Efficient Clustering Algorithms for Self-Organizing Wireless Sensor Networks

Rajesh Krishnan
BBN Technologies
10 Moulton Street
Cambridge, MA 02138, USA
Phone: +1 617 873 4684
krash@bbn.com

David Starobinski*
Boston University
8 Saint Mary's Street
Boston, MA 02215, USA
Phone: +1 617 353 0202
staro@bu.edu

Abstract

Self-organization of wireless sensor networks, which involves network decomposition into connected clusters, is a challenging task because of the limited bandwidth and energy resources available in these networks. In this paper, we make contributions towards improving the efficiency of self-organization in wireless sensor networks. We first present a novel approach for message-efficient clustering, in which nodes allocate local "growth budgets" to neighbors. We introduce two algorithms that make use of this approach. We analyze the message complexity of these algorithms and provide performance results from simulations. The algorithms produce clusters of bounded size and low diameter, using significantly fewer messages than the earlier, commonly used, Expanding Ring approach. Next, we present a new randomized methodology for designing the timers of cluster initiators. This methodology provides a probabilistic guarantee that initiators will not interfere with each other. We derive an upper bound on the expected time for network decomposition that is logarithmic in the number of nodes in the network. We also present a variant that optimistically allows more concurrency among initiators and significantly reduces the network decomposition time. However, it produces slightly more clusters than the first method. Extensive simulations over different topologies confirm the analytical results and demonstrate that our proposed methodology scales to large networks.

*This work was supported in part by the National Science Foundation under CAREER grant ANI-0132802 and grant ANI-0240333.

1 Introduction

Wireless sensor (and actuator) networks have critical applications in the scientific, medical, commercial, and military domains. Examples include environmental monitoring, smart homes and offices, surveillance, and intelligent transportation systems [1, 17, 21]. As societal reliance on wireless sensor network technology increases, we can expect the size and complexity of individual networks as well as the number of networks to increase dramatically.

For many applications, we envision large static[1] ad hoc networks consisting of hundreds, thousands, or even millions of inexpensive wireless sensor nodes that may be placed either regularly or irregularly. Wireless sensor networks can be exposed to highly dynamic, and hostile environments [21, 26], and therefore, they must be tolerant to the failure and loss of connectivity of individual nodes. Clearly, algorithms for wireless sensor networks must be *distributed* or *decentralized* in order to prevent single points of failure. It is also an essential requirement of wireless sensor networks to be *self-organizing*. By self-organization, we mean the process of autonomous formation of connectivity, addressing, and routing structures. This process typically involves the decomposition of the network into connected *clusters* of bounded size[2].

Self-organization of wireless sensor networks is challenging because of the tight constraints on the *bandwidth* and *energy* [1] resources available in these networks. Sensor nodes are typically battery powered devices. Despite advances in high-speed wireless access, the bandwidth of the shared wireless medium continues to remain a limited resource. Algorithms for self-organization that minimize the number of message transmissions (and receptions) can help to conserve energy. More importantly, in sensor networks, message efficiency permits a more judicious use of the limited bandwidth. Protocols with low message overhead are also preferable in military wireless sensor networks. For example, statistical methods of cryptanalysis require a large number of samples to be successful. Therefore, *low message complexity* is a highly desirable property of self-organization algorithms for wireless sensor networks.

In this paper, we make two major contributions towards improving the efficiency of self-organization in wireless sensor networks. In the first part of the paper, we present a novel approach for message-efficient clustering, in which nodes allocate local *growth budgets* to neighbors. This approach significantly reduces the number of messages exchanged and allows the cluster to grow based on local decisions rather than involving the initiator at

[1] By static we mean limited mobility; the topology can be dynamic due to power conservation modes.

[2] The *size* of a cluster is the number of nodes that it includes.

each round. We introduce two algorithms making use of this approach. The first algorithm called *Rapid* produces clusters of bounded size. This algorithm uses few messages, but has a poor worst-case analytical performance in terms of the cluster size produced. The second algorithm, which is recursive in nature, is called *Persistent* since it persistently tries to produce a cluster of the specified bound if possible. This algorithm significantly improves the worst-case behavior, that is, the smallest cluster size produced.

Through analysis and simulation, we demonstrate that our algorithms are significantly more efficient in terms of message complexity than the earlier, and commonly used, *Expanding Ring* algorithm [22]. For the construction of a single cluster within a network, the proposed algorithms have a worst-case message complexity that is polynomial in the specified cluster size, while the worst-case complexity of Expanding Ring can be very large (i.e. related to the size of the entire network, which need not be polynomial in the specified cluster size).

Furthermore, we analyze the message complexity of the Persistent and Expanding Ring algorithms for sequential decomposition of ring and clique topologies. These topologies represent sparse and dense topologies, respectively. Our analysis shows that the algorithms perform comparably in sparse topologies. However, in dense topologies, the Persistent algorithm uses far fewer messages on average than the Expanding Ring algorithm. Simulation results confirm these findings over a range of random topologies of varying density.

Closely related to the problem of distributed clustering is the problem of deciding how long an unclustered node must wait prior to initiating the process of clustering. The time that each node waits before it initiates clustering is a critical factor that influences the total number of clusters produced, as well as the time taken to cluster the network as a whole.

In the second part of this paper, we present a new randomized methodology for designing the initiator timers. In this methodology, each node maintains an exponentially distributed timer. We use the memoryless property of the exponential distribution to bound the probability that multiple initiators are concurrently active in the same neighborhood. Moreover, the memoryless property allows nodes to start their timers independently without requiring any synchronization.

We present a *conservative* method to determine the rate parameter of the exponential distribution for the timers, which offers a guarantee that initiators in the same neighborhood do not interfere with each other with desired probability $1 - \epsilon$, where $0 < \epsilon < 1$. We show analytically that the upper bound on the expected time for network decomposition using our method is $O(\ln n)$, where n is the number of nodes in the network. In comparison, a

sequential approach, in which only one initiator is allowed to be active in the network at any time, can achieve network decomposition in time which is at least linear in the size of the network.

We also present an *optimistic* variant that allows multiple initiators to be concurrently active in the same neighborhood. This method reduces the total network decomposition time significantly, and more importantly, does not require any knowledge of the network topology. However, it results in a slightly higher number of clusters than the conservative method.

Simulations over different topologies agree with the analytical results and serve to verify the models. Simulations further demonstrate that clustering based on our timer design approaches can scale to large networks and produce a number of clusters that is comparable to the baseline sequential approach.

The rest of this paper is organized as follows. In Section 2, we describe our message-efficient clustering algorithms and provide analytical and simulation results on their performance. We present our analytical methodology for the design of initiator timers in Section 3, and provide simulation results on the scalability and performance of this design. In Section 4, we discuss practical issues in the application of the results within the context of wireless sensor networks. We conclude in Section 5, with a summary of the contributions of this work and directions for future work. Bibliographic references are presented at the end.

2 Message-Efficient Distributed Clustering

2.1 Motivation and Related Work

As mentioned in the Introduction, self-organization of large sensor networks requires decomposition of the network into connected, non-overlapping clusters of bounded size. Bounding the cluster size is a requirement imposed by routing protocol complexity—the control message overhead decreases with cluster size up to the point that the large cluster size causes the intra-cluster control message overhead to dominate [13, 14, 22]. Furthermore, resource constraints or specific network architectures often impose limits on the cluster size.

Our goal is to devise message-efficient distributed algorithms that can form bounded-size clusters. In order to limit the total number of clusters, the achieved cluster sizes should be as close as possible to the specified bound. We note, however, that an arbitrary network cannot always be decomposed into non-overlapping clusters of a desired size. Consider, for

instance, a star topology. Once the hub node is assigned to a cluster, then all unclustered nodes will end up in single node clusters.

Despite the importance of the bound on cluster size, clustering algorithms so far have treated size as a secondary constraint, to be optimized in a post-processing phase [5, 22]. Ramamoorthy *et al.* apply an expanding ring approach [22], in which the cluster depth is progressively relaxed until the desired cluster size is exceeded. Additional messages are used to selectively drop excess nodes in the outermost layer. We show in the sequel of this section that this approach allows temporary violations of the bound and often leads to the transmission of an excessively large number of messages. More recently, Banerjee and Khuller [5] proposed new algorithms for constructing overlapping clusters of bounded size. Their algorithms also use a post-processing phase to enforce the size bound.

Heinzelman et al. [11] proposed an alternative clustering-based approach, called LEACH (Low-Energy Adaptive Clustering Hierarchy). This approach relies on the following two main assumptions (i) there exists a unique base-station with which all the sensors want to communicate; (ii) all the sensors have the ability to communicate directly with the base station. In order to save energy, the LEACH protocol selects a fraction p of the sensors to serve as cluster-heads, where p is a design parameter that must be engineered off-line. Cluster-heads communicate directly with the base station whereas other nodes forward their data through the cluster-heads (typically, the one closest to them). In order to share the energy load, the LEACH protocol implements a load-balancing procedure that allows different nodes to become cluster-heads at different times. Other work related to LEACH includes the PEGASIS protocol [16], in which nodes form a chain to achieve further energy savings.

Our current work differs substantially from the LEACH framework. First, we do not make any assumptions about the number of base stations in the network and/or their specific locations. Second, our algorithms do not rely on direct connectivity to function correctly. This is especially important in sensor network environments where obstacles and power constraints may prevent from two nodes to communicate directly. Third, the LEACH protocol does not attempt to bound the number of nodes per cluster. This number will typically depend on the location, density and transmission power of the nodes and on the particular choice of the cluster-heads. On the other hand, our approach guarantees an upper bound on the number of nodes per cluster irrespective of their placement and density, and does not rely on the transmission power being engineered for achieving any particular topological property. Finally, the main optimization focus of our paper is on message-efficiency rather than energy-efficiency (although these two metrics are clearly correlated). We envision that

our algorithms will be especially useful in those networks where bandwidth is the main performance bottleneck. For instance, this includes sensor networks where nodes are equipped with rechargeable sources of energy, such as miniature solar cells [7].

Subramanian and Katz propose general architectural guidelines for designing self-organizing wireless sensor networks [27]. In particular, their paper emphasizes the importance of implementing a hierarchical infrastructure in order to reduce routing complexity at every node. It also elaborates on the advantages of using multi-hop over single-hop communication. The algorithms that we propose in our paper follow these guidelines.

