A Self-stabilizing (k,r)-clustering Algorithm with Multiple Paths for Wireless Ad-hoc Networks

Andreas Larsson
Computer Science and Engineering,
Chalmers University of Technology
Email: larandr@chalmers.se

Philippas Tsigas
Computer Science and Engineering,
Chalmers University of Technology
Email: tsigas@chalmers.se

***Abstract*—Wireless Ad-hoc networks are distributed systems that often reside in error-prone environments. Self-stabilization lets the system recover autonomously from an arbitrary state, making the system recover from errors and temporarily broken assumptions. Clustering nodes within ad-hoc networks can help forming backbones, facilitating routing, improving scaling, aggregating information, saving power and much more. We present the first self-stabilizing distributed (k,r)-clustering algorithm. A (k,r)-clustering assigns k cluster heads within r communication hops for all nodes in the network while trying to minimize the total number of cluster heads. The algorithm uses synchronous communication rounds and uses multiple paths to different cluster heads for improved security, availability and fault tolerance. The algorithm assigns, when possible, at least k cluster heads to each node within O(r) rounds from an arbitrary configuration. The set of cluster heads stabilizes, with high probability, to a local minimum within O(gr log n) rounds, where n is the size of the network and g is an upper bound on the number of nodes within 2r hops.**

I. Introduction

Starting from an arbitrary state, self stabilizing algorithms let a system stabilize to, and stay in, a consistent state [6]. There are many reasons why a system could end up in an inconsistent state of some kind. Assumptions that algorithms rely on could temporarily be invalid. Memory content could be changed by radiation or other elements of harsh environments. Battery powered nodes could run out of batteries and new ones could be added to the network. It is often not feasible to manually configure large ad-hoc networks to recover from events like this. Self-stabilization is therefore often a desirable property of algorithms for ad-hoc networks. However, self-stabilization comes with increased costs, so a tradeoff is made. A self-stabilizing algorithm can never stop because you can not know when temporary faults occur, but it can converge to a result that holds as long as all assumptions hold. Furthermore, there are often overheads in the algorithm tied to the need to recover from arbitrary states. It can be added computations, increased size of messages or increased number of needed rounds to achieve something.

An algorithm for clustering nodes together in an ad-hoc network serves an important role. Back bones for efficient communication can be formed using cluster heads. Clusters can be used for routing messages. Cluster heads can be responsible for aggregating data, e.g. sensor readings in an ad-hoc sensor network, into reports to decrease the number of individual messages needed to rout through the network. Hierarchies of clusters on different levels can be used for improved scaling of a large network. Nodes in a cluster could take turns doing energy costly tasks to save power over all.

Clustering is a well studied problem. Due to space constraints, for references to the area in general, we point to the survey of the area with regard to wireless ad-hoc networks by Chen, Liestam and Liu in [4] and the survey by Abbasi and Younis in [1] for wireless sensor networks. We will focus on self-stabilization, redundancy and some security aspects. One way of clustering nodes in a network is for nodes to associate themselves with one or more cluster heads. In the (k,r)-clustering problem each node in the network should have at least k cluster heads within r communication hops away. This might not be possible for all nodes if the number of nodes within r hop from them is smaller than k. In such cases a best effort approach can be taken for getting as close to k cluster heads as possible for those nodes. The clustering should be achieved with as few cluster heads possible. To find the global minimum number of cluster heads is in general too hard, algorithms provide an approximation. The (1,r)-clustering problem, a subset of the (k,r)-clustering problem, can be formulated as a classical set cover problem. This was shown to be NP complete in [10]. Assuming that the network allows k cluster heads for each node, the set of cluster heads forms a total (k,r)-dominating set in the network. In a *total* (k,r)-dominating set the nodes in the set also need to have k nodes in the set within r hops, in contrast to an ordinary (k,r)-dominating set in which this is only required for nodes not in the set.

There is a multitude of existing clustering algorithms for ad-hoc networks of which a number is self-stabilizing. Johnen and Nguyen present a self-stabilizing (1,1)-clustering algorithm that converges fast in [9]. Dolev and Tzachar tackle a lot of organizational problems in a self-stabilizing manner in [7]. As part of this work they present a self-stabilizing (1,r)-clustering algorithm. Caron, Datta, Depardon and Larmore present a self-stabilizing (1,r)-clustering in [3] that takes weighted graphs into account.

There is a number of papers that do not have self-stabilization in mind. Fu, Wang and Li consider the (k,1)-clustering problem in [8]. In [13] the full (k,r)-clustering problem is considered and both a centralized and a distributed algorithm for solving this problem are presented. Wu and Li also consider the full (k,r)-clustering in [15].

Other algorithms do not take the cluster head approach. In [14], sets of nodes that all can communicate directly with each other are grouped together without assigning any cluster heads. In this paper malicious nodes that try to disturb the protocol are also considered, but self-stabilization is not considered.

A. Our Contribution

We have constructed the first, to the best of our knowledge, self-stabilizing (k,r)-clustering algorithm for ad-hoc networks. The algorithm is based on synchronous rounds and makes sure that, within $O(r)$ rounds, all nodes have at least k cluster heads (or all nodes within r hops if a node has less than k nodes within r hops) using a deterministic scheme. A randomized scheme complements the deterministic scheme and lets the set of cluster heads stabilize to a local minimum. It stabilizes within $O(gr \log n)$ rounds with high probability, where g is a bound on the number of nodes within $2r$ hops, and n is the size of the network.

We prove quick selection of enough cluster heads. Once the system fulfills the cluster head requirements, of k cluster heads within r hops for all nodes, the requirements will continue to hold from that point on. We also prove that the set of cluster heads converges towards a local minimum. Under the extra assumption that timers are synchronized, we show an upper bound on the number of rounds it takes, with high probability, for the set of cluster heads to reach a local minimum. Furthermore, experimentally we show that without this extra assumption used in the proof the system stabilizes approximately equally fast. We also present experimental results on how the algorithm copes with changes to the topology and on how our results compares with global optima.

Some initial ideas that lead to these results was previously published in [11]. It is a brief announcement with few technical details and without any proofs or experimental results.

B. Document Structure

Our contribution is presented as follows. In section II we introduce the system settings. Section III describes the algorithm. Section IV proves the properties of the algorithm. We discuss experimental results, security and redundancy and how different system settings would affects the properties of the algorithm in Section V.

II. System Settings

We assume a static network. Changes in the topology are seen as transient faults. We denote the set of all nodes in the network $\mathcal{P}$ and the size of the network $n = |\mathcal{P}|$. We impose no restrictions on the network topology other than that an upper bound, g, on the number of nodes within $2r$ hops of any node is known (see below).

The set of neighbors, N_i, of a node p_i is all the nodes that can communicate directly with node p_i. In other words, a node $p_j \in N_i$ is one hop from node p_i. We assume a bidirectional connection graph, i.e. that $p_i \in N_j$ iff $p_j \in N_i$.

Constants:
i : *id of executing processor.*
r : *number of hops within we consider a neighborhood.*
k : *the number of clusterheads to elect.*
g : *upper bound on the number of nodes within* $2r$ *hop.*
$T = 8gr$: *length of an escape period.*

Variables:
state $\in$ {HEAD, ESCAPING, SLAVE} :
  The state of the node. Initially set to SLAVE.
timer : *Integer. Timer for escape attempts. Initially set to* T-1.
estart : *Integer. The escape schedule. Initially set to* 0.
estate $\in$ {SLEEP, INIT, FLOOD, HOPE} :
  State for escape attempts. Initially set to SLEEP.
heads : *Set of Id:s. Initially set to* $\emptyset$.
S & *Z*: *Sets of* $<$ *Id,State*$>$ *tuples.*
  Initially set to $\{<$ *i,state*$>\}$.

