an actor node fails, the sensors associated with it will timeout after time $T_{AA}$. If other actor nodes exist, these sensors will listen to the periodic announcements from them via intermediate nodes and 'Bind' will ensure they get associated with the closest of them.
The worst-case time for a node to get reassociated with an actor node is $T_{AA}+F_{AA}*$Depth or O(Depth) where Depth if the distance in terms of number of clusters/ hops of the node from the new actor node.

### 5.2.4 Actor Node Join

The new actor node will announce itself with frequency $F_{AA}$. Nodes that are closer to this actor will change their association; others will retain their original association.
The worst-case time for a node to get associated with the new actor node is $F_{AA}*$Depth or O(Depth) where Depth is the distance of the node in terms of number of clusters/ hops from the new actor node.

### 5.2.5 Actor Node Movement

'Bind' checks the timestamp of AAA and updates the entry for the actor node as described earlier. The worst-case time in this case is the same as that in the case of Actor Node Failure or Actor Node Join.

**Observation 7:**

*'Ripples' is scalable to large network sizes.*

The sensor nodes only need to know their own location. Cluster heads need to know IDs of the cluster members, a maximum of six neighbors and one (or more for the purpose of redundancy) predecessor cluster head for selected number of actor IDs. All the operations dealing with perturbation are local in space as well as in time. The algorithm does not need global knowledge of the network. These characteristics make the algorithm scalable.

### 5.2.6 Demonstration of Actor Association and Infrastructure Maintenance Phase

In order to demonstrate the features part of the Infrastructure Maintenance phase as well as actor association, simulations were carried out using NS-2.27. NAM snapshots giving pictorial representation of these are shown in Appendix-B.

## 6 Variations to 'Ripples'

'Ripples' can be applied to a pure WSN scenario. The 'Actor' Node that initiates clustering could be the 'Base Station' or the 'Sink' node. Alternatively, some of the sensor nodes themselves can act as 'Initiator' nodes. Many methods have been proposed in the literature to select few of the nodes as initiators. A timer design method has been proposed in [2]. Our method can be used in conjunction with this work to let some sensor nodes initiate the clustering process.
If the application desires that the cluster heads remain close to the desired center of the cluster, our approach for clustering can be applied along with the 'Cell Shift' concept proposed in $GS^3$ – the nodes close to the center of the cell will be cluster heads; as these nodes die and the center becomes void, the cell structure can shift as a whole.
The cluster head rotation procedure can be synchronized such that the distance between two neighboring cluster heads remains nearly the same always, unless there is no node close to the desired position. This has the advantage of preserving sensing coverage with larger cluster radius.

## 7 Conclusions

In this paper, we presented 'Ripples', a message-efficient and sensing coverage-aware clustering algorithm with self-healing capability for wireless sensor and actor networks. The algorithm draws its strength from its deterministic behavior, ease of implementation and from considerably less control message overhead compared to most clustering algorithms proposed. We have presented detailed analysis of the algorithm's features. We also compared the message efficiency of 'Ripples' with that of 'Rapid' and 'Persistent', two of the efficient algorithms in the category of bottom-up approaches as well as with '$GS^3$', an algorithm with top-down approach. Simulation results demonstrate the algorithm's message efficiency and self-healing capability.
We focus on retaining uniform sensing coverage for the entire lifetime of the network. Coupled with its energy-saving operation of rotating cluster heads based on maximum energy left, the significantly less control message overhead can enhance the lifetime of the network substantially.
We have also outlined 'Bind' that works along with 'Ripples' and associates the sensor nodes with the actor node closest to them.
Since the number of neighbors as well as cluster size is fixed, relative direction of neighbors from self is known, and also the location of actor gets stored as the announcement comes, this can be exploited for routing of data with known delay. Frequency can be effectively reused across clusters. The medium access protocol can exploit the structure for allocating slots for synchronization between neighbors.

