Clustering for Load Balancing and Energy Efficiency in IoT Applications

Shesha Sreenivasamurthy
Univ. of California, Santa Cruz
shesha@ucsc.edu

Katia Obraczka
Univ. of California, Santa Cruz
katia@soe.ucsc.edu

Abstract

This paper explores clustering as a technique to improve energy efficiency for a variety of current and emerging IoT application scenarios. We introduce a novel load balanced clustering algorithm based on Simulated Annealing whose main goal is to increase network lifetime while maintaining adequate sensing coverage in scenarios where sensor nodes produce uniform or non-uniform data traffic. To this end, we also introduce a new clustering cost function that accounts not only for sensor node traffic load but also for the cost of communicating over physical distances. Through extensive simulations comparing the proposed algorithm to leading state-of-the-art clustering approaches, we show that our algorithm is able to improve both network lifetime as well as network coverage by keeping more sensor nodes alive for longer periods of time at lower computational cost.

1. Introduction

The Internet has forever changed the way we live, communicate and interact, work, play, shop, move, etc. And, as the Internet marches steadily from being an internetwork of computers to becoming an internetwork of things, or an internetwork of everything, what not so long ago was considered a vision of the "Computer of the 21st Century" [1] has become reality. Mark Weiser's idea of the 21st Century Computer [1], which was later reframed as "embedding the world", "instrumenting the world" [2], and "connecting the world" [3] with the emergence of wireless sensor networks, predicts that computing will be so ubiquitous, seamless, and embedded into our world that it will "weave itself into the fabric of everyday life until it becomes indistinguishable from it" [1].

Indeed, according to [4], it is anticipated that by 2020, there will be close to 30 billion connected devices worldwide and that number is expected to at least double by 2025. Internet of Things (IoT) application such as Smart Homes, Smart Grids, Smart Buildings, Intelligent Transportation Systems, Smart Cities are some of the expected contributors to this tremendous growth which poses daunting challenges. These challenges are significantly magnified when considering that the increase in number of connected devices will also be accompanied by increased device and network technology heterogeneity, as well as increased administrative decentralization and application diversity.

We have been exploring clustering techniques as a way to tackle the various IoT scalability dimensions previously discussed. Clustering is widely applied in a variety of disciplines to address scalability issues. It has been used extensively in distributed systems and in computer networking. As discussed in more detail in Section 2, clustering has also been employed in wireless, multi-hop, ad-hoc networks (or MANETs), and particularly in Wireless Sensor Networks (WSNs) to not only improve scalability, fault tolerance, and load balancing, but also as an effective energy savings technique. In these networked environments energy efficiency is of critical importance since devices are typically resource anemic and do not have access to continuous energy sources. Several IoT applications share these same requirements and may even impose more stringent demands. For example, "massive IoT" scenarios such as smart buildings, smart metering, transportation logistics and fleet management, industrial and agricultural monitoring are characterized by a very large number of low-cost, power-anemic devices, while "critical IoT" applications are characterized by stringent availability, latency, and reliability demands [5].

Load balanced clustering where nodes generate non-uniform traffic load is known to be an NP-hard problem [6]. Approximation algorithms such as gradient descent, hill climbing, genetic algorithms, and simulated annealing can be used to find approximation solutions. Simulated Annealing's strength is that its stochastic component avoids getting caught in local optimal solutions. In this paper, we investigate low-cost approximation algorithms for energy efficient, load balanced clustering that target IoT applications. We propose a novel clustering approach based on simulated annealing which works in both uniform and non-uniform load scenarios. We compare our clustering algorithm against leading state-of-the-art clustering approaches, namely e.g., Load-Balanced Clustering [7] and Genetic Algorithm Based Clustering [8] [1]. When compared against these existing approaches, the proposed simulated annealing algorithm is able to provide improved load balancing while increasing the network's lifetime and reducing energy consumption. The contributions of our work can be summarized as follows:

- We introduce a new class of clustering algorithms based on Simulated Annealing that is well suited for IoT scenarios where data traffic load is uniform across all nodes as well as in scenarios that exhibit non-uniform traffic load.

- We propose a novel clustering *cost function* based on a combination of: distance between sensor node and clusterhead, distance between clusterhead and the data sink, traffic load generated by sensor node, and clusterhead's current load. We apply the proposed cost function to both our simulated annealing algorithm as well as Genetic Algorithm Based Clustering [7].

- We compare our algorithm's performance against Load-Balanced Clustering (LBC) [7] and Genetic Algorithm Based Clustering [8]. Our experimental results show that the proposed simulated annealing algorithm is able to achieve better load balancing amongst the clusterheads, thus extending network lifetime and reducing energy consumption.

The rest of the paper is organized as follows – Section 2 discusses related work, network and energy models are explained in section 3, cost functions in section 4, overview of generic simulated annealing algorithm is described in section 5, experimental setup and results are discussed in section 7 and concluded in section 8.

2. Background and Related Work

Clustering has been widely used in distributed systems as a way to achieve improved scalability, robustness, and load balancing. In computer networks and particularly in wireless, multi-hop, ad-hoc networks (or MANETs), clustering has also been employed to improve energy efficiency. For instance, in MANET deployments where some nodes are equipped with larger batteries or have energy harvesting capabilities (e.g., solar panels), such nodes can act as *clusterheads*. MANET nodes can then organize themselves in clusters around clusterheads which can act as traffic relays for their clusters. Clusterheads in MANET, which typically have access to constrained energy sources, aggregate data for their cluster. Besides saving energy, aggregation at clusterheads also allows data to be pre-processed before being transmitted to its ultimate destination.

Wireless sensor networks, or WSNs for short, can be considered a particular type of MANET, where some or all nodes also have sensing capabilities, besides processing, storage, and communication. Clustering has also been used in WSNs to improve energy efficiency and extend WSNs' lifetime. Some WSN clustering algorithms, such as [9, 10, 11, 12, 13], organize nodes into clusters based on their distance to the clusterhead. Using a pre-determined set of requirements, e.g., minimum battery capabilities, nodes that satisfy such requirements announce their presence as clusterheads and the other WSN nodes choose the closest clusterhead. Another WSN clustering technique is to virtually divide the network into a grid, where nodes at the center of each cell are selected as clusterheads. All nodes within the cell join the cell's clusterhead [14, 15, 16].

Algorithms that use uniform degree across all clusterheads as clustering criterion have been proposed in [7, 17]. However, when clusterhead degree is uniform, clusterheads that are farther from the WSN data sink end up spending more energy to transmit data to the sink when compared to clusterheads closer to the sink. To overcome this problem, non-uniform clustering has proposed, where clusters closer to the sink will have more members compared to clusters farther away [18, 19, 20, 21].

As the Internet continues its evolution from an internetwork of computers to an internetwork of things, connected applications and devices as well as the underlying network technology become increasingly more diverse. This calls for algorithms that consider characteristics that vary across nodes, such as data traffic. In [8], a genetic algorithm based clustering technique that accounts for non-uniform sensor node traffic loads is introduced. However, it does not consider sensor-to-clusterhead nor clusterhead-to-data sink distances. Additionally, genetic algorithms may fall into the local maximum/minimum problem. They can also be very onerous in terms of processing complexity and thus energy needs [22]. Several efforts have proposed modifications to standard genetic algorithms to overcome these problems [23, 24, 25].