In the same paper, Subramanian and Katz also suggest an approach for clustering in which an initiator sends a solicitation message, and all the sensors that hear the initiator node respond to it. At the end of each round, the initiator increments its power and the clustering process continues until the number of nodes that respond are between a certain minimum and maximum bound on the cluster size. We note that this approach is very similar to the Expanding Ring approach of Ramamoorthy *et al.* [22], except that the initiator increments its power instead of the hop count. As shown later in our analysis, the worst-case message complexity of this approach can be very high, depending on the total number of sensors in the network rather than on the specified cluster size bound.

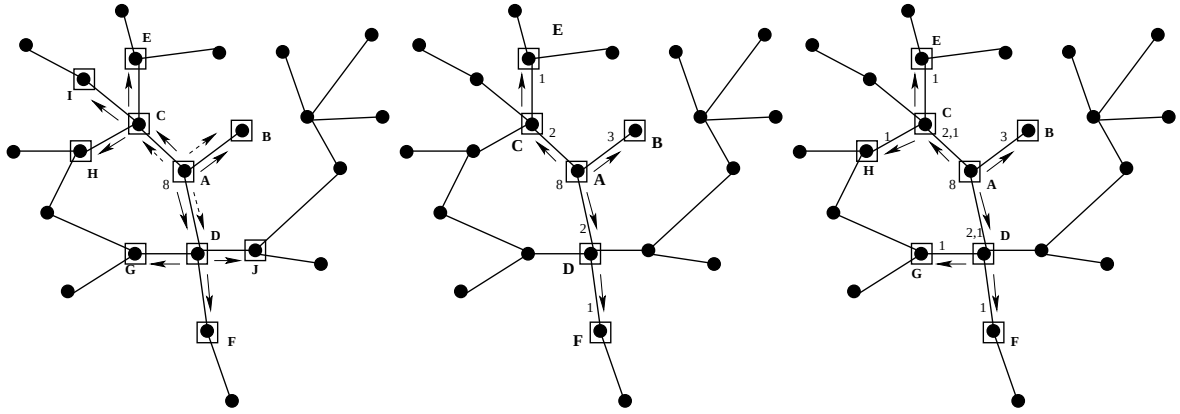
The rest of this section is organized as follows. In Section 2.2, we describe the proposed Rapid and Persistent algorithms and apply them to network decomposition of wireless sensor networks. We present a performance analysis of the algorithms in Section 2.3, followed by simulation results in Section 2.4.

2.2 Algorithms

In this section, we propose two novel algorithms to produce clusters of bounded size—the Rapid and Persistent algorithms. We also describe the earlier Expanding Ring algorithm [22], which we use as a baseline for performance evaluation. Detailed pseudo-codes of the algorithms can be found in Appendix A.

We provide the definitions of some terms used in this section. We consider a network $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges. An edge $(u, v) \in E$ exists iff nodes $u, v \in V$ can communicate directly with each other. Any two nodes that are connected by an edge are neighbors of each other. Nodes in G can send (and receive) messages to (from) their neighbors. Every node $v \in V$ has a unique identity $\mathrm{ID}(v)$. Each node initially knows its own identity and the identities of its immediate neighbors in G.

A *cluster* is a connected component of the original graph G together with a node desig-



(a) Expanding Ring (b) Rapid (c) Persistent

Figure 1: Distributed algorithms for bounded size clustering.

nated as the initiator in the cluster and a spanning tree (a by-product of the clustering) [3]. An *initiator* $v_0 \in V$ is a node that is permitted to initiate the process of clustering by sending messages. The *size* of a cluster is the number of nodes belonging to it. The bound on the maximum cluster size is B, where B is smaller than n. The *depth* of the cluster is the height of the associated spanning tree.

2.2.1 The Expanding Ring algorithm

Ramamoorthy *et al.* present an algorithm for bounded size clustering based on an expanding ring search [22]. The Expanding Ring algorithm proceeds in rounds starting with a maximum hop limit of 1. In each round of the algorithm, the initiator sends messages with a scope set to one hop more than the previous round. At the end of each round, the initiator knows the total number and IDs of nodes that are added in the last layer, that is, nodes at a distance equal to the maximum hops for that round. Eventually, after a few rounds, the cluster size bound will be exceeded. For example, in Figure 1(a), a bound of 8 is exceeded at 2 hops resulting in a cluster containing 10 nodes.

The initiator then sends another message containing a list of (arbitrarily chosen) nodes in the last layer to be dropped from the cluster in order to achieve the bound. This message is then flooded within the cluster.

2.2.2 The Rapid algorithm

In the Rapid algorithm, the initiator is assigned a budget of B, of which it accounts for itself and *evenly* distributes $B-1$ among its neighbors by sending a message to each one of them. An arbitrary subset is chosen if there are more neighbors than the budget. The neighbors that receive the message account for themselves and distribute the remaining among all their neighbors except the parent. The messages propagate until they reach a stage where the budget is exhausted. This algorithm is illustrated in Figure 1(b) in which a bound of 8 results in forming an actual cluster of size 6. Note that the initiator (node A) allocates a budget of 3 to a neighbor (node B) which is a leaf and can add only one node to the cluster.

Each node that receives a message sends an acknowledgment to its parent when either the budget is exhausted or it has received acknowledgments from all its children. The algorithm terminates when the initiator receives acknowledgments from all neighbors to which it sent a message. The acknowledgments can be used to convey the size and depth of the subtree[3] and the maximum hop count reached. The initiator can use this to determine the size and depth of the cluster. Furthermore, by including hop count information in the messages and acknowledgments, a modified algorithm can limit the depth of the cluster, but this can restrict the maximum cluster size achieved.

2.2.3 The Persistent algorithm

The Rapid algorithm presented in the previous section has a low message complexity (see Section 2.3.1). However, in the worst case it can produce clusters that are very small when compared to the desired bound (see Section 2.3.2). Decomposing a network into a large number of small clusters can lead to inefficient routing.

The Persistent algorithm is a recursive elaboration of the Rapid algorithm that significantly improves the worst-case behavior in terms of the cluster size produced.

Using the Persistent algorithm, the initiator is provided a budget of B, of which it accounts for itself and evenly distributes $B-1$ among its neighbors via messages. An arbitrary subset is chosen if there are more neighbors than the budget. The neighbors that receive the message, account for themselves and distribute the remaining among all their neighbors except the parent. The messages propagate until they reach a stage where the budget is exhausted.

In this algorithm, each node that receives a message does not send an acknowledgment

[3]The subtree is the part of the spanning tree that is accounted for by the budget allocated to this node.

to its parent immediately on receiving acknowledgments from all its children. It computes the size of the subtree and compares it to the budget allocated to it. It distributes the shortfall among its neighbors that either were not explored previously or met all previously allocated budgets. When either the budget is met or when further growth is not possible, it returns an acknowledgment to its parent. The algorithm terminates when the initiator determines that the bound has been met or when no further growth is possible (for example, there are not enough nodes adjacent to the cluster that are free to join this cluster).

This algorithm is illustrated in Figure 1(c). In this example, node A first allocates a budget of 3 to node B. Node B returns an acknowledgment to node A, with a budget consumption of 1. Node A then equally allocates the surplus budget of 2 among its other children (nodes C and D). The sequence $2, 1$ on the link from A to D indicates that A initially allocates a budget of 2 to D and then reallocates an additional budget of 1.

In the presence of a single initiator, the Persistent algorithm produces a cluster of the specified bound. When a cluster of the specified size cannot be constructed (because not enough unclustered nodes are available), it attempts to build the largest cluster possible.

2.2.4 Network decomposition

The Rapid, Persistent, and Expanding Ring algorithms that we described can each produce a single cluster of bounded size. We can apply any one of them to perform network decomposition into clusters of bounded size using the simple procedure described below.

Each node that comes up waits for clustering messages from its neighbors. When a timeout (randomly chosen within a configured interval) occurs, the node becomes an initiator itself and invokes one of the three clustering algorithms. The process continues until all the nodes in the network are clustered. In Section 3, we introduce a randomized methodology, based on the exponential distribution, to set the initiator timers.

2.3 Analysis

In the context of wireless sensor networks with energy-limited and bandwidth-limited devices, message complexity is one of the most important metrics to evaluate the efficiency of an algorithm. In this section, we analyze the message complexity of the three algorithms when applied to both the construction of a single cluster and network decomposition. We are interested in large networks, in which the upper bound on the cluster size $B << n$, where n is the network size.

For the purpose of analysis, we assume that during the clustering process: (i) the graph is connected, (ii) there are no changes in the network topology, and (iii) no messages are lost during clustering. In Section 4, we discuss how to deal with practical implementation issues such as node failures and loss of messages over unreliable links.

2.3.1 Message complexity of the algorithms to build a single cluster

In this section, we analyze the message complexity of the three algorithms in the presence of a single initiator in the network. Note that if acknowledgements are counted as separate messages, then the worst-case number of messages produced by each algorithm should be multiplied by two.

Lemma 1 *The Rapid algorithm uses at most B messages, where B is the specified upper bound on the cluster size.*

Proof: Each message sent by a node to one of its neighbors implies that the total budget is reduced by at least 1. Since the initial budget is B, there will be no more than B messages (or more precisely $B-1$ messages). ■

Lemma 2 *The Persistent algorithm uses at most $2B^2$ messages, where B is the specified upper bound on the cluster size.*

Proof: As a by-product of the clustering, the Persistent algorithm generates a spanning tree rooted at the initiator. The parent of each node in the cluster is the node from which it first receives a budget.

Each node in the cluster has a budget of at most B. Therefore, the budget can be reallocated at most B times by each node in the cluster. So the total number of messages on each edge of the spanning tree is B in the worst case. Therefore, the total number of messages sent on all spanning tree edges is B^2.

In addition, each node in the cluster could have sent messages to at most B neighbors that have not been included in the spanning tree as a child of the given node. The total number of messages sent on non-spanning tree edges is B^2. Thus, the Persistent algorithm uses at most $2B^2$ messages.

■

We note that the worst-case message complexity of $O(B^2)$ is optimal for any algorithm of this class (e.g., see [9]).