External functions and macros:
LBcast(m) : *Broadcasts message m to direct neighbors.*
LBrecv(m) : *Receives a message from direct neighbor.*
smallest(a,A) : *Returns the* **min**($|A|$,a) *smallest id:s in A.*
cds(A): $\{<j,s,t> \in A : t = \max_\tau \{\tau : <j,s,\tau> \in A\}\}$
cdj(B): $\{<j,t> \in B : t = \max_\tau \{\tau : <j,\tau> \in B\}\}$

Fig. 1. Constants, variables, external functions and macros for the algorithm in Fig. 2.

The neighborhood, G_i^r of a node p_i is all the nodes (including itself) at most r hops away from p_i. Let $g \geq \max_j |G_j^{2r}|$ be a bound, known by the nodes, on the number of nodes within $2r$ hops.

The system is synchronous and progresses in rounds. Each round has two phases. First in the receipt phase each node p_i receives messages from all of its immediate neighbors $p_j \in N_i$. Then in the step phase each node p_i after performing the appropriate calculations broadcasts a message to all nodes $p_j \in N_i$. We assume that a broadcast by a node p_i is received reliably by all processors $p_j \in N_i$ in the receipt phase of the respective round. In our proofs for convergence times of our algorithm we use an assumption of synchronized timers. Synchronized timers is *not* an assumption of the algorithm itself and is *not* needed for the algorithm to work correctly. Furthermore, we demonstrate experimentally that it does not significantly affect convergence times either.

III. Self-stabilizing Algorithm for (k,r)-clustering

The goal of the algorithm is, using as few cluster heads as possible, for each node p_i in the network to have a set of at least k cluster heads within its r-hop neighborhood G_i^r. This is not possible if a node p_i has $|G_i^r| < k$. Therefore, we require that $|C_i^r| \leq k_i$, where $C_i^r \subseteq G_i^r$ is the set of cluster heads in the neighborhood of p_i and $k_i = \min(k, |G_i^r|)$ is the closest number of cluster heads to k that node p_i can achieve. We do not strive for a global minimum. That is too costly. We achieve a local minimum, i.e. a set of cluster heads in which no cluster head can be removed without violating the (k,r) goal.

```
on step phase:
  if $timer < 0 \vee timer \geq T\text{-}1$
    $timer \leftarrow 0$
  else
    $timer \leftarrow timer + 1$
  $S \leftarrow Z$
  $heads \leftarrow \{j : < j, \text{HEAD}> \; \in S\}$
  /* Escaping */
  if $state$ in $\{\text{HEAD, ESCAPING}\}$
    $updateestate()$
    if $estate = \text{INIT} \wedge state = \text{HEAD} \wedge |heads| > k$
      $state \leftarrow \text{ESCAPING}$
      $heads \leftarrow heads \setminus \{i\}$
    else if $estate = \text{SLEEP} \wedge state = \text{ESCAPING}$
      $state \leftarrow \text{SLAVE}$
  if $state = \text{SLAVE}$
    $estate \leftarrow \text{SLEEP}$
    $estart \leftarrow 0$
  /* Add heads */
  if $|heads| < k$
    let $a = k \text{ -}|heads|$
    let $A = \{j: < j, \cdot> \; \in S\} \setminus heads$
    $heads \leftarrow heads \cup \text{smallest}(a, A)$
  /* Join and send state */
  for $each\ j \in heads$
    if $j \neq i$
      $forwardjoin(< j, r> )$
    else
      $state \leftarrow \text{HEAD}$
  $Z \leftarrow \{< i,state> \}$
  $sendstate(< i, state, r> )$

function $updateestate$:
  if $timer = 0$
    $estart \leftarrow uniformlyrandom(\{0, 1, \ldots T\text{-}2r\text{-}2\})$
  if $timer \in [0, estart\text{-}1]$:
    $estate \leftarrow \text{SLEEP}$
  else if $timer \in [estart, estart]$
    $estate \leftarrow \text{INIT}$
  else if $timer \in [estart{+}1, estart{+}2r\text{-}1]$
    $estate \leftarrow \text{FLOOD}$
  else if $timer \in [estart{+}2r, estart{+}2r]$
    $estate \leftarrow \text{HOPE}$
  else if $timer \in [estart{+}2r{+}1, T\text{-}1]$
    $estate \leftarrow \text{SLEEP}$

function $receivedstate(< j, jstate, ttl> ), i \neq j$:
  $js \leftarrow jstate$
  if $js = \text{ESCAPING} \wedge j \in heads$
    if $|heads| \leq k$
      $js \leftarrow \text{HEAD}$
    else
      $heads \leftarrow heads \setminus \{j\}$
  let $ss = \{s : < j, s> \; \in Z\} \cup \{js\}$
  if $\text{HEAD} \in ss$:
    $js \leftarrow \text{HEAD}$
  else if $\text{ESCAPING} \in ss$
    $js \leftarrow \text{ESCAPING}$
  else
    $js \leftarrow \text{SLAVE}$
  $Z \leftarrow \{< o, s> : < o, s> \; \in Z \wedge o \neq j\} \cup \{< j, js> \}$

  $ttl \leftarrow \mathbf{max}(1, \mathbf{min}(r, ttl))$
  if $ttl > 1$:
    $forwardstate(< j, jstate, ttl\text{-}1> )$

function $receivedjoin(< j, ttl> )$:
  $ttl \leftarrow \mathbf{max}(0, \mathbf{min}(r, ttl))$
  if $j = i \wedge estate \notin \{\text{INIT, FLOOD}\}$
    $state \leftarrow \text{HEAD}$
  else if $ttl > 1$
    $forwardjoin(< j, ttl \text{ -}1> )$

on $\text{LBrecv}(< j, jstateset, jjoinset> )$:
  for $each < o,ostate,ottl> \; \in jstateset$
    if $o \neq i$:
      $receivedstate(< o,ostate,ottl> )$
  for $each < o,ostate,ottl> \; \in jjoinset$
    $receivedjoin(< o,ottl> )$

function $forwardstate(tuple)$:
  $stateset \leftarrow stateset \cup tuple$

function $forwardjoin(tuple)$:
  $joinset \leftarrow joinset \cup tuple$

function $sendstate(tuple)$:
  $forwardstate(tuple)$
  $stateset \leftarrow \text{cds}(stateset)$
  $joinset \leftarrow \text{cdj}(joinset)$
  $\text{LBcast}(< i,stateset,joinset> )$
  $stateset \leftarrow \emptyset$
  $joinset \leftarrow \emptyset$
```

Fig. 2. Pseudocode for the self-stabilizing clustering algorithm.

The basic idea of the algorithm is for cluster heads to constantly broadcast the fact that they are cluster heads and for all nodes to constantly broadcast a list of nodes they consider to be cluster heads. This list of cluster heads consists both of nodes that are known to be cluster heads and, additionally, nodes that are elected to become cluster heads. The content of the broadcasts are forwarded r hops, but in an aggregated form to keep message sizes down. The election process might establish too many cluster heads. Therefore, there is a mechanism for cluster heads to drop their cluster head roles, to *escape*, eventually establishing a local minimum of cluster heads forming a total (k,r)-dominating set (or, if not possible given the topology, fulfilling $|C^r_j| \geq k_j$ for any node p_j). The choice of which nodes to pick when electing cluster heads is based on node ID in order to limit the number of unneeded cluster heads that are elected when new cluster heads are needed.