## *8 References*

[1] Y Yu, Krishnamachari B and Prasanna V K, "Energy-latency Tradeoffs for Data Gathering in Wireless Sensor Networks", In proceedings of IEEE INFOCOM 2004
[2] Krishnan R and Starobinski D, "Efficient clustering algorithms for self-organizing wireless sensor networks", Ad Hoc Networks, Elsevier, Article in Press, Corrected proof available online May 2004
[3] Ramamoorthy C V, Bhide A and Srivastava J, "Reliable clustering techniques for large, mobile packet radio networks", In proceedings of the 6th Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM'87), March-April 1987
[4] Zhang H and Arora A, "$GS^3$: scalable self-configuration and self-healing in wireless sensor networks", Computer Networks, Elsevier, Volume 43, Issue 4, November 2003, pp 459-480
[5] Akyildiz I.F and Kasimoglu I.H, "Wireless sensor and actor networks: research challenges", Ad Hoc Networks, Elsevier, Volume 2, Issue 4, October 2004, pp 351-367

[6] Ossama Younis and Sonia Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks", IEEE Tr on Mobile Computing, Vol 3, No 4, Oct-Dec 2004
[7] Adlakha, S. and Srivastava, M, "Critical Density Threshold for Coverage in Wireless Sensor Networks", Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC) 2003, Vol 3, pp 1615 - 1620
[8] Ahmed A.A, Shi H and Shang Y, "A survey on network protocols for wireless sensor networks", In proceedings of ITRE 2003, the IEEE International Conference on Information Technology, Research and Education, April 2003, pp 301-305
[9] Akyildiz I.F, Su W, Sankarasubramaniam Y and Cayirci E, "A survey on sensor networks", IEEE Communications Magazine, Volume 40, Issue 8, August 2002, pp 102-114
[10] Heinzelman W.B, Chandrakasan AP, Balakrishnan H, "An application-specific protocol architecture for wireless microsensor networks", IEEE Transactions on Wireless Communications, Volume 1, Issue 4, October 2002, pp 660-670
[11] Lindsey S and Raghavendra C.S, "PEGASIS: Power-efficient gathering in sensor information systems", In Proceedings of IEEE Aerospace Conference, March 2002, Volume 3, pp 3-1125 – 3-1130
[12] Liu J and Lin C.R, "Energy-efficiency clustering protocol in wireless sensor networks", Ad Hoc Networks, Elsevier, Article in press, corrected proof available online Dec 2003
[13] Bandyopadhyay S and Coyle E.J, "An energy-efficient hierarchical clustering algorithm for wireless Sensor Networks", In proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM'03), Vol 3, pp 1713-1723
[14] Cerpa A and Estrin D, "ASCENT: Adaptive self-configuring sensor network topologies", IEEE Transaction on Mobile Computing, Vol 3, No 3, Jul-Sep 2004, pp 272-285
[15] Hou, T.C and Tsai T.J, "An access-based clustering protocol for multihop wireless ad hoc networks", IEEE Journal on selected areas in communications, Vol 19, Issue 7, July 2001, pp 1201-1210
[16] Subramanian L and Katz R H, "An architecture for building self-configurable systems", In proceedings of the 1st ACM International Symposium on Mobile Ad Hoc Networking and Computing, November 2000
[17]: Krishnendu Chakrabarty and S.S. Iyengar, "Scalable Infrastructure for Distributed Sensor Networks", Springer-Verlag London Limited, Dec 1995. pp. 195
[18] Gupta H, Das S R, Gu Q, "Connected Sensor Cover: Self-Organization of Sensor Networks for Efficient Query Execution", Proceedings of MOBIHOC'03, June 1-3, 2003
[19] S Slijepcevic and M Potkonjak, "Power-Efficient Organization of Wireless Sensor Networks", In Proceedings of IEEE Intl Conf on Communications (ICC) 2001
[20] Melodia T, Pompili D, Gungor V C and Akyildiz IF, "A Distributed Coordination Framework for Wireless Sensor and Actor Networks", In the proceedings of MOBIHOC 2005, May 25-27, 2005
[21] S.S. Iyengar and R.R. Brooks (Editors), "Distributed Sensor Networks", Chapman and Hall/ CRC Press, Dec. 2004, pp. 1028
[22] Salhieh A, Weinmann J, Kochhal M and Schweibert L, "Power Efficient Topologies for Wireless Sensor Networks", Proceedings of 2001 ACM International Conference on Parallel Processing
[23] Wireless Sensor Networks: An Information Processing Approach by Feng Zhao and Leonidas Guibas, Elsevier Science Press
[24] The Network Simulator (NS-2) web site http://www.isi.edu/nsnam

# Appendix-A

**Algorithm:** 'Ripples'
**Given:** Network $G_A = (V_A, E_A)$ of actor nodes and $G_S = (V_S, E_S)$ of sensor nodes
**Purpose:** Form clusters of sensor network with the following properties: a) A cluster diameter is equal to the sensing range of the nodes. b) There is minimal overlap between clusters. c) The number of neighbors for each clusters is upper bounded by 6.
**Algorithm:**
**I. Actor Nodes:**

```
Procedure Actor_App
For v_A ϵ V_A at Loc_A = (x_A, y_A) and Depth_A = 0
begin
    for Every F_AA Cycle do
        Broadcast Actor Association Announcement(v_A, Depth_A, Loc_A)
    endfor
end
```