Lemma 3 *The Expanding Ring algorithm uses $O(n)$ messages, where n is the number of nodes in the network.*

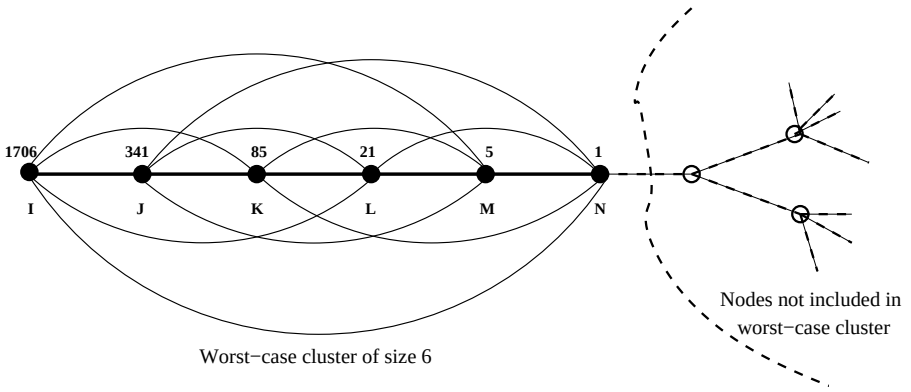


Figure 2: Example of a network in which a unique initiator using the Rapid algorithm produces the worst-case (smallest) cluster size. Given a budget of 1706, a cluster of size 6 is produced. Note that this is smaller than $e^{W(\ln 1706)} + 2 = 6.76$. The number near each node indicates the budget allocated to the node.

Proof: In any round in which the cluster size is less than B, the total number of messages exchanged is polynomial in B. In the worst case, all nodes in the network may be sent messages in the last round. Therefore, the worst-case message complexity of expanding ring search is $O(n)$ since $B << n$. ■

From the lemmas above, we see that the proposed Rapid and Persistent algorithms both have message complexity that is polynomial in B. The Expanding Ring algorithm has a worst-case message complexity that is not related to B and can be very large (e.g. $n = 2^B$).

2.3.2 Cluster size with a single initiator

In this section, we derive tight bounds on the worst-case cluster size produced by the Rapid algorithm with a single initiator present. The analysis shows that the Rapid algorithm may produce very small clusters in the worst case. On the other hand, it is fairly easy to show that the Persistent and Expanding Ring algorithms achieve the bound B in the presence of a unique initiator [15]. The proof of the following theorem is given in Appendix B.

Theorem 1 *Denote by $T(B)$ the worst-case (smallest) cluster size produced by the Rapid algorithm, with a unique initiator. Then $e^{W(\ln B)} \leq T(B) \leq e^{W(\ln B)} + 2$, where W is the Lambert-W function*[4].

In Figure 2, we provide an example construction that results in the worst-case (smallest) cluster size. The construction is a clique that includes a set of fast links that connect the

[4]The Lambert-W function, the inverse of $f(W) = We^W$, has the expansion $W(x) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} x^n$.

nodes of the cluster in a linear chain. The node at the end of the chain connects the clique to other nodes in the network. The other links have much higher delay, and the neighbor is clustered (along the fast path) before messages are delivered over the slow links. In such a network, the cluster grows by 1 at each stage, and the maximum possible fraction of the budget is wasted at each stage.

We note that the value of $e^{W(\ln B)}$ grows very slowly with B. For example, as B is increased from 1 to $60,393,152$, it grows only from 1 to 9. However, this worst-case analysis is very pessimistic. Our simulation results in Section 2.4 shows that the Rapid algorithm performs much better on average.

2.3.3 Message complexity of network decomposition

In this section, we analyze the message complexity of network decomposition of two regular topologies—clique and ring—that represent extremes of dense and sparse topologies respectively. The analysis provides insight into the relative message complexity of the algorithms both in sparsely- and densely-connected graphs. The results of our analysis are validated by simulations described later.

The analysis in the case of a general graph is difficult. The actual set of initiators and the specific sequence in which they become active will affect the number of clusters, the size of individual clusters, and the total number of messages used. Therefore, in order to simplify the message complexity analysis (and to isolate the effects of implementation details), we assume that only one initiator is active at any given time. In the case of the ring, we also assume that an unclustered neighbor of the last cluster is chosen as the next initiator. Furthermore, the regularity of the ring and clique topologies makes the choice of initiators less critical to the performance analysis. Our simulation-based comparisons (in Section 2.4) extend to other general topologies. In Section 3, we extend the analysis to cases in which multiple initiators are concurrently active.

In the ring and clique topologies, both the Persistent and Expanding Ring algorithms produce the same number of clusters. We next derive the worst-case and average-case message complexity of the algorithms in producing these clusters, where the average-case complexity is based on averaging over different runs on the same topology.

The analysis of the ring is fairly simple (see [15] for details). The Persistent and Expanding Ring algorithms have both a message-complexity of $O(n)$ (either worst-case or average) and thus perform comparably.

We now show that the Persistent algorithm has a much lower average message complexity

than Expanding Ring for the case of a clique.

Lemma 4 *Sequential network decomposition of a clique of n nodes using the Expanding Ring algorithm has a message complexity of at least n^2/B messages, where B is the specified upper bound on the cluster size.*

Proof: On the clique, the initiator in the Expanding Ring algorithm sends n messages to form a cluster of size B. After each cluster is formed, an unclustered neighbor is chosen as the next initiator. The process is repeated until the clique is decomposed into n/B clusters. Thus, the message complexity is at least n^2/B, since additional messages are required to drop excess nodes in each round. Note that the average case and the worst case are identical. ■

Lemma 5 *Sequential network decomposition of a clique of n nodes using the Persistent algorithm has an average message complexity of at most $n(\log n/B + 2)$ messages, where B is the specified upper bound on the cluster size.*

Proof: On the clique, the first initiator using the Persistent algorithm requires no more than B messages to create a cluster of size B. After each cluster is formed, an unclustered neighbor is chosen as the next initiator.

The last cluster to be built will require n messages in the worst case. This corresponds to the case when all $n - B$ nodes in the cluster are tried before any of the available B nodes are tried.

Let us consider the intermediate case when $i - 1$ clusters have been built and the i^{th} cluster is being constructed. For picking the very last node in the i^{th} cluster, we can choose from $n - iB + 1$ available nodes out of a total of n nodes. Therefore, the expected number of tries to pick the last node in this i^{th} cluster is $n/(n - iB + 1)$.

Since a total of B nodes must be picked for this cluster, the expected number of messages to build the i^{th} cluster is upper bounded by $nB/(n-iB+1)$. Thus, the expected total number of messages required for decomposing the clique topology is at most

$$\sum_{i=1}^{\frac{n}{B}-1} \frac{nB}{n - iB} + n. \tag{1}$$

For constant B, we obtain

$$\sum_{i=1}^{\frac{n}{B}-1} \frac{nB}{n - iB} + n = n\left(\sum_{i=1}^{\frac{n}{B}-1} \frac{1}{i} + 1\right) \leq n\left(\log n/B + 2\right). \tag{2}$$

Thus, the Expanding Ring algorithm has an average message complexity of $O(n \log n)$ for constant B. ■

The above result is representative of the expected behavior of the Persistent algorithm. However, in a worst-case scenario, the message complexity of the Persistent algorithm can be $O(n^2)$, as shown by the following lemma.

Lemma 6 *Sequential network decomposition of a clique of n nodes using the Persistent algorithm has a worst-case message complexity of at most* $\frac{n^2}{2B}$ *messages, where B is the specified upper bound on the cluster size.*

Proof: On the clique, the first initiator using the Persistent algorithm requires at most B messages to create a cluster of size B. After each cluster is formed, an unclustered neighbor is chosen as the next initiator. In the worst case, the subsequent initiators will try sending messages to all the clustered nodes before they find the unclustered nodes. Thus the total number of messages required is $B + 2B + \cdots + \frac{n}{B}B$ messages. Thus, the Expanding Ring algorithm has a worst-case message complexity less than $\frac{n^2}{2B}$ or $O(n^2)$ for constant B. ■

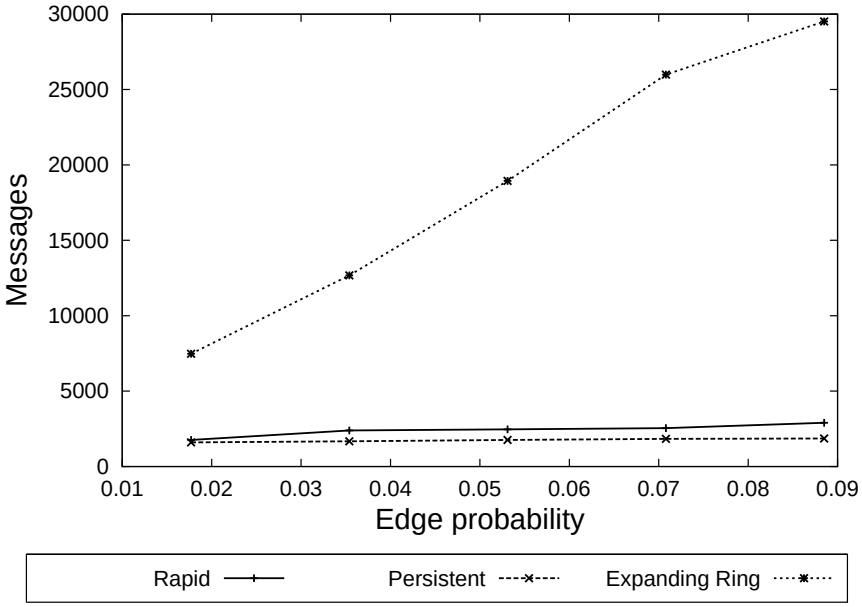
2.4 Simulation results

In this section, we compare the performance of the Rapid, Persistent, and the Expanding Ring algorithms. We implemented our simulations including the algorithms and the topology generators using the Java programming language.