In the remainder of this section, we provide a brief description of existing clustering techniques that are representative of two of the main classes of clustering algorithms for WSNs, namely Load-Balanced Clustering (LBC) [7] representing uniform degree clustering

[1] Our choice of representative clustering algorithms used in our comparative study is discussed in detail in Sections 2.1, 2.2, and 6.

and Genetic Algorithm Based Clusterings (GAC) [8]. In Section 7, these algorithms' performance is compared against our simulated annealing clustering technique.

2.1. Load-Balanced Clustering

Algorithm 1: Load Balanced Clustering

```
Input:
  S: List of Sensors
  C: List of Clusterheads
Output:
  A: Set of Sensors to CH assignment

LBC (S, C)

begin
    $A \leftarrow \emptyset$;
    // This loop creates the ESet of each CH
    foreach sensor in S do
        if sensor.reachability = 1 then
            // Assign sensor to the only CH in range
            ch ← Only CH in sensor's range;
            $A \leftarrow A \bigcup (sensor, ch)$;
            ESet[ch] ← ESet[ch] ⋃ sensor;
            RSet[ch].remove(sensor);
        end
    end
    // Sort CHs in ascending order of their cardinality
    Sort (C);
    // Expand ESet based on sensor's distance from CH
    foreach ch in C do
        CriticalDistance ← MinDistance (ESet[ch]);
        m ← Median (ESet[ch]);
        while CriticalDistance < m do
            sensor ← next sensor in ch's RSet;
            d ← Distance (ch, sensor);
            if d < CriticalDistance then
                // Assign sensor to ch
                $A \leftarrow A \bigcup (sensor, ch)$;
                ESet[ch] ← ESet[ch] ⋃ sensor;
                RSet[ch].remove(sensor);
            end
            CriticalDistance ← m;
            m ← Median (ESet[ch]);
        end
    end
    // Group remaining sensors based on reachability
    Groups ← Sensors grouped indexed by reachability;
    // Sort Groups in ascending order of reachability
    Sort (Groups);
    foreach G in Groups do
        while G != ∅ do
            sensor ← a sensor in G;
            ch ← MinimizeObjective (sensor);
            $A \leftarrow A \bigcup (sensor, ch)$;
            $G = G - sensor$;
        end
    end
    return A;
end
```

Load-Balanced Clustering (LBC) [7] assumes that all sensors and clusterheads know their relative positions either via GPS or pre-configuration. The LBC algorithm (shown in Algorithm 1) is executed by all clusterheads (CHs) and includes the following steps:

1. CH broadcasts `DiscoveryRequest` message to find sensors within its range.
2. CH waits for nodes to respond with `DiscoveryResponse` message, containing their current remaining energy level and position.
3. CH builds *RSet* containing information about all nodes within range and their distances. Exchange RSet among CHs using `RSetInfo` message.
4. CH executes clustering algorithm as shown in Algorithm 1.
5. CH broadcasts ClusterID to nodes in its cluster as determined by Algorithm 1 using `ClusterIt` message
6. Nodes update their CH accordingly.

Assuming that all RSets will be received by all CHs, they will all have consistent information about every CH's directly connected nodes. Thus, at the end of their clustering computation, all CHs will end up clustering nodes in exactly the same way. Upon receiving RSets from the other CHs, a CH will assign sensors that have only one clusterhead within their range (*reachability* = 1) to that CH as they do not have any other choice. When a sensor is assigned to a CH, it will be added to that CH's *ESet*. CHs are then sorted in ascending order of their cardinality, which is defined as the number of sensors in the CH's transmission range. This is to avoid CHs with lower cardinality, to have their cardinality, i.e., number of sensors, be reduced even further, which could lead to higher load imbalance.

Next, CHs' *ESet*s are expanded by adding sensors whose distance to the CH is less than the *critical distance*. Initially, the critical distance to a CH is set to the minimum distance between each node in the ESet and the corresponding CH, and is gradually updated to the median of distances in the ESet. Lastly, the remaining sensors are sorted in ascending order of their reachability. They are then assigned to clusterheads with the objective of minimizing the CHs' cardinality (load) variance.

2.2. Genetic Algorithm Clustering

Genetic Algorithms [26] are a class of heuristic-based stochastic algorithms inspired by evolution's natural selection process. It is generally used in NP-hard problems where the search space for the optimal solution is large (e.g., such as traveling salesman [27, 28]). In genetic algorithms, a set of all solutions to a problem, called *Population*, and a random subset of that population (*Initial Population*) are considered for further processing. Members of the population are called *Samples*, where each sample has a set of properties (or *chromosomes*). Each property is represented as a number and therefore, the set of properties is represented as an array of numbers. The properties of a pair of samples (*parents*) are altered systematically (*crossover*) or

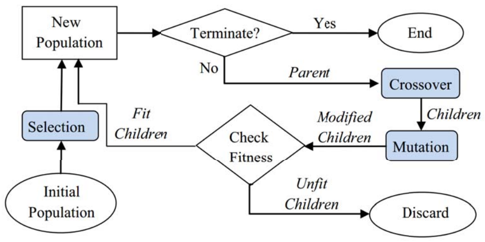


Figure 1: Flowchart of Genetic Algorithm [8]

mutated randomly (*mutation*) to produce new samples (*children*). The altered solutions become next generation population and only samples that have better *fitness* than the current population survive and others die. The goal of performing crossover and mutation operations is to arrive at a better solution defined using a *cost or fitness* function. The algorithm stops when a *termination* conditions such as, failure to find better solutions or achieving a pre-defined maximum number of iterations are met. A flowchart of the standard genetic algorithm is as shown in Figure 1.

In Kuila et.al [8], genetic algorithms are applied to the problem of clustering with unequal load. In their algorithm, which we refer to as GAC, a *sample* is defined as a valid sensor to CH assignment for all sensors in the network. A valid assignment is when a sensor is assigned to a CH that is within its communication range. First, a *population* of 200 valid random samples is generated. Of the 200 samples, the top 10% best fit samples are included in *BSet*, where the fitness $\frac{1}{\sigma}$ is defined by equation 1. At each iteration of GAC, a crossover operation is performed on two random samples selected from BSet to produce two children. The crossover point for each crossover operation, which is performed 75% of times, is chosen randomly. The two children thus produced are mutated 5% of times. Mutation is done as follows: first, a sensor is chosen randomly from a maximally loaded CH and is assigned to a CH with minimal load that is within the sensor's communication range. After crossover and mutation, children are carried over to the next generation if their fitness is better than every sample in the current generation. An equal number of older generation samples is removed from the population. The experiment continues until no better children are produced for successive 3 rounds or a maximum of 500 iterations have been completed. The fitness function is given by -

$$Fitness = \frac{1}{\sigma}, \text{ where} \qquad (1)$$
$$\sigma = \sqrt{\frac{\sum_{j=1}^{m}(\mu - W_j)^2}{m}}$$

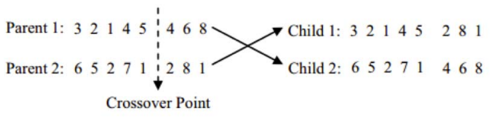


Figure 2: Crossover Operation [8]

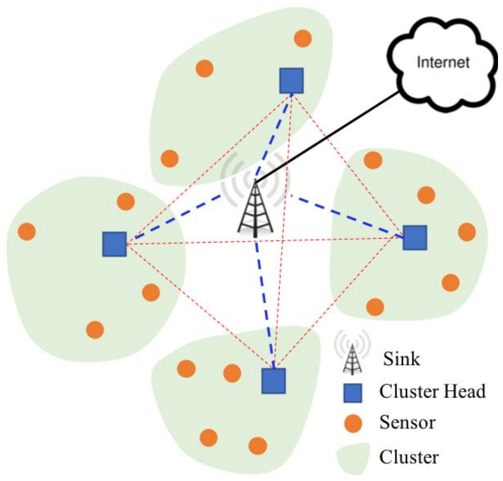


Figure 4: System Model: Nodes, Clusterheads, and Sink

where, μ(average load) $= (\sum_{i=1}^{n} d_i)/m$, d_i is the load of sensor s_i, W_j is the overall load of the clusterhead c_j, n is the number of sensors and m is the number of clusterheads. The crossover and mutation operations are illustrated in Figures 2 and 3, respectively.