**II. Sensor Nodes:**

```
Procedure Sensor_App
For v_S ϵ V_S at Loc_S = (x_S, y_S)
begin
    State = INITIAL_UNAWARE
    for every T_NO Cycle do
        if (RECEIVED packet) then
            case (packet_type) of
                Actor_Association_Announcement:
                    if (State = INITIAL_UNAWARE) then /* First announcement */
                        Center = Compute_Ideal_Cluster_Center(Packet->Depth, Packet->Loc)
                        Distance = Euclidean distance of node from Center
                        if (Distance is not more than Desired_Cluster_Radius) then
                        /* This is to prevent nodes beyond desired boundary listening to this announcement from
                        joining an unsuitable cluster */
                            State = INITIAL_AWARE
                        endif
                    else if (State = INITIAL_AWARE) then          /* Second announcement */
                        State = WAITING_TO_CLUSTER
                        wait_period = Distance * Resolution       /* Resolution = α described in text */
                    else if (State = CLUSTER_HEAD) then
                        Depth = Packet->Depth + 1
                        Actor_ID = Packet->Actor_ID
                        Broadcast Actor Association Announcement(Actor_ID, Depth, Loc_S)
                    endif
                Cluster_Announcement:
                    if ((State = INITIAL_AWARE) or (State = WAITING_TO_CLUSTER)) then
                        if (Center = Packet->Cluster_Center) then
                            State = CLUSTER_MEMBER
                            Join the Cluster from where announcement was made
                            Send Ack to the Cluster Head
                        endif
                    endif
                Cluster_Ack:
                    Register the node as a member
            endcase
        endif
        if (State = WAITING_TO_CLUSTER) then
```

```
            Decremement Wait_Period
            if (Wait_Period = 0) then                    /* Become Cluster Head */
                Broadcast Cluster_Announcement(v_S, Center) /* With self ID and Ideal Center Loc */
                Broadcast Actor_Announcement(Actor_ID, Depth, Center)
                /* Further the Actor Announcement */
            endif
        endif
    endfor
end
Procedure Compute_Ideal_Cluster_Center(Originating_Depth, Originating_Loc)
begin
    if (Originating_Depth = 0) then            /* Special case */
        Center = Originating_Loc
    else
        loc1.x = Originating_Loc.x + 2 * Desired_Cluster_Radius * sin(60)
        loc1.y = Originating_Loc.y
        dist1 = Euclidean distance between self and loc1

        loc2.x = Originating_Loc.x + Desired_Cluster_Radius * sin(60)
        loc2.y = Originating_Loc.y + Desired_Cluster_Radius  + Desired_Cluster_Radius * cos(60)
        dist2 = Euclidean distance between self and loc2

        loc3.x = Originating_Loc.x - Desired_Cluster_Radius * sin(60)
        loc3.y = loc2.y
        dist3 = Euclidean distance between self and loc3

        loc4.x = Originating_Loc.x – 2 * Desired_Cluster_Radius * sin(60)
        loc4.y = Originating_Loc.y
        dist4 = Euclidean distance between self and loc4

        loc5.x = loc3.x
        loc5.y = Originating_Loc.y - Desired_Cluster_Radius  - Desired_Cluster_Radius * cos(60)
        dist5 = Euclidean distance between self and loc5

        loc6.x = loc2.x
        loc6.y = loc5.y
        dist6 = Euclidean distance between self and loc6
        Center is the loc corresponding to min of dist 1 to 6.
    endif
end
```

# Appendix-B

We now demonstrate individual features of 'Ripples' and 'Bind' through NAM snapshots.

***a) Clustering through 'Ripples'***

Fig B-1 shows clusters formed through the 'Ripples' algorithm. 100 sensors are distributed randomly over and area of 50mx50m. Circles have been superimposed for illustration.