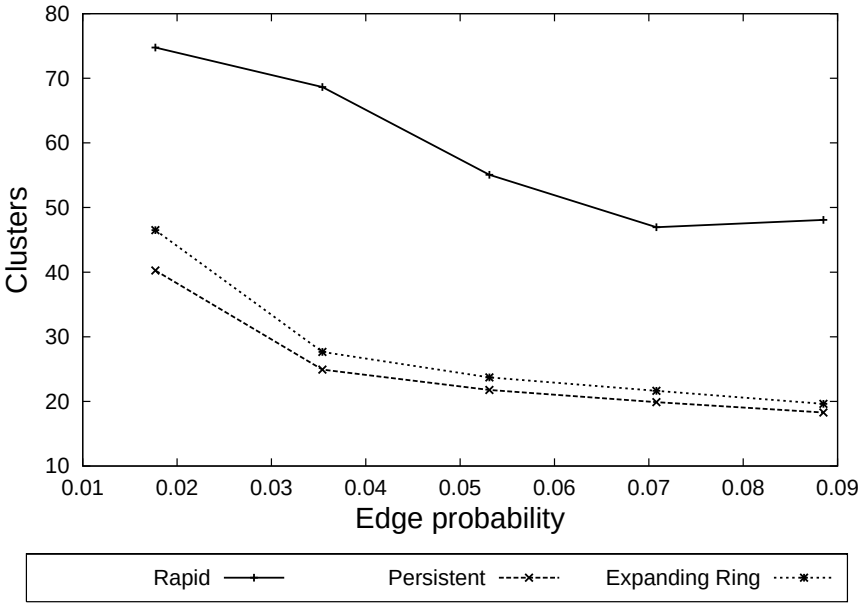
2.4.1 Random Topologies

We generated random graphs based on the Erdos-Renyi model [8]. In this model, a graph $G_{n,\alpha}$ consists of n nodes, and each possible edge between two vertices is present with independent edge probability α and absent with probability $1 - \alpha$.

For our simulations, we chose the edge probability α from the set $\{0.0177, 0.0354, 0.0531, 0.0708, 0.0885\}$. We chose the lowest value of 0.0177 to be higher than the empirically determined value of the edge probability below which the graph is almost surely not connected. The delay for all links was set to unit time. For the plots, the cluster size bound was chosen to be 32, but we observed similar results over a wide range of cluster size bounds. We compare the three algorithms using the number of clusters produced, the average cluster size and the number of messages exchanged. Each data point is computed by averaging the results from 50 runs each over 50 different 400-node graphs for each given α.



(a) Number of messages



(b) Number of clusters

Figure 3: Network decomposition performance with a cluster size bound of 32 and different α. Each point is averaged over 50 runs each over 50 different 400-node random graphs.

In practice, our algorithms can be applied with multiple initiators being active concurrently. However, in our simulations in this section, we allow only one initiator to be active at any given time. We do this in order to evaluate the message complexity of the algorithms independently of implementation-specific parameters such as timeouts. In Section 3, we introduce a methodology to set timers for clustering with concurrent initiators, and present simulation results using concurrent initiators.

Figure 3 shows network decomposition performance with a size bound of 32 for different graph densities. As expected, the simple Rapid algorithm produces a large number of clusters when compared to the other two algorithms. As the density increases, the number of messages required for network decomposition by both the Rapid and Persistent algorithms increases marginally; with the Expanding Ring algorithm the number of messages required increases rapidly as the density increases. At the same time, the Persistent algorithm produces roughly the same number of clusters as the Expanding Ring algorithm.

2.4.2 Sensor Networks Topologies

We consider a model for sensor networks in which n nodes are distributed in a square area of length l units, such that a single node is deployed at a random location within every square area of length $\frac{l}{\sqrt{n}}$ units. The average node density is $\frac{n}{l^2}$ nodes per unit area. In our model, two nodes have a link if and only if they are within a distance d_r units of each other. Thus, a node has about $(\pi d_r^2 n/l^2) - 1$ neighbors, ignoring the edges of the sensor field. This leads to an edge probability of $((\pi d_r^2 n/l^2) - 1)/(n-1)$, which is the probability that two nodes chosen at random in the network are connected by a link. In particular, for large n, the edge probability is approximately $\pi d_r^2/l^2$.

For our simulations presented in Table 1, we consider large networks of 15680 nodes that provide sufficient node density to guarantee connectivity in all cases considered. Each node has a radius of communication $d_r = 1$ unit. The nodes are spread over a square area whose sides range in length from 12 units to 56 units in order to cover the range of edge probabilities from 0.001 to 0.02. For each entry in the table, our results are averaged over 100 runs, namely 10 iterations with different seeds are run over 10 different random node placements of the nodes. The edge probabilities in the table have been obtained empirically (and are in close agreement with the analysis).

The results obtained for these sensor network topologies are very similar to those previously obtained for random graphs. The Persistent algorithm produces slightly less clusters than Expanding Ring, while the Rapid algorithm generates significantly more clusters than

Field Length	Edge Probability	Node Density	Clustering Algorithm	Number of Messages	Number of Clusters
56.0	0.00092	5.0	Expanding Ring	5.471×10^5	1375.16
			Persistent	5.497×10^4	1160.37
			Rapid	5.350×10^4	1868.45
40.0	0.00186	9.8	Expanding Ring	5.907×10^5	1218.26
			Persistent	5.106×10^4	1103.83
			Rapid	6.986×10^4	2330.12
25.0	0.00479	25.09	Expanding Ring	7.418×10^5	1077.81
			Persistent	4.938×10^4	1038.68
			Rapid	1.164×10^5	3883.00
18.0	0.00918	48.39	Expanding Ring	9.850×10^5	1033.04
			Persistent	4.816×10^4	1013.90
			Rapid	1.253×10^5	4179.74
15.0	0.01312	69.69	Expanding Ring	1.210×10^6	1017.60
			Persistent	4.780×10^4	1005.80
			Rapid	1.286×10^5	4287.63
12.0	0.02023	108.89	Expanding Ring	1.624×10^6	1004.90
			Persistent	4.781×10^4	996.95
			Rapid	1.370×10^5	4568.73

Table 1: Comparison of the performance of the three clustering algorithm for a sensor network of 15680 nodes with varying node densities. The cluster bound size is $B = 16$.

the two other algorithms. As predicted by our analysis, the number of messages required by Persistent is essentially independent of the network density. On the other hand, Expanding Ring uses more messages as the network gets denser, generating from 10 up to 30 times more messages than Persistent.

3 Timer Design for Distributed Clustering

3.1 Motivation

In Section 2, we considered the problem of message-efficient algorithms for distributed decomposition of a large network into clusters of bounded size. By distributed, we mean that unclustered nodes in the network must independently (using purely local decisions) become initiators and start clustering. In order for the approach to scale to very large networks, we require that the network must be decomposed rapidly (expected time to complete decom-

position should ideally be independent of the size of the network), and at the same time, for a given upper bound B on the cluster size, produce as few clusters as possible (to allow hierarchical routing to scale). We address this issue in this section.

The key underlying problem is the following. If the instants when initiators become active are set too far apart, the total time to complete the network decomposition will be too large. On the other hand, when several initiators are concurrently active, some initiators will produce clusters of size smaller than a specified bound B, since nodes in their neighborhood may be clustered by other initiators. Therefore, it is desirable that initiators be spaced apart both in time and space (e.g. distance in network hops), so that on average they allow initiators time and room to grow clusters of size B when possible.

Therefore, in this section, we address the following questions, which, to our knowledge, have not been addressed previously in the literature:

1. How can we set the initiator timeout values?

2. How can we determine the expected time to complete the network decomposition?

3. How do the expected time and number of clusters produced compare for different design choices?

One approach is to allow only one initiator to be active in the entire network at any given time. While this sequential approach will guarantee that two initiators will never compete, the total time required will be at least linear in the size of the network. This approach does not scale well for large networks.

Furthermore, the sequential approach is less robust, because it will require coordination among all the nodes in the network. Distributed consensus is required in order to ensure that there is only one initiator active at any time. Due to a result by Fischer, Lynch, and Paterson, consensus cannot be solved deterministically in an asynchronous system that is subject to even a single crash failure [10]. This impossibility results from the inherent difficulty of determining whether a process has actually crashed or is only very slow.

We present a new methodology to solve the problem. Our goal is to design the initiator timeouts in a way that bounds the probability that multiple initiators are concurrently active in the same neighborhood, while at the same time, ensures that all nodes in the network are clustered within a short period of time. We present a solution that achieves full network decomposition within expected time logarithmic in the network size. In addition, we propose an optimistic variant that reduces the total decomposition time, but results in a slightly

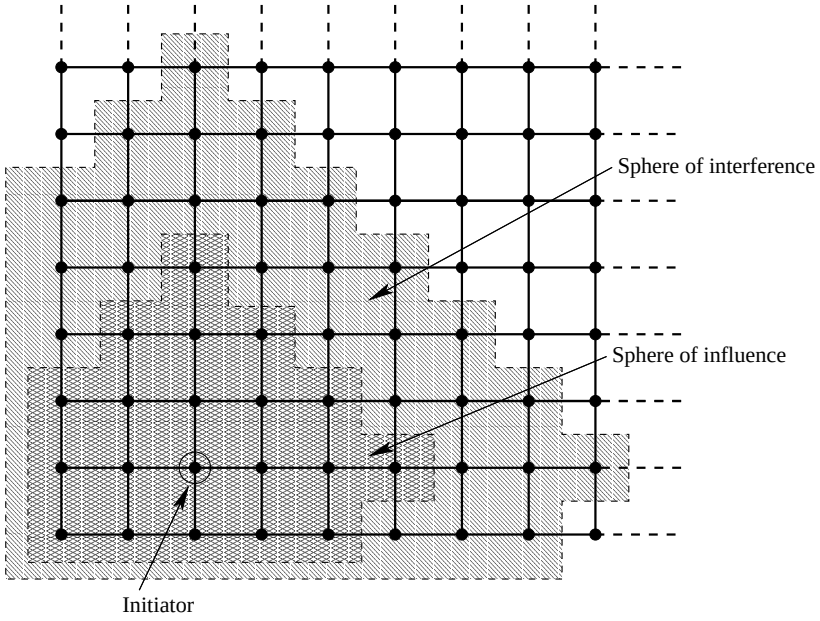


Figure 4: Spheres of influence and interference in mesh topology with cluster size bound= 4.

higher number of clusters. This latter method allows more initiators to be concurrently active. We present an analysis on our solutions, followed by simulation results over various topologies.