One could imagine an algorithm that in a first phase adds cluster heads and thereafter in a second phase removes cluster heads that are not needed. To achieve self-stabilization however, we cannot rely on starting in a predefined state. Recovery from an inconsistent state might start at any time. Therefore, in our algorithm there are no phases and the mechanism for adding cluster heads runs in parallel with the mechanism for removing cluster heads and none of them ever stops.

In each round each node sends out its state and forwards states of others. A cluster head node normally has the state HEAD and a non cluster head node always has state SLAVE. If a node p_i in any round finds out that it has less than k cluster heads it selects a set of other nodes that it decides to elect as cluster heads. Node p_i then elects established cluster head nodes and any newly picked nodes by sending a *join* message to them. Any node that is not a cluster head becomes a cluster head if it receives a join addressed to it.

We take a randomized approach for letting nodes try to drop their cluster head responsibility. Time is divided into periods of T rounds. A cluster head node p_i picks uniformly at random one round out of the $T - 2r - 1$ first rounds in the period as a possible starting round, $estart_i$, for an escape attempt. If p_i has more than k cluster heads in round $estart_i$, then it will start an escape attempt. When starting an escape attempt a node sets it state to ESCAPING and keeps it that way for a number of rounds to make sure that all the nodes in G_i^r will eventually know that it tries to escape. A node $p_j \in G_i^r$ that would get fewer than k cluster heads if p_i would stop being a cluster head can veto against the escape attempt. This is done by recording the state of p_i as HEAD and thus continuing to send joins addressed to it. If p_j, on the other hand, has more than k cluster heads it would not need to veto. Thus, by accepting the state of p_i as ESCAPING, p_j will not send any join to p_i. After a number of rounds all nodes $G_i \setminus \{i\}$ will have had the opportunity to veto the escape attempt. If none of them objected, at that point p_i will get no joins and can set its state to SLAVE.

If an escape attempt by p_i does not overlap in time with another escape attempt it will succeed if and only if $\min_{p_j \in G_i^r} |C_j^r| > k$. If there are overlaps by other escape attempts, the escape attempt by p_i might fail even in cases where $\min_{p_j \in G_i^r} |C_j^r| > k$. The random escape attempt schedule therefore aims to minimize the risk of overlapping attempts.

The pseudocode for the algorithm is described in Fig. 2 with accompanying constants, variables, external functions and macros in Fig. 1. In the step phase of each round lines 1-31 are executed. The code for the receipt phase can be found in lines 47-72. To have only one message for each node sent per round, all forwarding and sending of messages in lines 1-72 use the functions in lines 74-93 that collect everything that is to be sent until the end of the step phase where one message is sent out.

IV. Correctness

In Section IV-A we will show that within $O(r)$ rounds we will have $|C_i^r| \geq k_i$ for any node p_i. First we show that this holds while temporarily disregarding the escaping mechanism, and then that it holds for the general case in Theorem 1.

In Section IV-B we will show that a cluster head node p_i can become slave if it is not needed and if it tries to escape undisturbed by other nodes in G_i^{2r}. We continue to show that the set of nodes converges, with high probability, to a local minimum within $O(gr \log n)$ rounds under the assumption that the timers of all nodes in the network are synchronized in Theorem 2.

Finally we present the message complexity in Theorem 3 in section IV-C.

Definition 1: If all assumptions about the network hold and all nodes follow the protocol throughout the entire round s then round s is called a *legal* round.

Definition 2: For a node p_i to be a *cluster head* is equivalent to $state_i \in \{\text{HEAD}, \text{ESCAPING}\}$. For a node p_i to be a slave is equivalent to $state_i = \text{SLAVE}$. For a node p_j, we define C_j^r as the set of cluster heads in G_j^r. Furthermore, we define H_x to be the set of cluster heads in the network in a round x.

Definition 3: A node *initiates an escape attempt* in round s if lines 12-13 are executed in round s. In other words in round s node p_i has $state_i = \text{HEAD}$ at line 1, $|heads_i| > k$ after executing line 7 and then line 10 sets $estate_i$ to INIT. Thereafter, the condition holds at line 11 and lines 12-13 are executed.

A. Getting Enough Cluster Heads

In this section we build up a case showing that the algorithm will elect enough cluster heads. We show that nodes get to know their neighborhood (Lemma 1), that they get to know the state of nodes in their neighborhoods (Lemma 2), that cluster heads are elected (Lemmas 3, 4 and 5). Finally, in Theorem 1, we show that within a $O(r)$ rounds each node in the network have enough cluster heads within r hops (topology allowing).

We begin by showing that nodes get to learn their neighborhood, G_i^r.

Lemma 1: Assume that round s and all following rounds are legal. For any node p_i, $\{p_j : < p_j, \cdot > \in S_i\} = G_i^r$ holds in the step phase of round $s+r$ and throughout rounds $s+r+1+t$ for any non-negative t.

Proof: In every round any node p_j broadcasts its id and state (line 31) with *ttl* set to r. The *ttl* is a *time to live* value that denotes how many more hops a message should be forwarded and is decreased by one every time the associated message is received. When *ttl* reaches 1 the message is not forwarded any more. Therefore, during the following r rounds the id and the state is forwarded r hops away (lines 63-65). Consider a node $p_j \in G_i^r$ $(i \neq j)$ that is $\hat{r}$ hops away from p_i. At round $s + r + t$ node p_i gets the id and state message that originated from node p_j at round $s + r - \hat{r} + t$. As t is non-negative and $\hat{r} \leq r$ we know that p_j sent a message with its state and id at round $s + r - \hat{r} + t \geq s$. For node p_i itself: (1) it adds itself to Z during each round (line 30), and (2) the only other line that could change Z is line 61 and is not executed in case $j = i$ and (3) S is set to Z at the beginning of the step phase. Therefore $G_i^r \subseteq \{p_j : < p_j, \cdot > \in S_i\}$ in the step phase of round $s + r$ and thereafter.

A message with id and state of a node $p_j \notin G_i^r$ that is being received by some node p_i in round s could potentially lead to an id in S_i that is not in G_i^r However such a message can not be sent out in round s with a *ttl* greater than $t - 1$ (lines 63-65) and Z is cleared from nodes $p_j \neq p_i$ in every round in line 30. Therefore p_j could reach p_i in rounds $s + 1$ to $s + r - 1$, but not as late as $s + r + t$ for a non-negative

t. Therefore $G_i^r \supseteq \{p_j : < p_j, \cdot > \in S_i\}$ in the step phase of round $s + r$ and thereafter. ■

We continue with showing that nodes within r hops get to know the state of a node that stays in one state.

Lemma 2: If a node p_i has the same state σ in rounds s to $s+r-1$, then any node $p_j \in G_i^r \setminus \{p_i\}$ will receive the state σ and only state σ for p_i in round $s + r$.