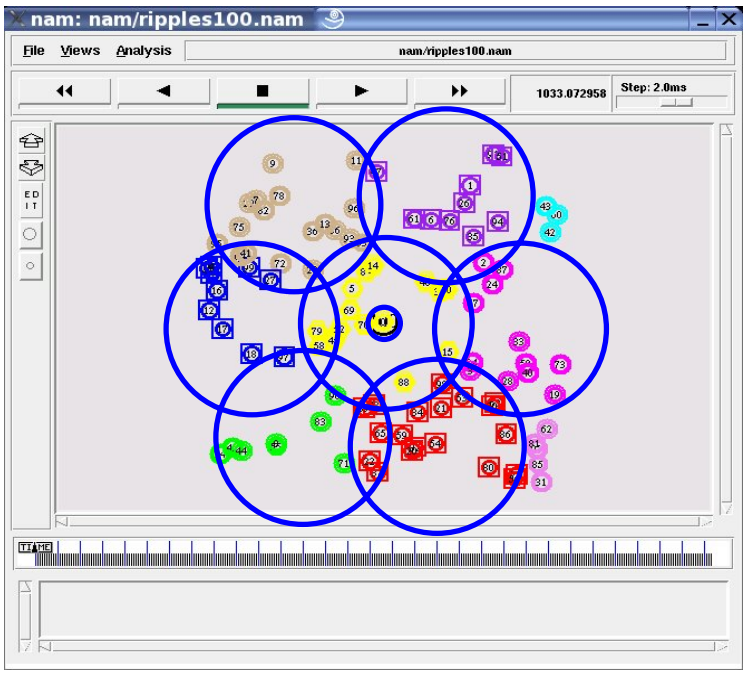



**Figure B-1: Clusters produced through 'Ripples'**

***b) Association of Sensor Nodes with Actors***

NAM snapshot shown in Fig B-2 has 100 sensor nodes and 2 actor nodes. The sensors attached to an actor are shown marked with same shape- circle/ square. If two actors are at the same distance, the one heard from first is selected for association. The association is made cluster-wise- the entire cluster is associated with an actor. Clustering is not shown separately here.

***c) Sensor Node Failure***

In Fig B-3, 5 sensors including a cluster head were failed. The snapshots show the scenario before and after the failure. The members where head failed have elected a new head and got reclustered.

***d) Sensor Node Join***

In figure B-4, 5 sensors were initially inactive. The snapshots show the scenario before and after the nodes were made active. The newly joined nodes have joined the appropriate clusters.

***e) Actor Node Failure***

In the scenario Fig B-5, shows the association before and after failure of the actor node at top right. Sensor nodes attached to the failed actor are now attached to the actor available.

***f) Actor Node Join***

In figure B-6 Actor 1 was originally inactive; left snapshot shows the scenario while clustering is still going on. The picture on the right shows snapshot taken after Actor 1 was made active. Sensors nodes have changed their association appropriately.

### *g) Actor Node Movement*

In Fig B-7, both actors were moved to new positions during simulation. The left and right snapshots show the scenario before and after the movement respectively.