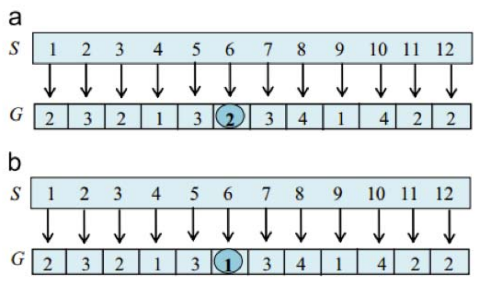

a
S 1 2 3 4 5 6 7 8 9 10 11 12
G 2 3 2 1 3 2 3 4 1 4 2 2
b
S 1 2 3 4 5 6 7 8 9 10 11 12
G 2 3 2 1 3 1 3 4 1 4 2 2

Figure 3: Mutation Operation [8]

3. System Model

In this work, we focus on IoT applications in which nodes are essentially stationary. Examples of such applications include Smart Environments such as Smart Homes, Smart Buildings, Smart Grids, Industrial and Agricultural Automation, etc. As such, in our system model we consider three components as illustrated in Figure 4: 1. *sensor nodes* that have sensing capabilities and generate different amounts of data (or *load*), 2. *clusterheads (CHs)* which are equipped with more computing power and larger batteries (or have access to continuous energy sources) than *sensor nodes* , and 3. *data sinks* which are typically the ultimate destination of data generated by *sensor nodes* and connect to the Internet. Note that while there is a single data sink in Figure 4, multiple data sinks can be used.

3.1. Network Model

Given the IoT applications we target in this work, we assume CHs are one hop distance from a sink. This is accomplished either by placing data sinks closer to clusters, and using multiple data sinks in deployments covering a larger area, and/or relying on the fact that CHs have radios that are powerful enough and thus have longer transmission range. We also assume that CHs are within each other's communication range. Note that these assumptions do not affect the generality of the proposed clustering mechanism. They mainly simplify how CHs communicate amongst themselves and with sinks since no routing mechanism is required.

For energy efficiency, one of the clustering criteria is that sensor nodes in a cluster are one hop distance from the CH. As previously noted, nodes are stationary and once deployed, their positions will not change. Each sensor node is assumed to know its geographic location position either through some localization mechanism or pre-configuration during installation. Sensor nodes share their position information with the CHs as part of clustering.

Each cluster is managed by only one CH and sensor nodes are members of only one cluster at any given time. However, a sensor node can be in the transmission range of one or more CHs. It is assumed that sensor nodes periodically send data to their CH and any required signaling information can be piggybacked onto sensor data.

The main goal of our clustering approach is to extend network lifetime, which we define as the number of data transfer rounds until the first CH fails. Once clusters are formed, re-clustering is triggered when sensor failures (e.g., due to energy depletion) are detected.

3.2. Energy Model

Our energy model is similar to the one in [9, 7, 8], but we introduce extensions to handle the case of unequal load, i.e., when sensor nodes generate different amounts of traffic. As such, total energy consumed to transmit information includes – 1. Energy consumed by sensing and processing which depends on the amount of data sensed and 2. Energy consumed for actual transmission which depends on the distance between data source and destination as well as the amount of data being transmitted. Energy consumed for receiving data depends only on the amount of data received. It is assumed that all data sensed is transmitted and there is no loss of information during transmission. These are represented by Equation 2 below. The notation used in our energy model is summarized in Table 1.

Variable	**Description**
E_{tx}	Energy for transmitting
E_{rx}	Energy for receiving
d	Euclidean distance separating nodes
b	Data transmitted (a constant for equal load)
b_{s_i}	Data transmitted by sensor s_i
b_{c_i}	Data transmitted by clusterhead c_i
l_{s_i}	Load co-efficient of sensor s_i
S_{c_i}	Members of clusterhead c_i
M	Maximum packet size

Table 1: Energy model notation

$$\begin{aligned} E_{tx} &= b(E_p + E_c d^2) \\ E_{rx} &= bE_p \end{aligned} \tag{2}$$

When, the load generated by each sensor is considered equal, b is a constant. However, in an unequally loaded system, b will be different for each node in the system. If $l_{s_i} \in \mathbb{Q}$, where $\mathbb{Q}$ is the set of rational numbers between 0 and 1, represents the load of sensor node s_i and $M \in \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, is the maximum data size, then the amount of data transmitted by sensor node s_i is given by -

$$b_{s_i} = Ml_{s_i} \tag{3}$$

If clusterhead c_i's cluster contains the set of sensor nodes S_{c_i}, then the amount of data transmitted by c_i to the sink is given by -

$$b_{c_i} = M \sum l_{s_i} \forall s_i \in S_{c_i} \tag{4}$$

Note that transmission distances between sensor nodes and clusterheads and clusterheads and data sinks are given by the euclidean distances between them.

4. Problem Formulation

In this section we formulate the clustering problem that we are trying to solve and introduce a novel cost function that is based on a combination of: distance between sensor node and clusterhead, distance between clusterhead and data sink, traffic load generated by sensor node, and clusterhead's current load. While earlier works consider only a subset of these parameters, we argue that in heterogeneous IoT environments, it is important to consider all of them. Table 2 summarizes the notation used in the formulation of our problem as well as our cost function. We start by summarizing our assumptions as follows:

- Sensors may generate non-uniform amounts of data.

- Clusterheads are equipped with more computing power and larger batteries than sensor nodes.
- Sensors are at one-hop from their clusterheads.
- Clusterheads are one-hop from each other and sink/s.
- All nodes in the network are stationary.
- Sensors are aware of their geo positions.
- Each cluster is managed by only one clusterhead.
- Sensors periodically send data to the clusterhead.
- All sensed data is transmitted.
- There are no losses when data is transmitted.
- Clusterheads do not perform data compression.
- The distance between any two nodes is given by their euclidean distance.
- Sensors fail only due to energy depletion.

Let $S = \{s_1, s_2, ..., s_n\}$ be the set of n sensor nodes and $C = \{c_1, c_2, ..., c_m\}$ be set of m clusterheads (CHs) in the system, where $n >> m$. Each CH c_j finds the set of sensor nodes $P_j = \{s_1, s_2, ..., s_p\} | P_j \subseteq S$, that are in its communication range. Using this information, which is exchanged among all CHs, each CH c_j constructs the set of CHs $Q_{s_i} = \{c_1, c_2, ..., c_q\} \forall s_i \in S | Q_{s_i} \subseteq C$, that are in communication range of s_i. Therefore, at the end of this process each c_j will know the set of CHs in communication range of each s_i.