Proof: Node p_i sends out its state with a ttl of r in each round (line 31). Nodes that receive this state message with a ttl greater than 1 will forward the state with a ttl of one less (lines 64-65). Thus a message from p_i originating in round $s - t$ (for a positive t) can possibly be received by nodes in G_i^r in the rounds $s-t+1$ to $s+r-t$, but not in round $s+r$ as that would need an original ttl of $r+t$. Furthermore a state σ' sent in round $s+r-1+t$ (for a positive t) can be received earliest in round $s+r+t$. Thus only states sent in rounds s to $s+r-1$ can be received by a node $p_j \in G_i^r$ in round $s+r$.

Now consider any node $p_j \in G_i^r \setminus \{p_i\}$. Let $\hat{r} \in [1, r]$ be the smallest number of hops between p_i and p_j. By the Lemma statement node p_i sends out state σ in round $s + r - \hat{r}$. That message is forwarded one step each round and $\hat{r}$ rounds later in round $s + r$ it reaches node p_j. ■

We now look how the addition of cluster heads work while temporarily disregarding the escaping mechanism. In this setting we will show that within a finite number of rounds we will have $|C_i^r| \geq k_i$ for any node p_i. Later on we will lift this restriction and show that $|C_i^r| \geq k_i$ will still hold even when regarding the more general case.

Lemma 3: Let round s and all following rounds be legal. Assume that the state of a node can never be ESCAPING, $estate$ is always SLEEP and that lines 8-18 are not going to be executed. With these assumption after round $s + 2r + t$ for a non-negative t any node p_i will have k_i cluster heads within r hops.

Proof: The limiting assumptions leave only one way for the state to change, namely by the execution of line 70 where state is set to HEAD.

From Lemma 1 we know that in round $s + r$, at the latest, node p_i will have all nodes in G_i^r in S_i. We also know that $|G_i^r| \geq k_i$. Let's look at round $s+r$. At line 20, $heads_i$ might already contain nodes. We have the one case where $|heads_i| \geq k \geq k_i$ already and one case where $|heads_i| < k$. In the second case lines 21-23 will be executed. Out of the set A of nodes in G_i^r that are not in $heads_i$, the smallest $\min(|A|, k - |heads_i|)$ nodes will be added to $heads_i$ in line 23. Thus after execution of line 23 $heads_i$ will contain $\min(|G_i^r|, k) = k_i$ nodes and at line 25 $|heads_i| \geq k_i$.

For each node $p_j \in heads_i$ either a join message with a ttl of r is sent out (at line 27, when $j \neq i$) or the state is set to HEAD directly (at line 29, when $j = i$). For the nodes $p_j \neq p_i$ the join messages are forwarded (line 72) to all nodes in G_i^r within r hops in r rounds (in a similar fashion as forwarded state as discussed in the proof of Lemma 1).

Each node $p_j \in heads_i$ thus gets a join addressed to itself at the latest in round $s + 2r$ and it will become a cluster head by setting its state to HEAD at line 70. Thus after round $s+2r$ any node p_i will have k_i cluster heads within r hops. ■

Now we consider the full escape mechanisms and show that a node that receive joins become a cluster head.

Lemma 4: Consider a node p_i that receives a join during the receipt phase of the legal round z that follows the legal round $z-1$. Then node p_i is a cluster head at the end of round z. Furthermore, if node p_i is a cluster at the end of round $z-1$ then it is a cluster head throughout the entire round z.

Proof: Let σ be $state_i$ and e be $estate_i$ at the reception of a join from any node in a legal round z which follows a legal round $z - 1$. We begin by showing that the only thing that can happen with $state_i$ during the receipt phase of round z is for it to either change to HEAD or to stay HEAD or ESCAPING. We have four different cases for different e and σ.

Case 1 $e \in \{\text{INIT}, \text{FLOOD}\} \wedge \sigma = \text{SLAVE}$: This cannot happen as (1) node p_i in the previous round (the legal round $z - 1$) could not have $state_i = \text{SLAVE}$ without having executed line 17 that sets $estate_i$ to SLEEP and (2) there is no way for $estate_i$ to change during the receipt phase of a round.

Case 2 $e \in \{\text{INIT}, \text{FLOOD}\} \wedge \sigma = \text{HEAD}$: No change to $state_i$, that remains HEAD.

Case 3 $e \in \{\text{INIT}, \text{FLOOD}\} \wedge \sigma = \text{ESCAPING}$: No change to $state_i$, that remains ESCAPING.

Case 4 $e \notin \{\text{INIT}, \text{FLOOD}\}$: Here $state_i$ is set to HEAD.

Furthermore, the only way for $state_i$ to be ESCAPING at the start of the step phase of round z is if $e \in \{\text{INIT}, \text{FLOOD}\}$. In that case $estate_i$ must have been set to FLOOD after line 10 in round $z - 1$ from which it follows that $estate_i \in \{\text{FLOOD}, \text{HOPE}\}$ after execution of line 10 in round z. Thus, the condition in line 14 does not hold in round z and line 15, the only line that can set $state_i$ to SLAVE, is not executed. Therefore, node p_i is a cluster head at the end of round z and if it were a cluster head at the beginning of round z it was so throughout the round. ■

In the following Lemma we show that a node that is continuously wanted as a cluster head eventually becomes one.

Lemma 5: Let s and all following rounds be legal rounds and assume a node p_j wants a node $p_i \in G_j^r$ to be cluster head as soon as it knows about it and is never willing to let it escape. In other words (1) if $p_i \notin heads_j$ after line 7 the condition in line 21 would always hold and $p_i \in A$ after executing line 22 and (2) the condition in line 50 would always hold.

Then node p_i will be a cluster head after round $y \leq s + 2r$ and throughout all following rounds.

Proof: Let $\hat{r}$ be the number of hops between p_i and p_j and let round x be the first round $\geq s$ in which node p_j receives a state from p_i. We know that $s \leq x \leq s + \hat{r}$. Furthermore, let y be the round in which p_i gets the join from p_j that was sent in round x. We know that $y = x + \hat{r}$ and thus $s + 1 \leq y \leq$

$s + 2\hat{r}$ and thus both round $x - 1$ and x are legal. According to Lemma 4, p_i will be a cluster head at the end of round y.

According to the assumptions, $p_i \in heads_j$ at line 25 in every round $\geq x$ and thus p_j sends a join to p_i in every such round. This means that p_i will receive a join in every round $\geq y$, and thus, by Lemma 4, be a cluster head in the step phase of round y and throughout the following rounds. ■

Now we can show that within $2r + 1$ legal rounds from an arbitrary configuration all nodes p_i have at least k_i cluster heads and that the set of cluster heads in the network can only stay the same or shrink from that point on.

Theorem 1: Let round s and all following rounds be legal. Then any node p_j will have k_j cluster heads within r hops in the step phase of round $s + 2r$ and throughout any following rounds. Moreover, a node that is not in H_x in a round $x \geq s + 2r$ can not be in H_{x+t} for a non-negative t and consequently $|H_{x+t}| \leq |H_x|$.

Proof:

From Lemma 3 we have seen that as long as the escape mechanism does not allow nodes to change its state to SLAVE after being a cluster head, any node p_j will have k_j cluster heads within r hops in the step phase of round $s + 2r$ and throughout any following rounds.

Furthermore, from Lemma 5 and its proof we have seen that as long as a node p_j wants to have node p_i as a cluster head p_i will remain a cluster head. Now we will look in what situations p_j does not want p_i as a cluster head even though it did at some earlier point in time.