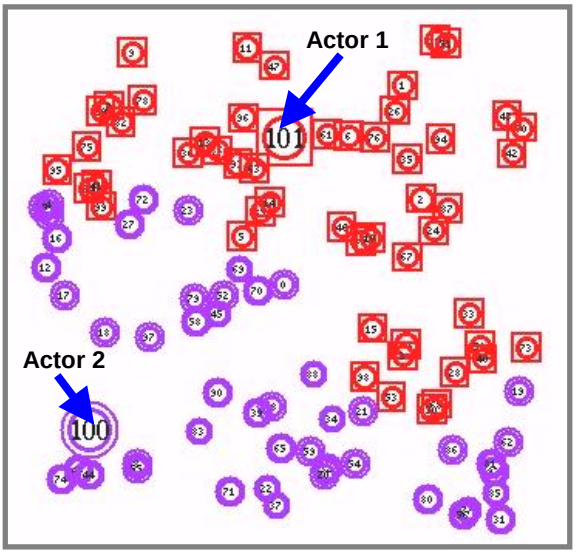



**Fig B-2: Association with closest actor**

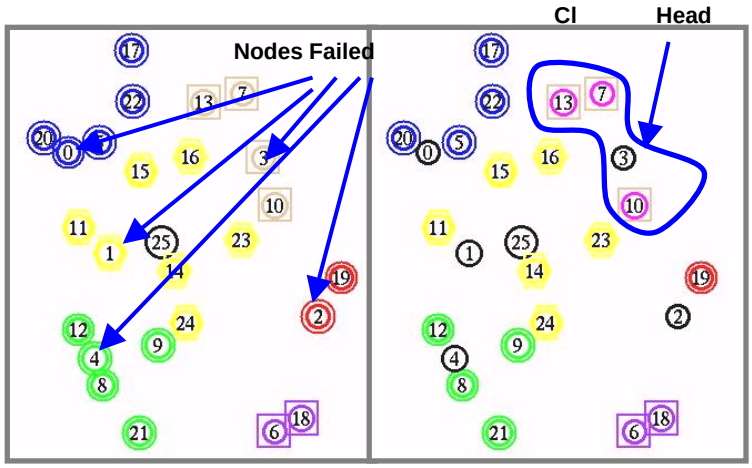



**Fig B-3: Sensor Node**

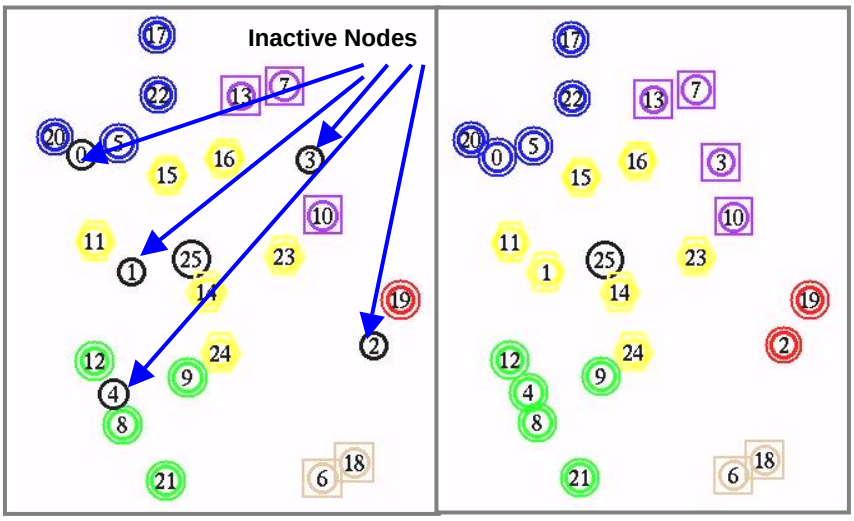