The problem is then to assign each sensor node s_i to CH c_j in order to maximize overall fitness, where fitness is defined as follows. At any given time, let $R_{c_j} = \{s_1, s_2, ..., s_k\} | R_{c_j} \subseteq S$, be the set of sensor nodes currently assigned to c_j. Fitness resulting from assigning s_i to c_j, $F(s_i, c_j)$, is given as the reciprocal of the total cost $TCost(s_i, c_j)$ of assigning s_i to c_j, i.e.,

$$F(s_i, c_j) = \frac{1}{TCost(s_i, c_j)} \tag{5}$$

Proposed cost function

The fitness maximization problem is converted to a cost minimization problem. As previously discussed, our proposed cost function cost accounts for: (1) Distance between sensor node and CH, (2) Distance between CH and data sink, (3) Load generated by sensor node, and (4) CH's current load. As such, the cost incurred by CH c_j, if it accepts sensor node s_i as a member of its cluster is given by -

$$\begin{aligned} Cost(c_j : s_i) = d(c_j, sink) \times \frac{\sum_{k=0}^{n} load(s_k)}{m} + \\ load(c_j) \times \frac{\sum_{k=0}^{m} d(c_k, sink)}{m}, \text{ where} \\ load(c_j) = \sum load(s_k) \; \forall s_k \in R_{c_j} + \\ load(s_i), \text{where } s_i \notin R_{c_j} \end{aligned} \tag{6}$$

The cost incurred by sensor node s_i if it accepts c_j as its CH is given by -

$$\begin{aligned} Cost(s_i : c_j) = d(s_i, c_j) \times \frac{\sum_{k=0}^{n} load(s_k)}{n} + \\ load(s_i) \times \frac{\sum_{k=0}^{m} d(s_i, c_k)}{m} \end{aligned} \tag{7}$$

As a result, the total cost, $TCost(s_i, c_j)$, is given by -

$$\begin{aligned} TCost(s_i, c_j) = \\ Cost(c_j : s_i) + \frac{(n+m)}{n} \times Cost(s_i : c_j) \end{aligned} \tag{8}$$

Generally, $d(c_j, sink) > d(s_i, c_j)$ and $load(c_j) > load(s_i)$. Therefore, $Cost(s_i : c_j)$ is multiplied by $\frac{(n+m)}{n}$ so that the total cost is not dominated by $Cost(c_j : s_i)$.

It is worth pointing out that the proposed cost function was motivated by some of our experimental results as reported in Section 7. More specifically, we noticed that when distance between nodes was not considered when accounting for cost in clustering, we did not observe significant improvements in network lifetime even after achieving better load balancing across clusterheads.

Applying new cost function to GAC

We applied our cost function to the Genetic Algorithm Clustering approach (GAC) [8] described in Section 2 with minor modification. In case of GAC, fitness of an entire sample should be evaluated, where a sample is a random valid assignment for all n sensor to one of m clusterheads. This is because, unlike simulated annealing, the solution is not built incrementally by evaluating a random assignment of sensor to clusterhead, rather a random valid assignment of all sensors to clusterheads is considered and fitness of entire solution is evaluated. Valid assignment means sensor s_i and clusterhead c_j pair are within communication range of each other. Adapted cost function for GAC is given by -

$$\begin{aligned} TCost(sample) = \sum_{\forall c_j \in C} \sum_{\forall s_i \in R_{c_j}} Cost(c_j : s_i) + \\ \frac{(n+m)}{n} \sum_{\forall c_j \in C} \sum_{\forall s_i \in R_{c_j}} Cost(s_i : c_j) \end{aligned} \tag{9}$$

5. Our Approach

Our approach to solving the problem of clustering in an unequal load environments uses simulated annealing (SA) [29] which is a well-known approximation technique to solve NP-hard problems, where the solution search space is quite large. While stochastic algorithms like hill climbing may get caught in local optima solutions, SA is designed to overcome the local optimum problem by accepting locally non-optimal solution with

Variable	Description
s_i	Sensor s_i
c_j	Clusterhead c_j
C	Set of all clusterheads
R_{c_j}	Set of sensors currently assigned to c_j
n	Number of sensors
m	Number of clusterheads
$F(s_i,c_j)$	Fitness resulting from assigning s_i to c_j
$TCost(s_i,c_j)$	Total cost of assigning s_i to c_j
$Cost(c_j,s_i)$	Cost incurred by c_j if s_i is accepted as its member
$Cost(s_i,c_j)$	Cost incurred by s_i if c_j is chosen as its clusterhead
$d(s_i,c_j)$	Euclidean distance between s_i and c_j
$d(c_j,sink)$	Euclidean distance between c_j and $sink$
$load(s_i)$	Load co-efficient of s_i

Table 2: Variables in cost functions

non-zero probability. As illustrated in Figure 5, hill climbing will terminate when it finds that solution-2 is less optimal than solution-1. However, SA may accept solution-2 with some probability hoping to find a solution better than solution-1, e.g., solution-3.

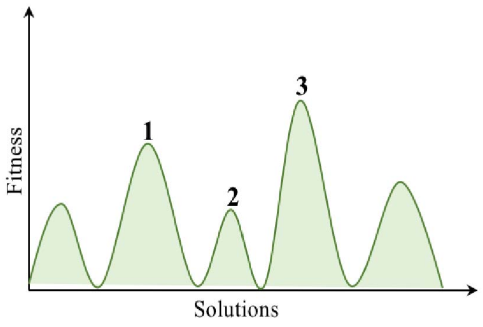


Figure 5: Simulated Annealing

Simulated Annealing's name is inspired by the physics of metal annealing, where metals are heated and cooled slowly in order to change their physical properties. Initially, the temperature is set to a high value and is gradually cooled. The rate of cooling determines the number of iterations the algorithm executes before stopping. At every iteration, a new solution is found at random. In our case, a new valid sensor-to-clusterhead assignment is chosen according to the *acceptable probability, AP* given by Equation 10. In other words, if the new solution is the best fit so far (i.e., new solution's cost is the lowest so far), then that solution is always accepted (i.e., with probability 1). Otherwise, the solution is accepted with probability $AP < 1$ calculated as a function of the solution's "fitness" which is evaluated using our proposed cost function described in Section 4. Initially, i.e., when the annealing temperature is high, non-optimal solutions are accepted with higher probability as the solution space has not been well explored and thus the probability of finding a better solution is higher. However, the *AP* decreases as the annealing temperature cools down. When the temperature cools down to zero, the current set of sensor-to-clusterhead assignments is accepted as the final solution.

$$AP = \begin{cases} 1 : if\ new\_cost < current\_best\_cost \\ e^{\frac{-(new\_cost - current\_best\_cost)}{T}} : \text{otherwise} \end{cases} \tag{10}$$

Besides the new cost function (described in Section 4), we also modified the standard simulated annealing algorithm to be executed in a distributed fashion by all clusterheads who broadcast the results of their computation to the members of their clusters. It should be noted that in order to guarantee that the simulated annealing computation carried out in all clusterheads yields the same result, identical seeds for the pseudorandom number generator are used. Additionally, all clusterheads must have consistent information about the network topology. Both of these requirements are achieved during step (2) of our distributed simulated annealing algorithm, whose pseudocode is shown in Algorithm 2.