3.2 Notations and definitions

We set up the notation and definitions that are used in the remainder of this section.

Definition 1 *The clustering time t_c is the time taken in the worst case by an initiator node to form a cluster of size at most B, using a given algorithm $\mathcal{A}$.*

We note that the time to produce a non-trivial cluster not exceeding a size bound B depends on B and the specific algorithm $\mathcal{A}$ chosen.

Next, we define two sets based on the local topology of the network that determine the time to construct a cluster and the potential for initiators to contend for clustering the same nodes in a network.

Definition 2 *A node x is in the* sphere of influence *(Ψ_i) of an initiator i, if a cluster constructed by i can contain x for a given maximum cluster size B.*

Definition 3 *A node j is in the* sphere of interference *(Φ_i) of an initiator i, if there exists a node x in the network within the sphere of influence of both i and j.*

In Figure 4, we illustrate the sphere of influence and interference for a rectilinear mesh topology with a cluster size bound of 4. The sphere of influence is the search space of an initiator for clustering, and can therefore affect the clustering time for some algorithms. Two concurrent initiators within the sphere of interference can potentially compete for the same node. The sizes of the spheres of influence and interference can be used to design the initiator timers.

Finally, we define *order statistics*, which offer a formal way to describe and study a population of random variables such as initiator timers [2].

Definition 4 *Given a sample of s random variables, $X_1, X_2, \cdots, X_s$, reorder them so that $X_{1:s} < X_{2:s} < \cdots < X_{s:s}$. Then the k^{th} order statistic of this sample of random variables is given by $X_{k:s}$.*

For notational convenience, we define $X_{0:s} = 0$.

3.3 Timer design approach

In this section, we present a conservative design approach to set the initiator timeouts. By conservative, we mean that a probabilistic guarantee is provided that multiple initiators do not interfere with each other.

In our approach, each node has a initiator timer that fires according to an exponential random distribution, $f(t) = \lambda e^{-\lambda t}$, $t > 0$. The node timers are independent and identically distributed (i.i.d). When the timer fires, the node wakes up and becomes an initiator. The randomness ensures that all nodes do not attempt to initiate clustering at the same time. Due to the memoryless property of the exponential distribution, it is not necessary for all timers to be started at the same time. Furthermore, we can use the memoryless property of the exponential distribution to derive bounds on the probability that initiators are not concurrently active within the the same neighborhood.

We try to bound the probability that another initiator is concurrently active within the sphere of interference of an active initiator. Our problem therefore, is to set the rate parameter λ for the initiator timers in order to ensure that the network is decomposed rapidly and at the same time, contention between concurrent initiators in the same neighborhood is minimized. We require with probability $1 - \epsilon$, $0 < \epsilon < 1$ that no other initiator within the sphere of interference Φ_i was active concurrently with a given initiator i. The design

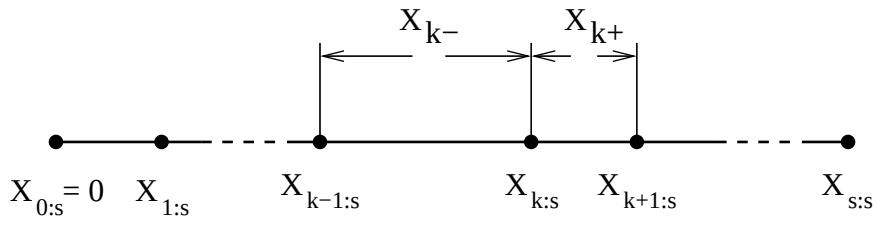


Figure 5: Sequence of firing of initiator timers within the sphere of interference of size $|\Phi| = s$. X_{k-} and X_{k+} are intervals generated by three consecutive timer firings.

parameter ϵ determines the aggressiveness of the setting. The higher the value of ϵ, the higher the likelihood that initiators can interfere with each other, and faster the decomposition is completed. Since the time required to form a cluster is bounded by t_c, we are interested in the time intervals of length t_c that immediately precede and follow the time instant when an initiator fires.

Suppose the initiator timer of node i fires at time t_i, and let i be the k^{th} timer to fire within Φ_i, where $k = 1, 2, \cdots s$, $s = |\Phi_i|$. The timers can be described using order statistics (see Definition 4). In particular, we are interested in order statistics of the exponential distribution for our design. Let us denote by X_{k-} (X_{k+} respectively) the interval between t_i of initiator i and the firing time of the previous (next) initiator in Φ_i. These are illustrated in Figure 5. Due to the memoryless property of the exponential distribution, $X_{k-} = X_{k:s} - X_{k-1:s}$ and $X_{k+} = X_{k+1:s} - X_{k:s}$ are also exponentially distributed:

$$Pr\{X_{k-} > t\} = e^{-(s-k+1)\lambda t},\ 1 < k \leq s \tag{3}$$

$$Pr\{X_{k+} > t\} = e^{-(s-k)\lambda t},\ 1 \leq k < s \tag{4}$$

Equation 3 computes the probability that the current (k^{th}) node timer will fire after an interval t has elapsed since the previous ($k - 1^{th}$) initiator fired. Equation 4 computes the probability that the next ($k + 1^{th}$) initiator will fire after an interval t from the time the current (k^{th}) initiator fired.

Since the time required to form a cluster is bounded by t_c, we require with probability $1 - \epsilon$ that no other initiator within Φ_i was active concurrently with i. In other words,

$$Pr\{X_{k-} > t_c \cap X_{k+} > t_c\} \geq 1 - \epsilon,\ 0 < \epsilon < 1 \tag{5}$$

The first and last initiators each have a strictly lower probability of experiencing inter-

ference than the second and penultimate initiators respectively. The first initiator to fire has no previous initiator to contend with. However other initiators may start before the first initiator completes clustering. The last initiator can contend only with initiators that fired previously. Other initiators may experience interference both from initiators that fired before them and after them. Therefore, it is sufficient to consider the range $1 < k < s$ covered by Equation 5. Since X_{k-} and X_{k+} are independent, we get

$$Pr\{X_{k-} > t_c \cap X_{k+} > t_c\} = Pr\{X_{k-} > t_c\} \cdot Pr\{X_{k+} > t_c\} \tag{6}$$

From Equations 3, 4, 5, and 6, we get,

$$\begin{aligned} 1 - \epsilon &\leq e^{-(s-k+1)\lambda t_c} \cdot e^{-(s-k)\lambda t_c} && (7) \\ &= e^{-(2s-2k+1)\lambda t_c} && (8) \end{aligned}$$

Taking logarithms on both sides, we get

$$\ln(1-\epsilon) \leq -(2s - 2k + 1)\lambda t_c \tag{9}$$

or,

$$\lambda \leq \frac{1}{(2s-2k+1)t_c} \ln \frac{1}{(1-\epsilon)} \tag{10}$$

The right hand side in Equation 10 is smallest for $k = 2$, therefore,

$$\lambda \leq \frac{1}{(2s-3)t_c} \ln \frac{1}{(1-\epsilon)} \tag{11}$$

From Equation 11, we see that our design of the rate parameter depends only on the sphere of interference. The expected value of the k^{th} timer is given by,

$$E[X_{k:s}] = \sum_{i=1}^{k} \frac{1}{(s-i+1)\lambda},\ 1 \leq k \leq s \tag{12}$$

The expected termination time using this design approach is bounded by the firing time for the last (isolated) node in the entire network, and this is given by:

$$E[X_{n:n}] = \sum_{i=1}^{n} \frac{1}{(n-i+1)\lambda} \tag{13}$$

$$< \frac{1}{\lambda}\left(\ln n + \gamma + \frac{1}{2n}\right) \quad (14)$$

$$\asymp \frac{\ln n + \gamma}{\lambda} \quad (15)$$

where $\gamma \approx 0.577215664901$ is the Euler-Mascheroni constant. In Equation 14, we use a result from [29], and the $\asymp$ symbol in Equation 15 denotes asymptotic equality as $n \to \infty$. We note that our probabilistic guarantee that initiators do not interfere is robust and does not require to start all the timers at the same time. However, the bound on the decomposition time of the entire network is valid only if the timers of all the nodes in the network have been started at the same time.

Thus our approach has an upper bound on the expected termination time that is logarithmic in the network size. Therefore, it is more scalable than the sequential clustering approach which has a termination time that is at least linear in the size of the network. Our design allows for a natural trade-off by varying the parameter ϵ. By varying ϵ from a setting close to 0 to a more aggressive setting close to 1, we can achieve faster termination times at the possible cost of more clusters produced. The approach is expected to work well in networks in which the spheres of influence and interference of initiator nodes are small when compared to the size of the network.

3.3.1 Clustering time

The expected termination time depends on the clustering time t_c. In this section, we provide the clustering time for the Rapid, Persistent, and Expanding Ring algorithms that were described in Section 2.

If the link delays are bounded from above by one time unit, then it is easy to see that the worst-case clustering time t_c is $2(B-1)$ units for the Rapid algorithm and $B(B-1)+2(B-1)$ units for the Expanding Ring algorithm. For these algorithms, t_c is independent of the sphere

Table 2: Worst-case clustering time of different algorithms for cluster size bound B and sphere of influence of size $|\Psi|$. Time is measured in units of maximum link delay.

Algorithm	Clustering Time
Rapid	$2(B-1)$
Persistent	$2B\|\Psi\|$
Expanding Ring	$B(B-1)+2(B-1)$

of influence.

The analysis for the Persistent algorithm follows the same lines as the message complexity analysis (for the single initiator case) presented in Section 2. The budget can be reallocated at most B times by each node on the spanning tree. So the total number of messages on each link of spanning tree including the responses is $2B$ in the worst case. Each node in the spanning tree could have sent messages to at most $|\Psi| - 1$ neighbors in the sphere of influence Ψ of the initiator that have not been included in the spanning tree as a child of the given node. The number of messages wasted is $2B(|\Psi| - 1)$. The total number of messages is at most $2B|\Psi|$. At worst, these messages are sent sequentially, and then the total clustering time is at most $2B|\Psi|$ time units.

The general formulas for the clustering time for different algorithms are summarized in Table 2. Tighter bounds for t_c can be found in special cases (for example, in a ring, $t_c = O(B)$ for the Persistent algorithm).