If $|heads_j| < k$ at line 20 in a round s, then node p_j finds up to $k - |heads_j|$ nodes in $\{p_i : < p_i, \cdot > \in S_j\} \setminus heads_j$ and sends a join to them in a round s. Assume $p_i \in G_j^r$ is one of the newly picked nodes in A after executing line 22 in round s. We call this set A in round s for $\widehat{A}$. Node p_i does not get the join until round $s + \hat{r}$ where $\hat{r}$ is the number of hops between nodes p_i and p_j.

As we saw in the proof of Lemma 4, if $state_i =$ ESCAPING $\wedge\, estate_i \in \{\text{INIT}, \text{FLOOD}\}$ does not hold in round $s+\hat{r}$ then p_i will send out HEAD. That will reach p_j in round $s+2\hat{r}$ and consequently $p_i \in heads_j$ in round $s+2\hat{r}$. If $state_i =$ ESCAPING and $estate_i \in \{\text{INIT}, \text{FLOOD}\}$ in round $s + \hat{r}$, node p_i will not send out HEAD in that round and p_j might not get HEAD from p_i in round $s+2\hat{r}$. If p_j got HEAD from some other node $p_l \in G_j^r$ in a round $x \in [s + 1, s + 2\hat{r}]$ node p_j might not want p_i as a cluster head any more. Node p_j will not send join to p_i in round x if (1) $|heads_j| \geq k$ at line 20 or (2) p_i has received HEAD from enough nodes not in $\widehat{A}$ so that p_i is not among the smallest nodes picked out in line 22 in round $s + 2\hat{r}$. On the other hand if none of these cases hold p_j will continue to send joins to p_i and by Lemma 5 node p_j will remain a cluster head.

The second way for a node $p_i \in G_j^r$ to be SLAVE even though it earlier were in $heads_j$ is to escape using the escape mechanism. In other words in some round z node p_i initiates an escape attempt. When receiving different states for a node, HEAD takes precedence over ESCAPING that takes precedence over SLAVE (lines 55-60). This combined with Lemma 2 means that in some round $y \in [z + 1, z + r]$ node p_i get the state ESCAPING from p_i and that in the previous round $y - 1$ p_j got HEAD from p_j.

Node p_j will have $p_i \in heads_j$ after executing line 7 in round $y - 1$ as p_j receives HEAD from p_i in round $y - 1$. Thus $p_i \in heads_j$ at line 47 in round y when p_j receives ESCAPING from p_i. If $|heads_j| \leq k$ at that point then p_j will interpret the state as HEAD for all purposes other than forwarding the message (lines 50-51). Thus $p_i \in heads_j$ after executing line 7 in round y as well, and p_j will send a join. By Lemma 5 and its proof, node p_i will remain cluster head in that case. If on the other hand $|heads_j| > k$ for node p_j at line 47 in round y then p_j removes p_i from heads and will consequently not send any join in round y. When p_j gets ESCAPING from p_i in the coming rounds $y + 1, y + 2, \ldots$, then p_i will not be in $heads_j$ and thus no joins will be sent in those rounds either. If node p_j receives ESCAPING for more than one node $p_l \in heads_j$ in rounds $y, y+1, \ldots$ then p_j will let them go in first come first served fashion. When node p_j decides to let a node p_l go it is immediately removed from $heads_j$ in line 53. Thus, node p_i will not let so many nodes go that $|heads_j| \geq k$ would not be fulfilled (if $k_j < k$ no node is ever allowed to go).

Finally, a node $p_i \notin G_j^r$ might be in $heads_i$ in a round $z \in [s, s + r - 1]$ but by Lemma 1 such a node is not in S_j in round $s + r$ and thus node p_i will pick some other node instead of such a p_j to send join to in round $s+r$ if not earlier.

So any node p_j will have k_j cluster heads within r hops in the step phase of round $s + 2r$ and throughout any following rounds. Therefore in any of these rounds no node will fulfill the condition in line 20. Hence, no node p_i that is not a cluster head at the beginning of the state phase of round $s + 2r$ can be picked by any node p_j in line 22. Therefore no such node p_i can become a cluster head in the state phase of round of round $s + 2r$ or thereafter.

Thus a node that is not in set of cluster heads in the entire network at round x, H_x, for a round $x \geq s + 2r$ can never be in H_{x+t} for a non-negative t. Moreover, $|H_{x+t}| \leq |H_x|$ for any $x \geq s + 2r$ and any non-negative t. ■

B. Convergence to a Local Minimum

In this section we show that the set of cluster heads converges to a local minimum. We show that a cluster head node that is not needed can escape the cluster head responsibility if not interfered by other escape attempts (Lemma 6). We show that an unneeded cluster head node escapes within $O(gr)$ rounds with high probability under assumptions of synchronized timers (Lemma 7). Finally, in Theorem 2, we show that with high probability the entire network reaches a local minimum within $O(gr \log n)$ rounds. We begin by looking at the escape of an uninterfered node.

Lemma 6: Consider a round s for which all rounds from $s - 2r - 1$ and forward are legal. Assume that node p_i initiates an escape attempt in round s and assume that in rounds $[s, s + r]$

all nodes $p_j \in G_i^r$ have $|C_j^r| > k$. If no other node $p_l \in G_i^{2r}$ than node p_i initiates an escape attempt in any round $\in [s-2r-1, s+r-1]$ then node p_i will set $state_i$ to SLAVE in round $s+2r+1$ and have $state_i =$ SLAVE throughout any round $s+2r+1+t$ for a positive t.

Proof: Assume that a node p_l initiates an escape attempt in round x. In round $x+2r$ node p_l will set $estate_l$ to HOPE. In all rounds in $[x, x+2r]$ node p_l will send out $state_l =$ ESCAPING. If node p_l gets a join to itself in the receipt phase of round $x+2r+1$ it sets $state_i$ to HEAD in line 70. Otherwise p_l sets $state_l$ to SLAVE in line 15 in round $x+2r+1$. Let σ be the $state_i$ that is sent out by p_l in round $x+2r+1$. We know that $\sigma \neq$ ESCAPING. We assume that node p_l does not initiate any more escape attempts in the time span we are looking at. Therefore node p_l sends out σ in the rounds in $[x+2r+1, x+3r]$. By Lemma 2, in round $x+3r+1$ all nodes $p_{j'} \in G_l^r \setminus \{p_l\}$ receives σ and only σ for p_l in the receipt phase. Therefore, either all nodes $p_j \in G_l^r$ (including p_l) have $p_l \in heads_j$ (if $\sigma =$ HEAD) or none of them have $p_l \in heads_j$ (if $\sigma =$ SLAVE) after executing line 7 in the step phase of round $x+3r+1$. This continues to hold in the receive phase of round $x+3r+2$. Thus in round $x+3r+1+t$, for a positive t, no node $p_j \in G_l^r$ can have $p_l \in C_j^r$ without having $p_l \in heads_j$.

Now if node p_i initiates an escape attempt in round s, by Lemma 2, all nodes $p_j \in G_i^r$ will receive ESCAPING and only ESCAPING for node p_i in round $s+r$. As we saw above no node p_l initiating an escape attempt in a round $\leq s-2r-2$ can in round $s+r$ be in C_j^r, for a node $p_j \in G_l^r$, without being in $heads_j$. By the Lemma assumptions, no node $p_l \in G_i^{2r}$ makes an escape attempt in a round in $[s-2r-1, s+r-1]$. In addition, consider a node p_l' that initiates an escape attempt in a round $s+r-1+t$ for a positive t. A node $p_j \in G_i^r$ can only receive ESCAPING from that escape attempt in rounds $\geq s+r+t$. Therefore a node $p_j \in G_i^r$ will in round $s+r$ receive ESCAPING for node p_i but not for any other node.