As illustrated in Algorithm 2, our clustering mechanisms consists of 5 steps as described below. These steps are similar to the steps used in the Load-Balanced Clustering approach as described in Section 2.1, except for the clustering phase, in which we use our simulated annealing algorithm.

1. `DiscoveryPhase`: Clusterheads broadcast `DiscoveryRequest` messages to find sensor nodes within its range. Then, they wait for nodes to respond with a `DiscoveryResponse` message, containing the node's current remaining energy level and position.
2. `RSetExchangePhase`: Clusterheads build their *RSet* containing information about all nodes within range and their distances. Additionally, each clusterhead also adds a random seed to *RSet*. The random seed of the lowest-ID clusterhead will be used by all clusterheads as their pseudo-random number generator seed for their Simulated Annealing computation. This ensures that all clusterheads cluster sensors in exactly the same way. RSet's are exchanged among clusterheads using `RSetInfo` messages.
3. `ClusteringPhase`: Our simulated annealing algorithm described in Algorithm 2 is executed in this phase.
4. `CompletionPhase`: In this phase, each clusterhead informs nodes in their cluster that it is their parent by broadcasting their ClusterID using

`ClusterIt` message.

5. `MaintenancePhase`: In this phase, periodic heartbeats are exchanged amongst clusterheads. On a sensor failure, the clusterhead of the failed sensor informs other clusterheads of the event and all clusterheads execute the steps above starting at the `RSetExchangePhase`.

Algorithm 2: Load Balanced Clustering with Equal and Unequal Loads

```
Input: S: List of Sensors
Output: A: Set of Sensors to CH assignment

NEW(S)
begin
    A ← ∅;
    Shuffle(S);
    foreach sensor in S do
        chlist ← List of CHs in sensor's range;
        if chlist.length = 0 then
            Mark sensor dead;
            continue
        end
        else if chlist.length = 1 then
            /* Assign sensor to chlist[0] */;
            A ← A ⋃ (sensor, chlist[0]);
            continue
        end
        Shuffle(chlist);
        ch ← Pop first element from chlist;
        cost ← TCost(sensor, ch);
        T ← 1.0;
        $T_{min}$ ← 0.0001;
        $\alpha$ ← 0.9;
        while $T > T_{min}$ & chlist.length > 0 do
            $ch_t$ ← Pop first element from chlist;
            newcost ← TCost(sensor, $ch_t$) ;
            P ← AP(cost, newcost, T);
            if P > random() then
                ch ← $ch_t$;
                cost ← newcost;
            end
            T ← $\alpha T$;
        end
        /* Assign sensor to ch */;
        A ← A ⋃ (sensor, ch);
    end
    return A;
end
```

6. Experimental Methodology

We experimented with our simulated annealing clustering algorithm using a Python custom simulator. To conduct a comparative performance study, besides our own clustering algorithm, we also implemented Load-Balanced Clustering (LBC) [7], Genetic Algorithm Clustering (GAC) [8], and GAC using our proposed cost function as described in Section 4. We chose to use LBC and GAC in our comparative performance study since they have been extensively referenced as representatives of the non-stochastic and stochastic clustering mechanisms, respectively. We compare the following metrics in our study -

1. Standard deviation of load on all cluster heads after clustering. The load on the clusterhead will be the sum of loads of all its cluster members.
2. Network lifetime, which we define as the number of data transfer rounds until the first CH fails. One round includes data transfer from all sensors to the sink via their clusterheads.
3. Network energy consumption, which is equal to the sum of energy expended by all sensors and clusterheads in the network.
4. Rate of sensor failure.
5. Clustering time, which is the total time consumed for the convergence of the clustering algorithm.

To implement the different clustering algorithms under study, our simulator employs a modular design so that most of the code except for the core clustering algorithms is reused for all mechanisms tested. The simulator supports *save* and *restore* features, which are used to save and restore system information after bootstrapping, so that different algorithms can be executed with the same scenarios (e.g., exactly the same node placement, etc). It can also be used to save information after clustering is performed that can be loaded by a visualization tool to display resulting clusters (as shown in Figures 8 and 9.

Simulation parameters and the values used in our experiments are summarized in Table 4. Sensor nodes within range of clusterheads and vice-versa are determined based on the maximum transmission range and euclidean distance separating them. This forms the RSet of clusterheads and sensors, respectively. To model traffic load generated by sensor nodes, each node's load coefficient (l_{s_i}) is randomly generated between 0 and 1 and do not change in the course of the experiment.

The network's total energy consumption is calculated as the sum of the energy consumed by cluster formation and data transfer as follows:

- **Energy consumed during cluster formation:** each clusterhead subtracts the amount of energy required to transmit one `DiscoveryRequest` message, receive '*r*' `DiscoveryResponse` messages, where '*r*' is the cardinality of clusterhead's RSet, transmit one `RSetInfo` message, receive '$m-1$' `RSetInfo` messages, where '*m*' is the number of clusterheads, and one `ClusterIt` message. Each sensor node subtracts the amount of energy required to receive '*q*' `DiscoveryRequest` messages, where '*q*' is the cardinality of sensor's RSet, transmit one `DiscoveryResponse` message and one `ClusterIt` message. For control messages, we use a maximum length of 32bytes and we use the maximum transmission range as the distance separating the nodes. Energy consumed to transmit and receive information is determined according to the energy model described in Section 3.2.

- **Energy consumed when transferring data:** the amount of data transmitted by each sensor node is calculated as the product of the node's traffic load l_{s_i} and the maximum data size M. The energy consumed e_{s_i} by each sensor node is thus equal to the energy consumed to transmit data of length $M * l_{s_i}$ over the distance separating the sensor node and its clusterhead. At every round, each sensor node reduces its energy e_{s_i}, which will be different for each sensor. The reduction of energy continues until energy of sensor nodes and/or clusterheads become zero. At this point the sensor node(s) or clusterhead(s) are considered to have failed. A sensor node failure triggers all clusterheads to start at the `RSetExchangePhase` as explained earlier. Data transfer continues until a clusterhead fails, which terminates the experiment. In our current experiments, energy depletion is considered as the only mode of failure.

GAC Parametric Sensitivity

The GAC's implementation reported in [8] uses a single set of values for the algorithm's parameters as described in Section 2.2. We then wanted to investigate how sensitive GAC's performance, i.e., load balancing and network lifetime (rounds of data transfer), is to these parameters. We experimented with 384 different combinations with various population sizes (100, 200, 350, 500), BSet percentages (10, 20, 40, 50), crossover percentages (50, 70, 90), mutation percentages (5, 10, 20, 40), and number of maximum iterations (500, 1000). It turns out that the algorithm is not significantly sensitive to these parameters as shown in Table 3, as more than 95% of the experimental results were within two standard deviations. Therefore, we used the same experimental parameters as in [8].