3.3.2 Spheres of influence and interference

The expected termination time also depends on the size of the sphere of interference $s = |\Phi|$ in the network. In Table 3, we present the size of the spheres of influence and interference for a number of topologies of interest. As before, we consider large networks with $B << n$.

In addition to the ring, mesh, and clique topologies, we also consider a sensor network model. For this model, we use an irregular random topology in which nodes are uniformly distributed over an area, and the maximum number of nodes deployed per unit area is bounded by ρ. The network topology is determined by a unit disk graph, that is, nodes can communicate with all other nodes that are within a specified distance d_r. We note that the mesh topology as well as the sensor network topology used in the simulations of Section 2.4.2 are special cases of this model.

Table 3: Sizes of the spheres of influence ($|\Psi|$) and interference ($|\Phi|$) for various topologies for cluster size bound B.

Topology	$\|\Psi\|$	$\|\Phi\|$
Ring	$1 + 2(B-1)$	$1 + 4(B-1)$
Rectilinear Mesh	$1 + 2B(B-1)$	$1 + 4(B-1)(2B-1)$
Clique	n	n
Sensor Net Model	$\rho\pi(B-1)^2 d_r^2$	$4\rho\pi(B-1)^2 d_r^2$

3.4 An optimistic approach to speed up decomposition

In practice, multiple initiators can be concurrently active within the sphere of interference without significantly impacting the average cluster size (and the total number of clusters), as evidenced by our simulation results presented later in Section 3.5. This is especially true in the case of dense graphs, in which there are a large number of nodes within the sphere of interference. The approach described in the previous section is conservative and tries to avoid interference between initiators. If we allow for more concurrency, we can reduce the overall clustering time. In this section, we present a more optimistic approach to improve the network decomposition time.

In this approach, we allow for up to $\frac{n}{B}$ initiators to be concurrently active (on average) in the entire network within the clustering time t_c. The intuition for using this particular number is that at least $\frac{n}{B}$ clusters will be formed in a network of n nodes with an upper bound B on the cluster size. This can be expressed by the following equation:

$$E[X_{i+\frac{n}{B}:n} - X_{i:n}] \geq t_c \, , \;\; 0 \leq i \leq n - \frac{n}{B} \tag{16}$$

From Equation 12, it follows that the left hand side of Equation 16 is smallest for $i = 0$, therefore, it is sufficient to consider:

$$E[X_{\frac{n}{B}:n} - X_{0:n}] \geq t_c \tag{17}$$

For this approach, we choose $t_c = B^2$ in general irrespective of the local network topology or the specific clustering algorithm chosen. With a unique initiator, this is the maximum time required by any optimal algorithm that achieves the bound B, as derived in Section 2. This is sufficient for the Expanding Ring algorithm even with multiple initiators, and based on our experience from simulations, this value is sufficient for the Persistent algorithm on average with multiple initiators.

As before, we compute the value of the rate parameter λ for the initiator timers using the exponential distribution. Setting $s = n$, $k = \frac{n}{B}$, and $t_c = B^2$, from Equations 12 and 17, we get:

$$B^2 \;\leq\; \sum_{i=1}^{\frac{n}{B}} \frac{1}{(n-i+1)\lambda} \tag{18}$$

$$\leq\; \frac{1}{B\lambda} \tag{19}$$

Therefore, by transposing λ and B^2, we get: $\lambda \leq \frac{1}{B^3}$. Thus in order to minimize the expected time for network decomposition, we set $\lambda = \frac{1}{B^3}$. In Section 3.5, we show using simulations that $\lambda = \frac{1}{B^3}$ performs well over a range of network topologies. Setting $\lambda = \frac{1}{B^2}$ results in a larger number of clusters, whereas, $\lambda = \frac{1}{B^4}$ results takes a longer time for the decomposition to complete.

We note that the only design parameter used in this approach is the cluster size bound. This has the advantage that no knowledge of the network topology is required to set the initiator timers. Simulation results on the performance of the optimistic approach are provided in the next section.

3.5 Simulation results

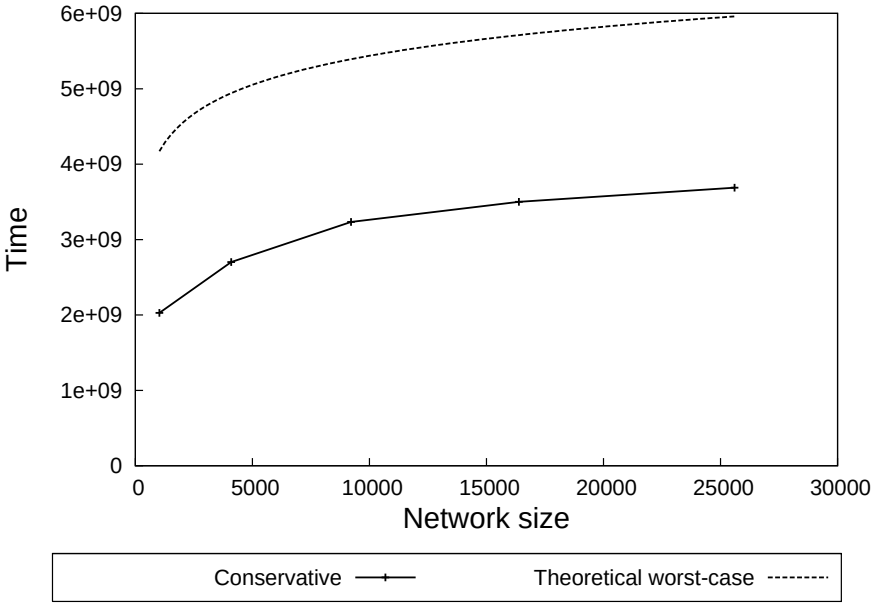
In this section, we present simulation results of the timer design approach over mesh topologies (additional simulation results for ring, clique, random graph and sensor network topologies can be found in [15]). In our simulations, we assume that the link delay is equal to one unit of time for each link.

In order to study the scalability of our approaches, we simulated the conservative approach, the optimistic approach, and an ideal baseline sequential approach (that guarantees that only one initiator is active in the entire network at any time) over mesh topologies of size varying from $1,024$ nodes to $25,600$ nodes.

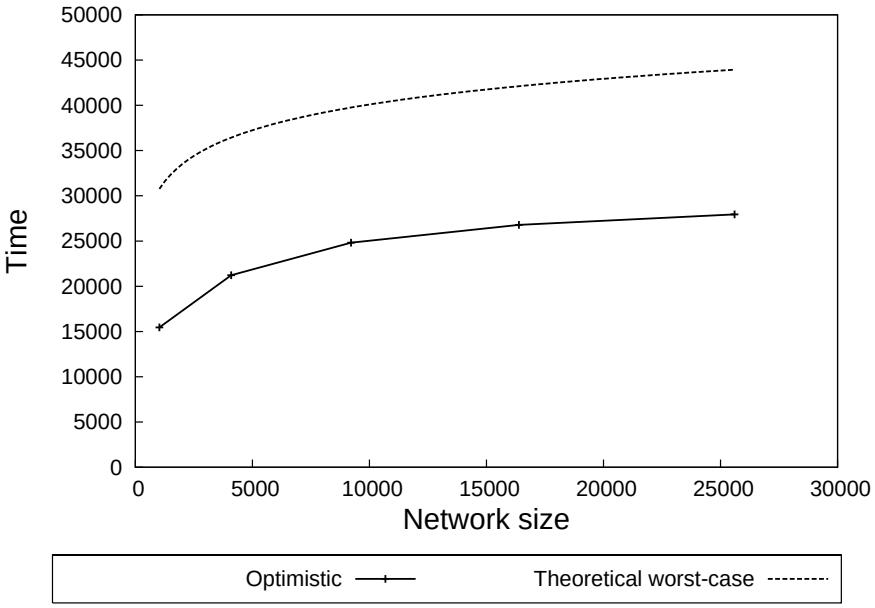
In Table 4, we summarize the number of clusters produced by the three approaches. We have used a computed value of $\lambda = 1.8 \times 10^{-9}$ for the conservative approach and $\lambda = 2.44 \times 10^{-4}$ for the optimistic approach. From this table, we see that the conservative approach using $\epsilon = 0.1$ produces the same number of clusters as the ideal sequential baseline.

Table 4: Number of clusters for conservative, optimistic, and sequential approaches over mesh topologies of various sizes. We used $\epsilon = 0.1$ for the conservative approach.

Network Size	Number of clusters		
	Sequential	Conservative	Optimistic
1,024	95.2	95.2	95.9
4,096	377.2	377.2	382.9
9,216	847.3	847.3	861.2
16,384	1,500.6	1,500.6	1,531.6
25,600	2,363.4	2,363.4	2,397.8



(a) Completion time for the conservative approach



(b) Completion time for the optimistic approach

Figure 6: Scalability of network decomposition with increasing network size. Each point is averaged over 100 runs each of the conservative approach with an ϵ of 0.1, the optimistic approach, and the sequential baseline on mesh topology for a cluster size bound of 16. The theoretical worst case on the average completion time is computed using Equation 15. Time is measured in units of maximum link delay.

The optimistic approach produces a slightly larger number of clusters.

Figure 6(a) shows that the time taken to cluster the entire network increases logarithmically with the network size, as predicted by our analysis, and the upper bound provided by Equation 15 is quite tight. From Figure 6(b), we observe that the optimistic approach exhibits a similar scaling behavior while running significantly faster than the conservative approach. We see from these results that our timer design scales well with network size when using the Persistent algorithm.

4 Discussion of Practical issues

In this section, we discuss the issues in practical application of our algorithms and our timer design to the self-organization of large wireless sensor networks. These issues are related to the robust maintenance of network structure under node failures and loss of messages over unreliable links.

Neighbor discovery: The clustering algorithms we proposed in this paper require that each node must know its immediate neighbors. We can achieve this requirement using well-known neighbor discovery protocols, for example, the one proposed in [23]. In this approach, each node periodically broadcasts a beacon message, which is discovered by other nodes that hear the beacon message. For robustness, a k-of-n scheme can be used, in which a node remains a valid neighbor as long as k out of every n beacons are received successfully. Therefore, this scheme can be used not only to identify neighbors for our clustering algorithms, but also to determine when a neighbor is no longer reachable.