In rounds $[s, s+r]$ all nodes $p_j \in G_i^r \setminus \{p_i\}$ have $|C_j^r| > k$. Therefore, when p_j receives ESCAPING for p_i in round $s+r$ either (1) $p_i \notin heads_j$ because p_j received ESCAPING and had $|heads_j| > k$ in some round in $[s+1, s+r-1]$ or (2) $p_i \in heads_j$ and $|heads_j| > k$ in which case p_j removes p_i from $heads_j$ at line 53. Thus in the step phase of rounds in $[s+r, s+2r]$ no node p_j sends a join to p_i. Therefore, in the round $s+2r+1$ no join is received by p_i and therefore p_i sets $state_i$ to SLAVE in round $s+2r+1$. There is no round $\geq s$ in which p_i sends out HEAD and, by Theorem 1, no node will need to add new nodes as cluster heads in any round $\geq s$. Hence, node p_i will have $state_i =$ SLAVE in the step phase of round $s+2r+1$ and throughout any round $s+2r+1+t$ for a positive t. ■

Definition 4: We say that the timers of the nodes in the network are synchronized if $timer_i = timer_j$ for all pair of nodes $p_i, p_j \in \mathcal{P}$ for all legal rounds.

Under the added assumption of synchronized timers we show that an unneeded cluster head node either escapes within $O(gr)$ rounds, unless it becomes needed due to other escaped cluster heads.

Lemma 7: Let round s and all following rounds be legal. Furthermore, let $g = \max_j |G_j^{2r}|$ be a bound on the number of nodes within $2r$ hops. Consider a node p_i that is a cluster head in any round $\geq s$. Assume that $|C_j^r| > k$ holds for all nodes $p_j \in G_i^r$ from round $s+2r$ and as long as p_i remains a cluster head. If the timers of all nodes in the network are synchronized and $T = 8gr$, node p_i will be SLAVE in any round $s+2r+8(\beta+1)gr-2+t$ for a non-negative t with probability at least $1-2^{-\beta}$.

Proof: From Theorem 1 we know that from round $s+2r$ nodes can only go from being cluster heads to being slaves. Consider a cluster head node p_i. Let x_i^0 be the first round $\geq s+2r$ in which $timer_i = 0$ at line 8. As long as node p_i remain a cluster head it will execute line 35 every round $x_i^t = x_i^0 + tT$, for a non-negative t and a given T, and schedule an escape attempt in the *period* $\Pi_i^t = [x_i + tT, x_i + (t+1)T - 1]$. Node p_i picks one of the first $T-2r-1$ rounds in the period, uniformly at random and independently from any other random choice, to initiate an escape attempt in. Thus the probability that node p_i initiates an escape attempt in any given round is $\leq 1/(T-2r-1)$.

Now consider a period Π_i^t in which p_i initiates an escape attempt. Let D_i^t be the set of rounds $[x_i^t - 2r - 1, x_i^t + r - 1]$. The number of nodes that could be cluster heads in G_i^{2r} is bounded by g. If $F_{i,l}^t$ is the event that a node $p_l \in G^{2r}$ initiates an escape attempt in any round in D_i^t then $P[F_{i,l}^t] \leq (3r+1)/(T-2r-1) =: \rho$. Let A_i^t be the event that none of the nodes in G_i^{2r} initiate an escape attempt in a round in D_i^t. We say that A_i^t is the event that node p_i gets an *uninterfered escape attempt* in period Π_i^t. Then we get

$$\begin{aligned} P[A_i^t] &\geq (1-\rho)^{g-1} = [\mu := \frac{1}{\rho}] \\ &= \left(\left(1-\frac{1}{\mu}\right)^{\mu-1}\right)^{(g-1)/(\mu-1)} \\ &> \left(\frac{1}{e}\right)^{(g-1)/(\mu-1)} = \exp\left(-\frac{g-1}{\mu-1}\right) \\ &= \exp\left(-\frac{g-1}{\frac{T-2r-1}{3r+1}-1}\right). \end{aligned} \tag{1}$$

We can simplify this, using the fact that $r \geq 1$, to get that for $T = 8gr$ we have $P[A_i^t] > 1/2$.

According to the Lemma assumption the timers of all nodes in the network are synchronized. Thus we have a global $x^t = x_i^t$ and $\Pi^t = \Pi_i^t$ holding for all nodes $p_i \in \mathcal{P}$. Consider a period starting in round z. The earliest in a period a node p_i can initiate an escape attempt is in round z when $estart_i = 0$. The latest a node p_j could initiate an escape attempt in the period starting in $z-T$ is in round $(z-T)+(T-2r-2) = z-2r-2$. Thus by Lemma 6 an escape attempt initiated in an earlier period cannot affect an escape attempt in this period. The latest in a period a node p_i can initiate an escape attempt is in round $z+T-2r-2$ when $estart_i = T-2r-2$. However $z+T-2r-2+r-1 < T$ and therefore, by Lemma 6, no escape attempt in a later period could affect this period. This together with the fact that the random choices in different executions of the line 35 are all mutually independent would make what happens in different rounds mutually independent. However if a node p_i becomes SLAVE in a round t it is not doing an escape attempt in round $t+1$ which only increases the probability for A_j^{t+1} for another node p_j. Therefore, by assuming independence the calculated lower bound on the probability of an undisturbed escape attempt gets worse.

Consider the β periods Π^0 to $\Pi^{\beta-1}$ and let $A_i = \bigcup_{t=0}^{\beta-1} A_i^t$. Thus with the assumption of period independence that gives us a worse bound we get

$$\begin{aligned} P[A_i] =& P\left[\bigcup_{t=0}^{\beta-1} A_i^t\right] = 1 - P\left[\bigcap_{t=0}^{\beta-1} \bar{A}_i^t\right] = 1 - \prod_{t=0}^{\beta-1} P[A_i] \\ >& 1 - \prod_{t=0}^{\beta-1} \frac{1}{2} = 1 - 2^{-\beta}. \end{aligned} \tag{2}$$

The latest period Π^0 could start is $s+2r+T-1$ in which case $\Pi^{\beta-1}$ ends in round $s+2r+T-1+\beta T-1 = s+2r+8(\beta+1)gr-2$. ∎

From Theorem 1 we got that all nodes p_i have at least k_i cluster heads within r hops in $2r$ rounds after an arbitrary configuration.

Assuming that the timers of all nodes in the network are synchronized, we show that with at least probability $1-2^{-\alpha}$ the set of cluster heads in the network stabilizes to a local minimum within $O((\alpha+\log n)gr)$ rounds.

Theorem 2: Let round s and all following rounds be legal. Consider round $f = s+2r+8(\alpha+\log n+1)gr-2$, where n is the number of nodes in the network. Assume that the timers of all nodes in the network are synchronized. Then, with at least probability $1-2^{-\alpha}$, in round f there will be no cluster head node p_i in the network for which $\min_{p_j \in G_i} |C_j^r| > k$ holds and $H_{f+t} = H_f$ holds for any positive t.

Proof: We use the notations Π^t, x^t and A_i and the concept of uninterfered escape attempts from the proof of Lemma 7.