	AVG	SD	Within 1-SD	Within 2-SD
Load	4.16	0.05	67.19%	95.31%
Rounds	4217.19	181.74	67.45%	96.10%

Table 3: GAC algorithm's sensitivity to experimental parameters

We also explored modifications to GAC to overcome the local optima problem as proposed in [23, 24, 25]. To this end, we perturbed the algorithm by carrying some genetically weak children to the next generation with different probability (i.e., 0, 10, and 25%) for various numbers of CHs (10, 15, 30, 45) using 300 sensor nodes and averaged over 100 runs. We did not observe any significant changes in the results when compared to the original GAC approach which carries only better children to the next generation. The standard deviation of load (0.0199) was just 0.25% from average load (8.275) and standard deviation of network lifetime (24.53) was 0.83% from average network lifetime (2948.18). These results show that simple perturbations to the original GAC algorithm do not significantly impact its performance. More detailed experimentation with different optimization techniques will be considered as part of our future work.

7. Results

We have conducted extensive simulations to evaluate our proposed simulated annealing based clustering algorithm. In this section, we present our experimental results which were obtained by running two sets of experiments, one where distances between nodes were not considered when computing the cost function and the other set where node distances were accounted. We start by describing the experimental setups used.

7.1. Experimental Setup

As discussed in Section 6, we experiment with 4 different clustering algorithms - 1. LBC [7], 2. GAC [8], 3. GAC with our cost function described in Section 4, labeled "GANC", and 4. our simulated annealing algorithm, labeled "NEW". Results presented here are averaged of 100 runs. We ran the algorithms with both equal and unequal loads, varying the number of sensors from 100 to 350 (in increments of 50) and using $10, 15, 30, 45$ clusterheads. We assume the network area to be 200x200 meter^2, which approximately the size of a large office building or a medium shopping mall. Position of sensors and clusterheads are randomly chosen and do not change as mobility is not considered and sink is positioned at the center (100, 100). Transmission ranges of protocols like Bluetooth, WiFi and 802.15.4 transmission range are approximately between 10 and 100 meters [30, 31]. Therefore maximum transmission range of sensors is set to 85m. In order for the clusterhead to communicate amongst each other, the maximum transmission distance is set to 300m, which is approximately the length of the diagonal of the network area and can be achieved using long-range protocols like LTE [32] or LoRa [33]. Remaining simulation parameters such as sensors' and clusterheads' initial energy, E_c, E_p and message sizes are set to be the same values as used by previous researches [9, 11, 8]. Table 4 summarizes our experiments' simulation parameters and their values.

7.2. Load-balancing without node distances

In this first set of experiments, we focus on improving load balancing, i.e., reducing load variance, across clusterheads. As such we do not account for distance between nodes when computing clustering cost. We only show results using 30 clusterheads as we do not observe

Parameters	Values
Area	200x200 m^2
Load Type	Equal & Unequal
Sink position	(100,100)
Sensor & Clusterhead position	Random
Sensors	100-350
Clusterheads	10-45
Max sensor range	85m
Max clusterhead range	300m
Initial sensor energy	2 joules
Initial clusterhead energy	10 joules
Load co-efficient per sensor	Unique random value $\in [0,1)$
E_c	50x10^{-9} joules/bit
E_p	10x10^{-12} joules/bit
Max data size	256 bytes
Max control message size	32 bytes
Number of simulation iterations	100

Table 4: Simulation parameters

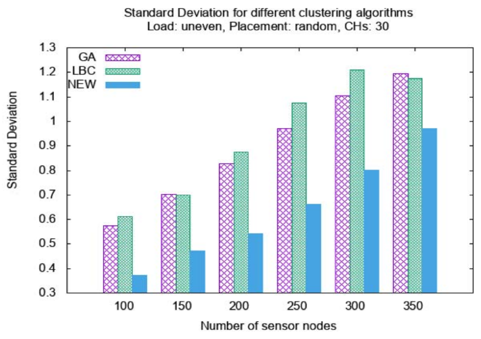


Figure 6: Standard deviation in clusterhead load without considering distance in cost metric

significant changes when we vary the number of CHs. As shown in Figure 6, we observe that NEW yields significant improvement in load balancing (up to 40% less load variance) when compared to LBC and GAC. The gap in load variance reduction increases with the number of sensors.

However, as shown in Figure 7, there was no significant improvement in NEW's network lifetime. As expected, network lifetime decreases as number of sensor nodes increases which causes the load on the clusterheads to increase.

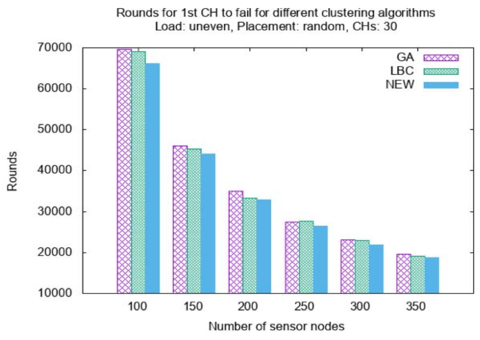


Figure 7: Network lifetime without distance in cost metric

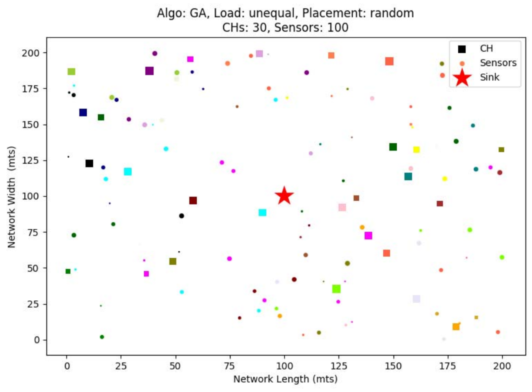


Figure 8: Clusters formed by GAC algorithm minimizing clusterhead load variance for sensors with unequal load.

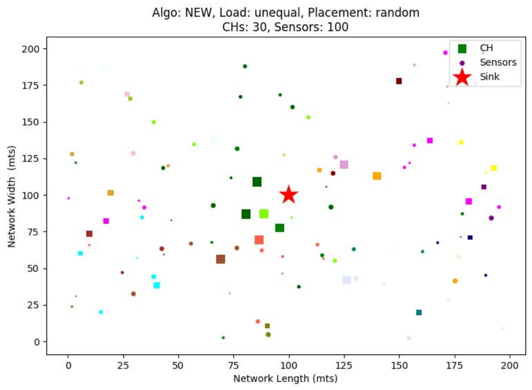
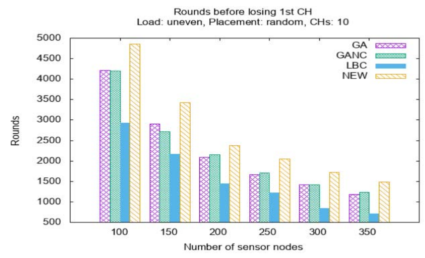


Figure 9: Clusters formed by NEW algorithm minimizing cost function using load and distance metrics for sensors with unequal load

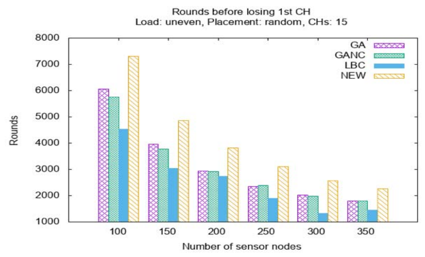
To understand why we did not see substantial improvement in network lifetime, we visually inspect clusterhead load for the case when sensor nodes are generating non-uniform data traffic as shown in Figure 8. All three algorithms (GAC, LBC and NEW) show similar clustering behavior and therefore, only results for the GAC algorithm are shown in Figure 8. The size of the clusterheads and sensor nodes are indicative of their load - the higher the load, the bigger their size. We observe that while the size of all clusterheads are similar, implying that they are equally loaded, clusterheads farther away from the sink deplete their energy faster than those near the sink due to longer distance. Therefore, clusterheads farther from the sink fail quickly which leads to decrease in network lifetime.