Arrival of new nodes: When a new node is deployed in a sensor network (or a node wakes up from sleep), the neighborhood of this node may have already been clustered. In this case, the node will not receive any labeling message before its initiator timer fires, and it will attempt to create a new cluster. Since its neighbors are already clustered, the node will end up being in a small cluster by itself. As a result, the network may become fragmented into a large number of small clusters as new nodes arrive.

However, it is possible that one or more of its neighboring clusters are smaller than the bound. Therefore, the new node should attempt to join a neighboring cluster prior to initiating its own cluster. It can do so by sending a join request to each of its neighbors.

Upon receiving a join request, a node consults its initiator, and upon approval, sends an accept response to the new node. If the new node receives multiple accept responses, it must send confirmation to one of the neighbor and reject the others. This confirmation can

then be forwarded to the initiator node. This scheme is similar to the one described in [6].

Routing: In very large networks, it may become necessary to group clusters into higher level clusters. Standard approaches to routing in hierarchical multi-cluster, multi-hop wireless networks can be used in conjunction with the clustering algorithms (see [25, 28] and references therein). Many of these require global, possibly abstracted, knowledge of the network topology. A hierarchical distance vector protocol (e.g. [18]), hierarchical link state routing protocol (e.g. [23]), or hazy sighted routing protocol [24] may be used to acquire topology knowledge for routing.

Partition detection and healing: In general, with hierarchical clustering, two kinds of partitioning can occur. In the first case, the entire network is partitioned. Healing such a catastrophic partition involves recovery or replacement of failed links or nodes. In the second case, the entire network is not partitioned, but a cluster is internally partitioned due to the loss of nodes or links. This latter case, called the *subnetwork partition* problem, has been addressed in detail in [20].

In the context of the Persistent algorithm, we propose the following approach for dealing with partitions arising from the loss of a node or link. As we described earlier, the neighbor discovery procedure doubles as a robust link maintenance protocol. Using this a node can determine when its parent is no longer reachable, and can then attempt to compute an alternative path to the initiator of the cluster using knowledge of the cluster topology[5]. The node can then inform the initiator, which can then compute a different spanning tree for the cluster. If no path to the initiator can be found, then the node must initiate a relabeling procedure that causes the subtree of this node to become an independent cluster.

Loss or delayed delivery of messages: We analyzed the clustering algorithms in Section 2 in ideal settings in which no messages are lost. In practice, the clustering protocol must expect loss of messages, and therefore, we must use timeouts and retransmissions to ensure reliable message delivery. For example, we can use a link level retransmission scheme such as the one those specified for IEEE 802.11 [12] networks. In error-prone environments, forward error correction codes may be applied in conjunction with retransmissions to protect against message corruptions [4].

If the network can deliver messages after very long delays, then a sequence number scheme must be implemented to identify and discard old messages [19]. In extremely hostile environments, we may need to consider a number of more radical techniques [26].

Load balancing: The clusterhead (the initiator or another elected node in the cluster)

[5]The initiator can convey the cluster topology to all nodes in the cluster.

may commonly be used to forward messages outside the cluster, and as a result the increased load can deplete its energy rapidly. To ameliorate this, our algorithms can be used with load balancing approaches such as LEACH [11], in which different nodes become initiators in different rounds for sharing the energy load. Such initiator rotation could also allow for recovery in the event of failed nodes.

5 Conclusions and Future Work

In this paper, we made the following contributions.

Message-efficient clustering algorithms: We presented a novel approach to message-efficient clustering based on the concept of allocating *growth budgets* to neighbors. We presented a basic algorithm *Rapid*, which allocates budgets to neighbors once, and an enhanced algorithm *Persistent*, which reallocates the shortfall until the bound is reached or no more growth is possible. Unlike the expanding ring approach [22], our algorithms do not involve the initiator in each round, and do not violate the specified upper bound on the cluster size at any time.

Performance analysis of the clustering algorithms: We showed that the Persistent approach has optimal worst-case message complexity $O(B^2)$ (where B is the cluster size bound), when only a single initiator is present, whereas, the Expanding Ring algorithm has a higher[6] worst-case complexity $O(n)$ (where n is the number of nodes in the network). Although the Rapid algorithm has a message complexity that is linear in B, it can produce extremely small clusters of size $O(e^{W(\ln B)})$ (where W is the Lambert W function) in the worst case, even when only a single initiator is present.

Through analysis of sequential decomposition of regular topologies, we showed that in densely connected topologies, the Persistent algorithm is expected to use fewer messages on average than the Expanding Ring algorithm. On clique topologies, the Persistent algorithm requires only $O(n \log n)$ messages on average, whereas, the Expanding Ring algorithm requires $O(n^2)$ messages. We verified the analytical results using simulations over ring and clique topologies.

Simulations of the clustering algorithms: We confirmed our analytical results through extensive simulations over a wide range of topologies. We showed that our algorithms produce clusters of bounded size and low diameter, using significantly fewer messages than the expanding ring approach. In our simulations, the Persistent algorithm produced the

[6]In large networks, particularly wireless sensor networks, $n > B^2$.

fewest clusters using the fewest messages.

Timer design methodology: We addressed the issue of how long nodes must wait before they become initiators. We presented a methodology for designing the initiator timers. In our methodology, each node has an exponentially distributed timer. We introduced and used the concepts of the *sphere of influence* and the *sphere of interference* of an initiator to determine the rate parameter λ of the exponential distribution. Our method can offer a guarantee that initiators in the same neighborhood do not interfere with each other with chosen probability $1 - \epsilon$, where $0 < \epsilon < 1$.

We also presented an optimistic variant that reduces the total network decomposition time significantly, by allowing more initiators to be simultaneously active. More importantly, this variant does not require any knowledge of the network topology. In particular, we show through analysis that $\lambda = \frac{1}{B^3}$ is expected to work well. Through simulations, we have confirmed that this setting works well in practice. However, it results in a slightly higher number of clusters than the conservative approach.

Scalability to large networks: We derived the upper bound on the expected time for network decomposition using our method, which is asymptotically $\frac{\ln n+\gamma}{\lambda}$ (where γ is the Euler-Mascheroni constant). In comparison, a sequential approach in which only one initiator is allowed to be active at any time, can achieve network decomposition in time which is at least linear in the size of the network. Furthermore, in large networks, our approach is more robust than the sequential approach, since the latter requires distributed consensus. Through simulations over different topologies we demonstrated that the Persistent algorithm in conjunction with our timer design approaches can scale to large networks and produce a number of clusters comparable to the baseline sequential approach.

In summary, our clustering algorithms and timer design methodology can be applied to achieve self-configuration of large wireless sensor networks, and other networks. Based on a survey of related work, we believe our work is unique in its focus on distributed clustering algorithms that achieve message efficiency as well as a bounded cluster size. Furthermore, our work offers a rigorous approach to the design of initiator timers that can scale to very large networks. As a future work, it would be interesting to investigate other possible approaches to set the timers and compare them with the method proposed in our paper.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks (Elsevier)*, 38(4):393–422, March 2002.

[2] B.C. Arnold, N. Balakrishnan, and H.N. Nagaraja, *A first course in order statistics*, Wiley series in probability and mathematical statistics, John Wiley, New York, NY, USA, ISBN: 0471574163, 1992.

[3] B. Awerbuch, A. V. Goldberg, M. Luby, and S.A. Plotkin, "Network decomposition and locality in distributed computation," *Proceedings of the IEEE 30th Annual Symposium on Foundations of Computer Science*, Research Triangle Park, NC, USA, 1:364–369, 30 October–1 November 1989.

[4] E. Ayanoglu, S. Paul, T.F. LaPorta, K.K. Sabnani, and R.D. Gitlin, "AIRMAIL: a link-layer protocol for wireless networks," *ACM/Baltzer Wireless Networks*, 1(1):47–60, February 1995.

[5] S. Banerjee and S. Khuller, "A clustering scheme for hierarchical control in multihop wireless networks," *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, Anchorage, AK, USA, 2:1028–1037, 22–26 April 2001.

[6] C.-C. Chiang, "Routing in clustered multihop, mobile wireless networks with fading channel," *Proceedings of the IEEE Singapore International Conference on Networks (SICON'97)*, 1:197–211, April 1997.

[7] K.A. Delin and S.P. Jackson, "The Sensor Web: A New Instrument Concept," *Proceedings of SPIE's Symposium on Integrated Optics,* San Jose, CA, USA, January 2001. (Available online at http://sensorwebs.jpl.nasa.gov/resources/sensorweb-concept.pdf)

[8] P. Erdos and A. Renyi, "On the evolution of random graphs," *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[9] P. Fraigniaud, A. Pelc, D. Peleg, and S. Perennes, "Assigning labels in unknown anonymous networks," *Proceedings of the 19th Annual ACM SIGACT-SIGOPS symposium on Principles of Distributed Computing*, Portland, OR, USA, 1:101–111, July 2000.

[10] M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of distributed consensus with one faulty processor," *Journal of the ACM*, 32(2):374–382, 1985.

[11] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocols for wireless microsensor networks," *Proceedings of the 33rd Hawaiian International Conference on Systems Science (HICSS-33)*, Maui, HI, USA, 4–7 January 2000.

[12] IEEE 802.11, 1999 Edition (ISO/IEC 8802-11: 1999) IEEE Standards for Information Technology–Telecommunications and Information Exchange between Systems–Local and Metropolitan Area Network–Specific Requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[13] L. Kleinrock and F. Kamoun, "Hierarchical routing for large networks: performance evaluation and optimization," *Computer Networks (Elsevier)*, 1(1):155–174, 1977.

[14] R. Krishnan, R. Ramanathan, and M. Steenstrup, "Optimization algorithms for large self-structuring networks," *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, New York, NY, USA, 1:71–78, 21–25 March 1999.

[15] R. Krishnan, "Efficient Self-Organization of Wireless Sensor Networks," *Ph.D. Dissertation*, Boston University, Boston, USA, January 2004.

[16] S. Lindsey and C.S. Raghavendra, "PEGASIS: Power Efficient GAthering in Sensor Information Systems," *Proceedings of IEEE Aerospace Conference*, Volume 3, pp. 3-1125–3-1130, Big Sky, MT, USA, 9–16 March 2002.