Let $\beta = \alpha + \log n$, where $n = |\mathcal{P}|$. Let A be the event all nodes in the network get at least one uninterfered escape attempt in the periods Π^0 to $\Pi^{\beta-1}$. We get that

$$\begin{aligned} P[A] =& 1 - P[\bar{A}] = 1 - P[\bigcup_{p_i \in \mathcal{P}} \bar{A}_i] \geq \text{[Boole's inequality]} \\ \geq& 1 - \sum_{p_i \in \mathcal{P}} P[\bar{A}_i] \geq \text{[Lemma 7]} \geq 1 - \sum_{p_i \in \mathcal{P}} 2^{-\beta} \\ =& 1 - n2^{-\beta} = 1 - 2^{\log n - \beta} = 1 - 2^{-\alpha}. \end{aligned} \tag{3}$$

Thus by the proof of Lemma 7 all nodes in the network gets an uninterfered escape attempt with at least probability $1-2^{-\alpha}$ by the round $f = s+2r+8(\alpha+\log n+1)gr-2$. Together with Lemma 7 This concludes that with high probability all nodes p_i for which $|C_i^r| > k$ holds at their uninterfered escape attempt will have set $state_i$ to SLAVE by round f. From this follows that at round f there is no node for which $\min_{p_j \in G_i} |C_j^r| > k$ holds. Hence, by Lemma 1, no node p_i that is cluster head in round f can ever set $state_i$ to SLAVE in a round $\geq f$ and $H_{f+t} = H_f$ holds for any positive t. ∎

C. Message Complexity

We now show the message complexity for the algorithm.

Theorem 3: Let round s and all following rounds be legal. Then the size, in bits, of the message sent by a node p_i in any round $\geq s+r$ is in $O(|G_i| \cdot (\log n + \log r))$.

Proof: By Lemma 1 we have that for any node p_i, $\{p_j :< p_j, \cdot > \in S_i\} = G_i^r$ holds in the step phase of round $s+r$ and throughout rounds $s+r+1+t$ for any non-negative t. Following the proof steps of that Lemma, we can conclude that the only nodes represented in $stateset_i$ and $joinset_i$ are the ones in G_i^r.

The only message a node p_i transmits in a round, is transmitted in line 91. Before that, $stateset_i$ is shrunk in line 89 so that, for every possible pair of node id j and state s, only the maximum ttl is kept. Similarly, $joinset_i$ is shrunk in line 90, so that for every possible node id j, only the maximum ttl is kept.

Each $stateset$ entry contains a node id which is encoded in $\log n$ bits, one of three possible states that is encoded in 2 bits and a ttl value that is encoded in $\log r$ bits. Each $joinset$ entry contains a node id and a ttl value. Thus the number of bits transmitted per node in G_i^r is in $O(\log n + \log r)$. Therefore, the size, in bits, of the message sent by a node p_i in any round $\geq s+r$ is in $O(|G_i| \cdot (\log n + \log r))$. ∎

V. Discussion

We prove convergence within $O(gr \log n)$ rounds with high probability under the assumption of independent rounds (apart from the obvious dependence that nodes that escape are out of the contention for uninterfered escape attempts). To see if the timers really need to be synchronized to achieve this performance we did simulations of the algorithm for various settings of k and r and network densities. We placed n nodes with a communication radius of 1 uniformly at random in a 5 by 5 rectangular area, with varying n for different experiments.

From our experiments we concluded that when $T = 8gr$ we get a much faster convergence than the upper bound we have proved. Setting T even lower decrease the convergence time even further.

A representative picture of the general results can be seen in Fig. 3. The experiments show that when all $timer_j$ are independently and uniformly distributed in $[0, T-1]$ at beginning of the experiment the convergence time is not far from what it is in the case with synchronized timers. We also see that if the nodes starts up with random information in their variables the convergence time is faster than for an initialized

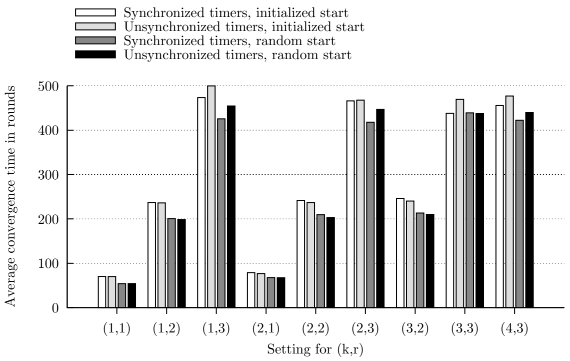

Fig. 3. Simulation results of the algorithm indicating that synchronization of timers is not needed. Here $T = gr/2$, $n = 39$.

start where each node p_i does not know G_i and sets itself as cluster head in the first round. To conclude we can with good margin use the result of convergence within $O(gr \log n)$ rounds from Theorem 2 for the unsynchronized setting.

For the rest of the experiments we did a small change to the algorithm. A node that is added to the network do not elect new cluster head nodes for the first r full rounds. It performs all parts of the algorithm except that the condition in line 20 is always regarded as false in those rounds and lines 21 to 23 are not executed. It is trivial to make this mechanism self-stabilizing.

In Fig. 4 we can see the convergence behavior of the set of cluster towards local minima over time. The same trend can be seen for other choices of k and n.

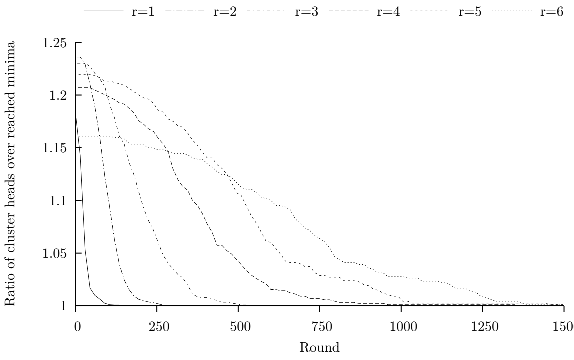

Fig. 4. Cluster head overhead over time as a ratio between number of cluster heads in given round over the eventually reached local minimum number of cluster heads. Here $T = gr$, $k = 2$ and $n = 39$.

We have performed experiments to investigate the range of possible results regarding the (k,r)-dominating sets generated by our algorithm on random graphs. We compare the global minima with results given by our algorithm and with the worst (i.e., largest) possible local minima in Fig. 5. The results of our algorithm is in general placed in the middle between the global minima and the worst possible local minima. We can also see that even the worst possible local minima are still quite close to the global minima. Thus even if our algorithm is "unlucky" it provides a good result. This trend holds true for other choices of n and k as well.