7.3. Load-balancing accounting for node distances

As discussed in Section 4, these results inspired us to develop a new cost function that considers all four

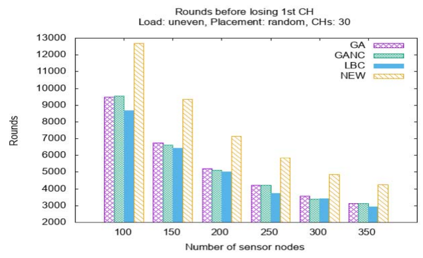


(a) Network lifetime with 10 clusterheads

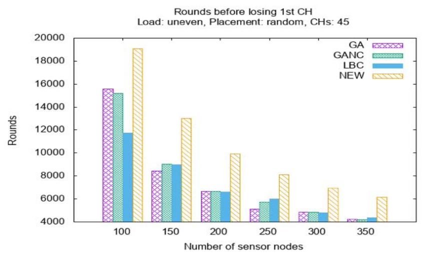


(b) Network lifetime with 15 clusterheads

Figure 10: Network lifetime with new cost function for sensors with unequal load with 10 and 15 clusterheads



(a) Network lifetime with 30 clusterheads



(b) Network lifetime with 45 clusterheads

Figure 11: Network lifetime with new cost function for sensors with unequal load with 30 and 45 clusterheads

metrics - sensor node load, clusterhead load, as well as distance from sensor to clusterhead, and distance from clusterhead to sink. When clustering is performed using the new cost function, we notice, as illustrated in Figure 9, that clusterheads closer to sink are more heavily loaded than those farther away. This depletes the energy among the clusterheads more evenly, increasing network lifetime. As previously pointed out, in order to conduct a fair comparative study, besides using the new cost function in our simulated annealing algorithm, we also apply it to the GAC algorithm, which we call *GANC* (GA New Cost).

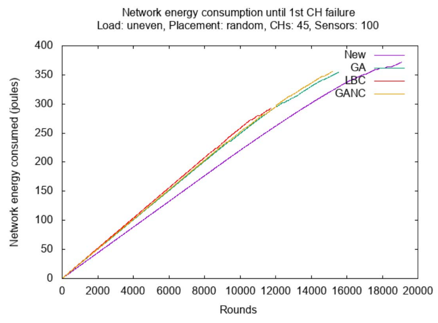
Figures 10 and 11 shows network lifetime when the new cost function is used for the non-uniform sensor load case. Improvement in network lifetime is observed for different number of sensor nodes and clusterheads. Our algorithm increases network lifetime by an average of 30% compared to GAC and an average of 60% compared to LBC. LBC yields the lowest network lifetime as it is not designed to accommodate unequal load. We believe that GANC does not show much improvement over GAC because of the local maxima drawback.

NEW also results in lower energy consumption and at any point in time will have more sensor nodes alive as shown in Figure 12a. Energy consumed during clustering is same for all protocols as the number of messages transmitted is the same. However, energy consumed during data transfer depends on how well the load is balanced. NEW consumes approximately 15% and 20% less energy compared to GAC and LBC, respectively per round. Due to lower energy consumption per round, data can be transferred for more rounds, which leads to increase in network lifetime and sensor nodes will also be alive for longer duration. In summary, besides attaining the primary goal of improved network lifetime, NEW also improves coverage by keeping a higher number of sensor nodes alive as shown in Figure 12b.

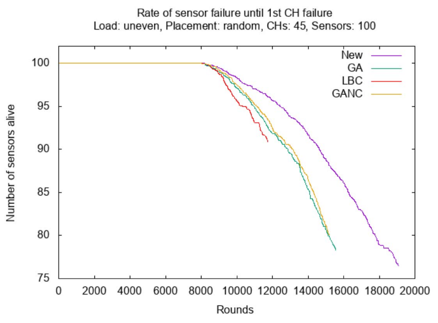
NEW also outperforms LBC, GAC, and GANC for scenarios where sensor nodes generate uniform load as shown in Figure 13[2]. Note that NEW yields better load balancing than LBC which was specifically designed for uniform load applications.

Lastly, we compare the computational cost of the different clustering algorithms by measuring the time it takes for the algorithm to perform clustering once all the information required for clustering is exchanged. As shown in Figure 14a, which plots results for runs with 10 and 45 clusterheads, NEW algorithm is significantly faster when compared to GAC in all cases, and faster than LBC in most of cases. In cases where clusterhead-to-sensor ratio is high, there is higher probability that more clusterheads are within the range of each sensor nodes. Therefore, simulated annealing may take more time to converge, when compared to LBC, as it will have to explore more solutions (i.e., clusterhead to sensor assignment). We observe in Figure 14b that NEW is slower than LBC with 100 sensors but the difference in time shrinks as the number of sensors increases and as a result clusterhead- to-sensor ratio decreases.

[2]Due to space limitations and the fact that the trends in performance are similar in all cases, we only show network lifetime results for 10 and 45 clusterheads in Figure 13.

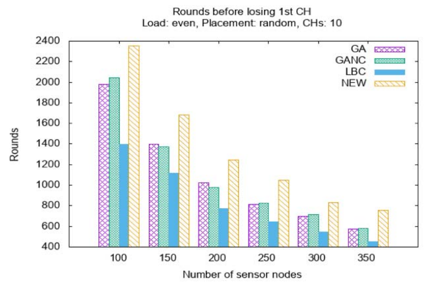


(a) Energy Consumption

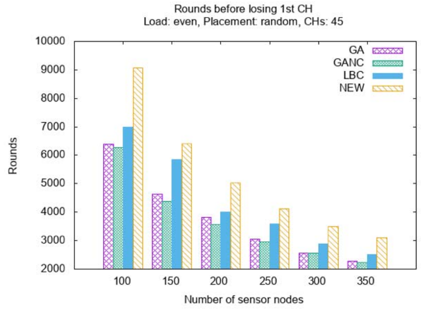


(b) Sensor Failure

Figure 12: Total energy consumption and sensor failure using NEW

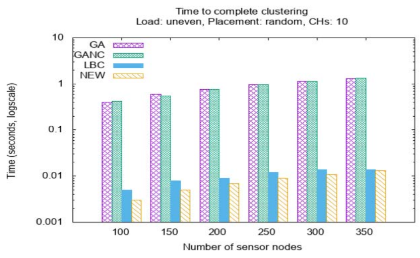


(a) Network lifetime with 10 clusterheads

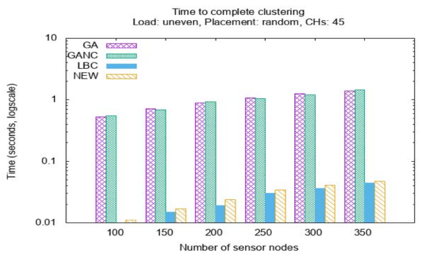


(b) Network lifetime with 45 clusterheads

Figure 13: Network lifetime for uniform sensor load scenarios



(a) Clustering time with 10 clusterheads.