**Fig B-4: Sensor Node Join**

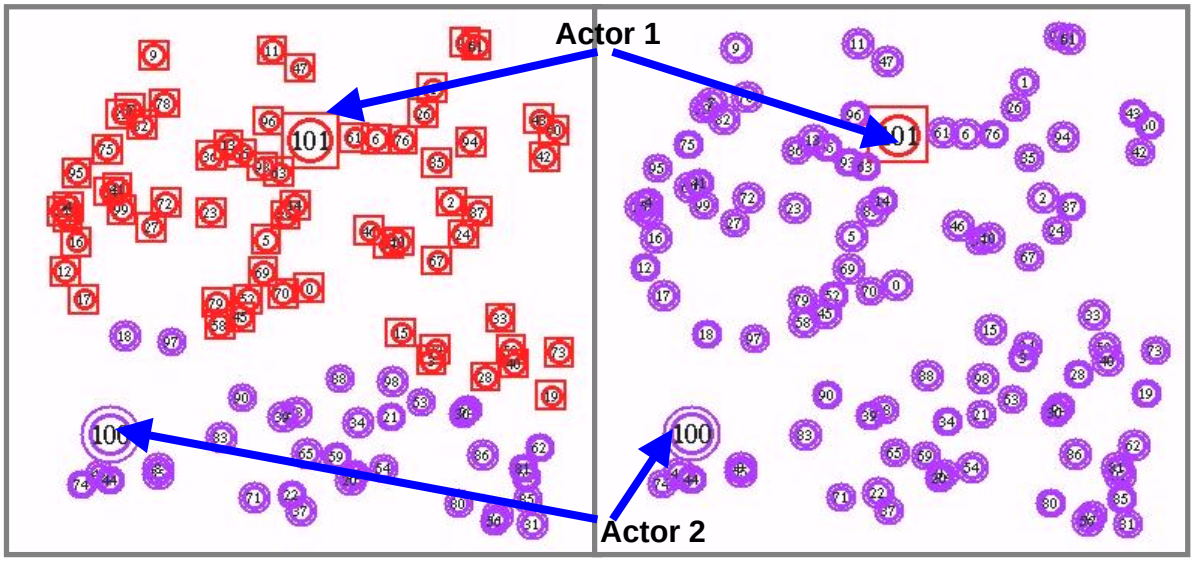



**Fig B-5: Actor Node Failure**

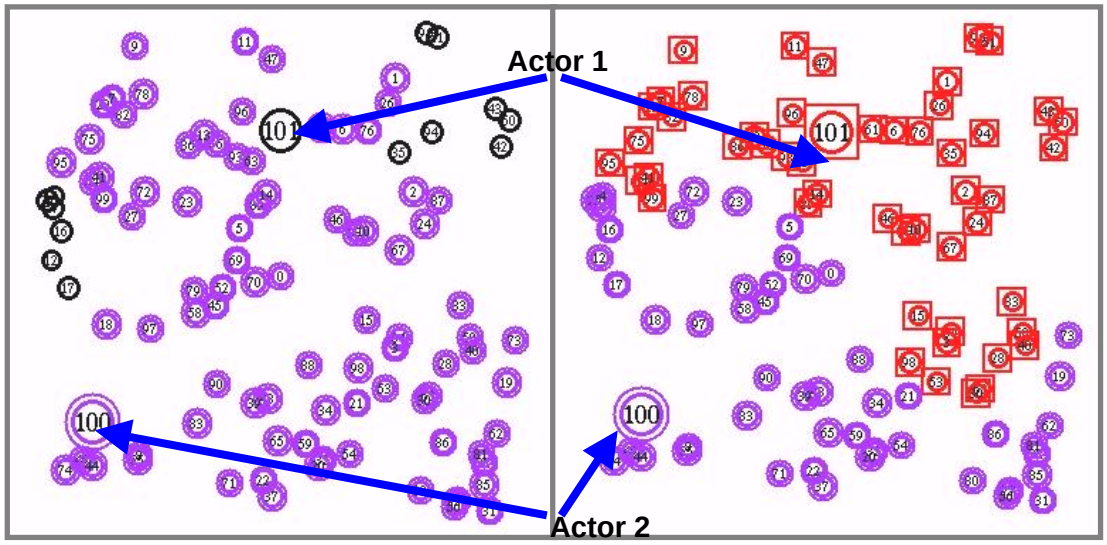



**Fig B-6: Actor Node Join**

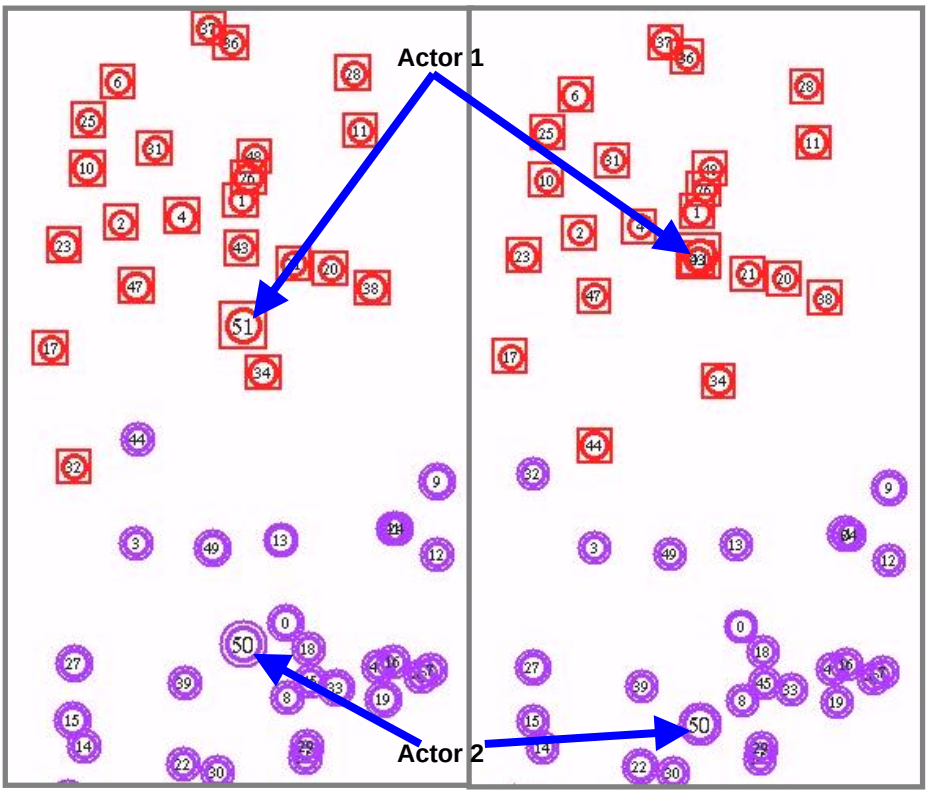



**Fig B-7: Actor Node Movement**