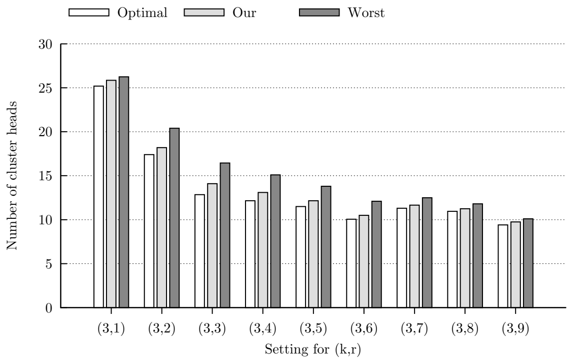

Fig. 5. Comparison between global optima, results of our algorithm and worst possible local minima for $n = 31$

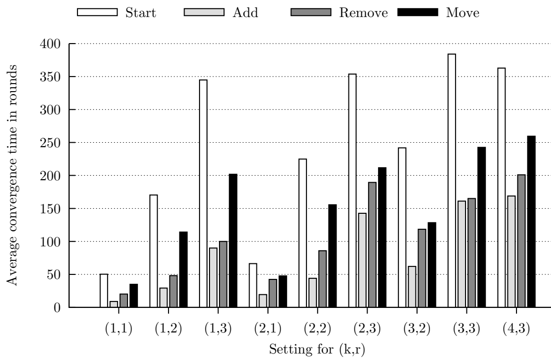

Fig. 6. Convergence times from a fresh start, after 5% node additions, after 5% node removes and after 5% node moves. Here $T = gr$ and $n = 39$.

We also performed experiments on recovery from small changes to the topology from a converged state. The convergence times from a newly started network ("Start") is compared in Fig. 6 with the convergence times after a change to a initially converged network. We investigate 5% added nodes ("Add"), 5% removed nodes ("Remove") or 5% moved nodes ("Move"). We achieve similar results for other choices of n as well.

We can see that the least obtrusive change to the topology is added nodes. The chance is good that a new node ends up in a position in the network were there already is enough or close to enough cluster heads already. Remove is more expensive than add. Cluster heads could be among the nodes that are removed. Additionally removed nodes can also have been used as links between nodes and their cluster heads. A move is like both a remove and an add (without the rounds of abstaining from electing new cluster heads). Therefore, it is anticipated that this case converges slower than the ones with only adds or only removes.

The flooding of messages makes sure that if there exist multiple paths of at most length r between a node p_i and a node p_j then joins and state updates will traverse all possible paths. This can give us higher fault tolerance if there are communication disturbances on some links (i.e. between some immediate neighbors) and also higher availability for nodes to reach their cluster heads.

The multiple paths can also give applications higher security if some nodes in the network can be compromised. If there is at least one path of at most r hops between a node p_i and a node p_j that is not passing through any compromised nodes then the flooding makes sure that node p_i and p_j gets to know about each other. Moreover, if p_j wants p_i to be cluster head then the compromised nodes cannot stop that. If nodes add information to the messages about the paths they have taken during message forwarding then the nodes get to know about the multiple paths. With this knowledge they can in an application layer use as diverse paths as possible to communicate with their cluster heads. Thus even if a compromised node is on the path to one cluster head and drops messages or do other malicious behavior there can be other cluster heads for where there is no compromised nodes on the chosen paths.

Consider a compromised node p_c that can lie and not follow protocol. First assume that p_c cannot introduce node id:s that does not exist (Sybil attacks, [12]) or node id:s for nodes that are not within G_c^r (wormhole attacks, [5]) and that p_c cannot do denial of service attacks. Then p_c can make any or all nodes within G_c^r become and stay cluster heads by sending joins to them or having them repeatedly go on and off cluster head duty over time by alternating between sending joins and letting the node escape. Consider a node p_i that is a cluster head and has a path to a node p_j of length $\leq r$ hops that does not pass through p_c. In this situation p_c can not give the false impression that p_i is not a cluster head as HEAD takes precedence over ESCAPING that takes precedence over SLAVE at message receipt. If p_c on the other hand is in a bottleneck between nodes without any other paths between them then it can lie about a node p_l being a cluster heads and refuse to forward any joins to p_l. Now if we assume that p_c is not restricted in what id:s it can include in false messages it can convince a node p_l that nodes not in G_l^r are cluster heads. In the worst case it can eventually make p_l rely exclusively on non-existent cluster heads with paths that all go through p_c. In any case the influence by a compromised node p_c is contained within G_c^{2r} as the maximum ttl of a message is r and is enforced at message receipt.

The flooding of messages might make the algorithm too expensive in some sensor networks with limited battery power. If that is the case the algorithm might be run on an overlay network for which the flooding becomes much cheaper. For instance a self-stabilizing spanning tree algorithm like the one in [2] might be used to set up this overlay network. This on the other hand effectively removes all the pros of having multiple paths so it is a trade off between redundant paths and message costs.

VI. Conclusions

We have presented the first self-stabilizing (k, r)-clustering algorithm for ad-hoc networks. A deterministic mechanism guarantees that all nodes, if possible for the given topology, have k cluster heads within r hops. A randomized mechanism lets the set of cluster heads stabilize to a local minimum. We have shown, under the extra assumption of synchronized timers, that the set of cluster heads converges, with high probability, to a local minimum within $O(gr \log n)$ rounds, where g is an upper bound on number of nodes within $2r$ hops, and n is the size of the network. With simulations we have shown that even without this extra assumption the system converges much faster than the proved bounds and that g does not have to be known very accurately. We have also discussed how the algorithm can help us with fault tolerance and security and that the algorithm can be run on an overlay network, e.g. a spanning tree, if message costs needs to be reduced.

References

[1] Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.*, 30(14-15):2826–2841, 2007.

[2] Sudhanshu Aggarwal and Shay Kutten. Time optimal self-stabilizing spanning tree algorithms. In *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 400–410, London, UK, 1993. Springer-Verlag.

[3] Eddy Caron, Ajoy Kumar Datta, Benjamin Depardon, and Lawrence L. Larmore. A self-stabilizing k-clustering algorithm using an arbitrary metric. In *Euro-Par*, pages 602–614, 2009.

[4] Yuanzhu Peter Chen, Arthur L. Liestman, and Jiangchuan Liu. *Clustering Algorithms for Ad Hoc Wireless Networks*, volume 2, chapter 7, pages 154–164. Nova Science Publishers, 2004.

[5] Yih chun Hu, Adrian Perrig, and David B. Johnson. Wormhole detection in wireless ad hoc networks. Technical report, Rice University, Department of Computer Science, 2002.

[6] Shlomi Dolev. *Self-Stabilization*. MIT Press, March 2000.

[7] Shlomi Dolev and Nir Tzachar. Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. *Theor. Comput. Sci.*, 410(6-7):514–532, 2009.

[8] Yongsheng Fu, Xinyu Wang, and Shanping Li. Construction k-dominating set with multiple relaying technique in wireless mobile ad hoc networks. In *CMC '09: Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing*, pages 42–46, Washington, DC, USA, 2009. IEEE Computer Society.

[9] Colette Johnen and Le Huy Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theor. Comput. Sci.*, 410(6-7):581–594, 2009.

[10] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[11] Andreas Larsson and Philippas Tsigas. Self-stabilizing (k,r)-clustering in wireless ad-hoc networks with multiple paths. In *OPODIS'10, 14th International Conference On Principles Of Distributed Systems*, Tozeur, Tunisia, December 2010.

[12] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks: analysis & defenses. In *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 259–268, New York, NY, USA, 2004. ACM.

[13] Marco Aurélio Spohn and J. J. Garcia-Luna-Aceves. Bounded-distance multi-clusterhead formation in wireless ad hoc networks. *Ad Hoc Netw.*, 5(4):504–530, 2007.

[14] Kun Sun, Pai Peng, Peng Ning, and Cliff Wang. Secure distributed cluster formation in wireless sensor networks. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 131–140, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Yiwei Wu and Yingshu Li. Construction algorithms for k-connected m-dominating sets in wireless sensor networks. In *MobiHoc '08: Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 83–90, New York, NY, USA, 2008. ACM.