(b) Clustering time with 45 clusterheads

Figure 14: Clustering time non-uniform load scenarios sensors

8. Conclusion and Future Work

This paper introduced a novel clustering algorithm that targets a variety of current and emerging IoT applications such as Smart Environments (e.g., Smart Homes, Smart Buildings, Smart Grids, etc.), Industrial and Agricultural Automation to name a few, where IoT nodes are stationary. The main goal of clustering when applied to IoT applications is to maximize energy efficiency and consequently the IoT's network lifetime. Our experiments show that, when compared to leading state-of-the-art clustering approaches, our simulated annealing based algorithm is able to improve both network lifetime as well as network coverage by keeping more sensor nodes alive for longer periods of time at lower computational cost. This is the case for IoT scenarios where sensor nodes generate uniform and non-uniform data traffic. More specifically, our algorithm increases network lifetime by an average of 30% compared to genetic algorithm based clustering and an average of 60% compared to non-stochastic clustering. It increases energy efficiency by around 10% and its average convergence time is 75 times faster than genetic algorithm based clustering and is approximately as fast as non-stochastic clustering.

As future work, we plan to relax the clustering requirement that sensors are only one hop away from their clusterhead. We also plan to consider sensor failures other than energy depletion as well as the trade-off between re-clustering immediately when sensor failures are detected versus the resulting additional processing and communication cost. Additionally, we will explore other definitions of network lifetime [34].

References

[1] M. Weiser, "The computer for the 21st century.," *Mobile Computing and Communications Review*, vol. 3, no. 3, pp. 3–11, 1999.

[2] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, "Instrumenting the world with wireless sensor networks," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 4, pp. 2033–2036, IEEE, 2001.

[3] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, "Connecting the physical world with pervasive networks," *IEEE pervasive computing*, vol. 1, no. 1, pp. 59–69, 2002.

[4] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[5] "Massive IoT in the city," Nov 2016 (accessed Jan 06, 2018). Available at https://www.ericsson.com/en/mobility-report/massive-iot-in-the-city.

[6] C. P. Low, C. Fang, J. M. Ng, and Y. H. Ang, "Efficient load-balanced clustering algorithms for wireless sensor networks," *Computer Communications*, vol. 31, no. 4, pp. 750–759, 2008.

[7] G. Gupta and M. Younis, "Load-balanced clustering of wireless sensor networks," in *Communications, 2003. ICC'03. IEEE International Conference on*, vol. 3, pp. 1848–1852, IEEE, 2003.

[8] P. Kuila, S. K. Gupta, and P. K. Jana, "A novel evolutionary approach for load balanced clustering problem for wireless sensor networks," *Swarm and Evolutionary Computation*, vol. 12, pp. 48–56, 2013.

[9] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks (LEACH)," in *System sciences, 2000. Proceedings of the 33rd annual Hawaii international conference on*, pp. 10–pp, IEEE, 2000.

[10] A. Manjeshwar and D. P. Agrawal, "TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks.," in *IPDPS*, vol. 1, p. 189, 2001.

[11] S. Lindsey and C. S. Raghavendra, "PEGASIS: power-efficient gathering in sensor information systems," in *Aerospace conference proceedings, 2002. IEEE*, vol. 3, pp. 3–1125, IEEE, 2002.

[12] O. Younis and S. Fahmy, "Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Transactions on mobile computing*, vol. 3, pp. 366–379, 2004.

[13] S. Yi, J. Heo, Y. Cho, and J. Hong, "Peach: Power-efficient and adaptive clustering hierarchy protocol for wireless sensor networks," *Computer communications*, vol. 30, no. 14, pp. 2842–2852, 2007.

[14] J. N. Al-Karaki, R. Ul-Mustafa, and A. E. Kamal, "Data aggregation in wireless sensor networks-exact and approximate algorithms," in *High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on*, pp. 241–245, IEEE, 2004.

[15] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang, "TTDD: Two-tier data dissemination in large-scale wireless sensor networks," *Wireless networks*, vol. 11, pp. 161–175, 2005.

[16] T. Sharma, R. Joshi, and M. Misra, "Gbdd: grid based data dissemination in wireless sensor networks," in *Advanced Computing and Communications, 2008. ADCOM 2008. 16th International Conference on*, pp. 234–240, IEEE, 2008.

[17] A. K. Ghosh, A. K. Bairagi, M. A. Kashem, M. Rezwanul Islam, and A. A. Uddin, "Energy efficient zone division multihop hierarchical clustering algorithm for load balancing in wireless sensor network," *International Journal of Advanced Computer Science and Applications*, vol. 2, pp. 92–97, 2011.

[18] C. Li, M. Ye, G. Chen, and J. Wu, "An energy-efficient unequal clustering mechanism for wireless sensor networks," in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.*, pp. 8–pp, IEEE, 2005.

[19] F. Ishmanov and S. W. Kim, "Distributed clustering algorithm with load balancing in wireless sensor network," in *Computer Science and Information Engineering, 2009 WRI World Congress on*, vol. 1, pp. 19–23, IEEE, 2009.

[20] Y. Liao, H. Qi, and W. Li, "Load-balanced clustering algorithm with distributed self-organization for wireless sensor networks," *IEEE sensors journal*, vol. 13, no. 5, pp. 1498–1506, 2013.

[21] S. Soro and W. B. Heinzelman, "Prolonging the lifetime of wireless sensor networks via unequal clustering," in *19th IEEE International Parallel and Distributed Processing Symposium*, pp. 8–pp, IEEE, 2005.

[22] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[23] M. Rocha and J. Neves, "Preventing premature convergence to local optima in genetic algorithms via random offspring generation," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 127–136, Springer, 1999.

[24] J. Andre, P. Siarry, and T. Dognon, "An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization," *Advances in engineering software*, vol. 32, no. 1, pp. 49–60, 2001.

[25] V. M. Kureichick, V. V. Miagkikh, and A. P. Topchy, "Genetic algorithm for solution of the traveling salesman problem with new features against premature convergence," *Proc. of GA+ SE*, vol. 96, 1996.

[26] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[27] B. F. Al-Dulaimi and H. A. Ali, "Enhanced traveling salesman problem solving by genetic algorithm technique (tspga)," *World Academy of Science, Engineering and Technology*, vol. 38, pp. 296–302, 2008.

[28] Z. H. Ahmed, "Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator," *International Journal of Biometrics & Bioinformatics (IJBB)*, vol. 3, no. 6, p. 96, 2010.

[29] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, "Op-

timization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[30] E. Ferro and F. Potorti, "Bluetooth and wi-fi wireless protocols: a survey and a comparison," *IEEE Wireless Communications*, vol. 12, no. 1, pp. 12–26, 2005.

[31] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and zigbee standards," *Computer communications*, vol. 30, no. 7, pp. 1655–1695, 2007.

[32] S. Sesia, M. Baker, and I. Toufik, *LTE-the UMTS long term evolution: from theory to practice*. John Wiley & Sons, 2011.

[33] L. Alliance, "Lorawan specification," *LoRa Alliance*, 2015.

[34] I. Dietrich and F. Dressler, "On the lifetime of wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 1, p. 5, 2009.