[17] H.O. Marcy, J.R. Agre, C. Chien, L.P. Clare, N. Romanov, and A. Twarowski, "Wireless sensor networks for area monitoring and integrated vehicle health management applications," *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Portland, OR, USA, (Collection of Technical Papers, Volume 1, AIAA Paper 99-4557), 9–11 August 1999.

[18] S. Murthy and J.J. Garcia-Luna-Aceves, "An efficient routing protocol for wireless networks," *ACM Mobile Networks and Applications Journal*, Special issue on Routing in Mobile Communication Networks, 1(2):183–197, 1996.

[19] R.J. Perlman, "Fault-Tolerant Broadcast of Routing Information," *Computer Networks (Elsevier)*, 7:395–405, 1983.

[20] R.J. Perlman, "Hierarchical networks and the subnetwork partition problem," *Computer Networks (Elsevier)*, 9:297–303, 1985.

[21] S. Ray, D. Starobinski, A. Trachtenberg, and R. Ungrangsi, "Robust Location Detection with Sensor Networks," *IEEE JSAC (Special Issue on Fundamental Performance Limits of Wireless Sensor Networks)*, to appear.

[22] C. V. Ramamoorthy, A. Bhide, and J. Srivastava, "Reliable clustering techniques for large, mobile packet radio networks," *Proceedings of the 6th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '87)*, San Francisco, CA, USA, 1:218–226, 31 March–2 April 1987.

[23] R. Ramanathan and M. Steenstrup, "Hierarchically-organized, multihop mobile wireless networks for quality-of-service support," *ACM/Baltzer Mobile Networks and Applications*, 3(1):101–119, June 1998.

[24] C. Santivanez, R. Ramanathan, and I. Stavrakakis, "Making link-state routing scale for ad hoc networks," *Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, USA, 1:22–32, 4–5 October 2001.

[25] M. Steenstrup, "Cluster-Based Networks," In *Ad Hoc Networking*, edited by Charles E. Perkins, Chapter 4, Addison-Wesley Professional Computing Series, ISBN: 0201309769, 2001.

[26] J.P.G. Sterbenz, R. Krishnan, R.R. Hain, A.W. Jackson, D. Levin, R. Ramanathan, and J. Zao, "Survivable mobile wireless networks: issues, challenges, and research directions," *ACM Workshop on Wireless Security (WiSe)*, Atlanta, GA, USA, 1:31–40, 28 September 2002.

[27] L. Subramanian and R.H. Katz, "An architecture for building self-configurable systems," *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking and computing*, pp. 63–73, Boston, MA, USA, 2000.

[28] C.-K. Toh, *Ad hoc mobile wireless networks: protocols and systems*, Prentice Hall, ISBN: 0130078174, December 2001.

[29] R.M. Young, "Euler's constant," *Mathematical Gazette 75*, 472:187–190, 1991.

APPENDIX A: Pseudo-Code of Algorithms

In this appendix, we provide detailed pseudo-code of the clustering algorithms described in Section 2.

Algorithm 1 Expanding Ring

Require: Network $G = (V, E)$, every node knows its neighbors, and the cluster size bound B.
Ensure: Cluster of maximum size B containing the INITIATOR.

```
1: for all nodes v ∈ V do
2:   if INITIATOR then
3:     hops ← 0                                   /* Number of hops from initiator */
4:     repeat
5:       hops ← hops + 1
6:       SEND hops to all neighbors
7:       wait for all neighbors to respond
8:       reject hops received from other nodes
9:       compute cluster size and set of nodes in last layer
10:    until cluster size ≥ B or no growth achieved in the last round
11:    choose excess nodes from last layer to drop
12:    SEND list of nodes to drop to all neighbors
13:    DECIDE cluster
14:    END
15:  else
16:    while RECEIVE message do
17:      if received hops from parent then
18:        /* Node assigns as parent first neighbor it receives hops from */
19:        hops ← hops − 1
20:        if hops = 0 then
21:          add self to last layer
22:        else
23:          SEND hops to all neighbors except parent
24:          wait for all neighbors to respond
25:          reject hops received from other nodes
26:        end if
27:        compute subtree size and set of nodes in last layer
28:        SEND subtree size and last layer to parent
29:      else if received list of nodes to drop from parent then
30:        if self in list then
31:          mark self unclustered
32:        else
33:          SEND list of nodes to drop to children
34:        end if
35:      else
36:        reject hops received from non-parent node
37:      end if
38:    end while
39:  end if
40: end for
```

Algorithm 2 Rapid

Require: Network $G = (V, E)$, every node knows its neighbors, and the cluster size bound B.
Ensure: Cluster of maximum size B containing the INITIATOR.

```
1: for all nodes v ∈ V do
2:   /* Receive budget and account for self and parent */
3:   if INITIATOR then
4:     β ← B − 1                                   /* β tracks available budget */
5:     δ ← degree(v)                               /* δ tracks available neighbors */
6:   else
7:     RECEIVE budget from node x
8:     parent ← x
9:     β ← budget − 1
10:    δ ← degree(v) − 1
11:  end if
12:  /* Allocate residual budget among neighbors */
13:  if β ≥ 1 then
14:    γ ← ⌊β/δ⌋                                 /* γ tracks budget per neighbor */
15:    SEND budget γ + 1 to β modulo δ arbitrary non-parent neighbors of v
16:    SEND budget γ (if non-zero) to all other non-parent neighbors of v
17:  end if
18:  /* Wait for contacted neighbors to respond */
19:  while not received subtree size from neighbors sent a budget do
20:    RECEIVE message
21:    if received budget from other node y then
22:      SEND zero subtree size to node y
23:    end if
24:  end while
25:  /* Termination of the algorithm */
26:  if INITIATOR then
27:    DECIDE cluster size (= total subtree size)
28:    END
29:  else
30:    SEND total subtree size to parent
31:    END
32:  end if
33: end for
```

Algorithm 3 Persistent

Require: Network $G = (V, E)$, every node knows its neighbors, and the cluster size bound B.
Ensure: Cluster of maximum size B containing the INITIATOR.

```
1: for all nodes v ∈ V do
2:   /* Receive initial budget and account for self and parent */
3:   if INITIATOR then
4:     β ← B − 1, δ ← degree(v)          /* β, δ track available budget and neighbors */
5:   else
6:     RECEIVE budget from node x
7:     parent ← x, β ← budget − 1, δ ← degree(v) − 1
8:   end if
9:   REALLOC:                                  /* Persistently reallocate budget */
10:  while β > 0 AND δ > 0 do
11:    /* Allocate budget to eligible neighbors */
12:    γ ← ⌊β/δ⌋                              /* γ tracks budget per neighbor */
13:    SEND budget γ+1 to β modulo δ arbitrary non-parent neighbors of v that are unexplored
       or have met earlier budgets, and budget γ (if non-zero) to all other non-parent neighbors
       of v that are unexplored or have met earlier budgets
14:    /* Wait for contacted neighbors to respond */
15:    while not received subtree size from neighbors sent a budget do
16:      RECEIVE message
17:      if received budget from other node y then
18:        SEND zero subtree size to node y                    /* Reject offer */
19:      end if
20:    end while
21:    β ← β− total subtree size                  /* Determine residual budget */
22:    δ ← number of neighbors that have either exhausted their budget or remain unexplored
23:  end while
24:  /* Termination of the algorithm */
25:  if INITIATOR then
26:    DECIDE cluster size (= total subtree size)
27:    END
28:  else
29:    SEND total subtree size to parent
30:    while RECEIVE message do
31:      /* Handle budget reallocations */
32:      if received budget from parent then
33:        goto REALLOC:
34:      else if received budget from other node y then
35:        SEND zero subtree size to node y
36:      end if
37:    end while
38:  end if
39: end for
```

APPENDIX B: Proof of Theorem 1

In this appendix, we prove Theorem 1. We first show that the worst-case cluster size $T(B)$ produced by the Rapid algorithm is lower bounded by $e^{W(\ln B)}$.

Lemma 7 *$T(B) \geq e^{W(\ln B)}$, where W is the Lambert-W function.*

Proof: Suppose the worst-case cluster size produced is $T(B) \leq B$. In order to achieve the worst case, the algorithm must achieve the minimum growth possible at each stage (namely 1), and waste the maximum possible fraction of the total budget available at that stage.

Clearly, each node in the worst-case cluster has no more than $T(B)$ neighbors and at least one neighbor which will be able to use the budget, since G is connected and $n > B$.

In the worst case, the initial budget B is reduced by a factor of $T(B)$ at each stage. With a growth by 1 node at each stage and an initial budget of B, the total number of steps does not exceed $T(B)$.

Since the budget must be exhausted after $T(B)$ stages, this leads to the inequality:

$$\frac{B}{T(B)^{T(B)}} \leq 1. \tag{20}$$

The solution for $f(x)^{f(x)} = x$ is $f(x) = e^{W(\ln x)}$ where $W(x)$ is the Lambert W-function. Thus $T(B) \geq e^{W(\ln B)}$. ■

Next, we show that the upper bound on the worst-case cluster size produced by the Rapid algorithm is $e^{W(\ln B)} + 2$.

Lemma 8 *$T(B) \leq e^{W(\ln B)} + 2$, where W is the Lambert-W function.*

Proof: Let the cluster size produced by the Rapid algorithm in the construction shown in Figure 2 be $T(B) \leq B$ for $1 \leq B \leq n$. We are interested in the non-trivial cases of $B \geq T(B) > 2$.

The initiator is provided a budget of B of which it allocates 1 for itself. The budget available at the second stage along the fast path is

$$\frac{B-1}{T(B)-1} \leq \frac{B}{T(B)-2}. \tag{21}$$

By construction, all other neighbors of the initiator join the cluster (along the fast path) before receiving the budget along the slow links. All budget allocations on the slow links are therefore wasted.

Similarly, the budget available to the third stage along the fast path is at most

$$\frac{B}{(T(B)-2)^2}. \tag{22}$$

The process continues and the budget after $T(B)-1$ stages is 1, that is,

$$\frac{B}{(T(B)-2)^{(T(B)-2)}} \geq 1. \tag{23}$$

Setting $g(B) = T(B) - 2$ and solving for $B \geq g(B)^{g(B)}$, we get, $T(B) \leq e^{W(\ln B)} + 2$, where $W(x)$ is the Lambert W-function. ■