# Self-stabilizing (k,r)-Clustering in Wireless Ad-hoc Networks with Multiple Paths*

Andreas Larsson and Philippas Tsigas

Chalmers University of Technology and Göteborg University
{larandr,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 in many ways like forming backbones, facilitating routing, improving scaling, aggregating information and saving power. A (k,r)-clustering assigns cluster heads so that exists $k$ cluster heads within $r$ communication hops for all nodes in the network while trying to minimize the total number of cluster heads. We present the first self-stabilizing distributed (k,r)-clustering algorithm. The algorithm uses synchronous communication rounds and uses multiple paths to different cluster heads for providing improved security, availability and fault tolerance. From any starting configuration the algorithm quickly assigns enough cluster heads and stabilizes towards a local minimum using a randomized scheme.

**Keywords:** Clustering, Self-Stabilization, Ad-hoc Networks, (k,r)-dominating sets.

## 1 Introduction

An algorithm for clustering nodes together in an ad-hoc network serves an important role. It can be used for back bone formation, routing, data aggregation, improve scaling and energy saving by taking turns. Clustering is a well studied problem. Due to space constraints we point to the survey of the area with regard to wireless ad-hoc networks by Chen et al. in [2] for references to the area in general. 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, if possible, at least $k$ cluster heads within $r$ communication hops away. Assuming that the network topology allows $k$ cluster heads for each

* The research leading to these results has received funding from the Swedish Civil Contingencies Agency (MSB) and has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257007.

C. Lu, T. Masuzawa, and M. Mosbah (Eds.): OPODIS 2010, LNCS 6490, pp. 79–82, 2010.

node, the set of cluster heads forms a *total (k,r)-dominating set* where the nodes in the set, the cluster heads, also need to have $k$ nodes in the set within $r$ hops. The dominating set problem is well known to be NP-hard in general. Therefore, instead of looking for a global minimum, approximate algorithms are proposed.

Starting from an arbitrary state, self stabilizing algorithms let a system stabilize to, and stay in, a consistent state [3]. There is a multitude of existing clustering algorithms for ad-hoc networks, of which a number is self-stabilizing. A self-stabilizing (1,1)-clustering algorithm that converges fast is presented in [5]. A lot of organizational problems is tackled in a self-stabilizing manner and a self-stabilizing (1,r)-clustering algorithm is presented in [4]. Weighted graphs is taken into account in the self-stabilizing (1,r)-clustering in [1]. Algorithms for the full (k,r)-clustering problem is presented in [7] and [9], but neither is self-stabilizing. The algorithm presented in [8] groups nodes together without assigning cluster heads. It considers malicious nodes inside the network, but is not self-stabilizing.

### 1.1 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 (if the topology permits it) 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 $O(g \cdot r \cdot \log n)$ 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.

Our contribution is presented as follows. In section 2 we introduce the system settings. Section 3 describes the algorithm. We discuss multiple paths, proofs and experiments, and conclude in Section 4.

## 2 System Settings

We assume a static network. Changes in the topology are seen as transient faults. 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. For a node $p_i$ in the network, we denote the nodes within one hop $N_i$, i.e. the nodes to which it can directly send messages. The nodes within $r$ hops from $p_i$, including $p_i$ itself, is denoted $G_i^r$. We assume undirected communication graphs, i.e. $p_i \in N_j$ iff $p_j \in N_i$. The system is synchronous and progresses in rounds. Each round has two phases. In the receipt phase each node $p_i$ receives messages from all nodes in $N_i$. In the step phase each node $p_i$ broadcasts a message to all nodes $p_j \in N_i$ and that is received reliably in the next receipt phase.

## 3 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 $k_i = \min(k, |G_i^r|)$ cluster heads within

```
1  on step phase:
2    increase timer modulo T
3    schedule escape attempt uniformly at random from [0,T-2r-2 ] if timer = 0
4    if state = HEAD and |heads|> k and it is time to attempt escape then
5      state ← ESCAPING
6      remove i from heads
7    if state = ESCAPING for 2r+1 rounds (all nodes in G_i^r had the chance to veto escape) then
8      state ← SLAVE
9    if |heads|< k then
10     add more nodes in G_i^r to heads, if there are any left to add
11   Broadcast state and heads with a TTL of r to direct neighbors, together with queued forwards
12
13 on receive jstate and jheads originating from node p_j
14   if i ∈ jheads and state was not set to ESCAPING within the last 2r rounds then
15     state ← HEAD
16     add i to heads
17   if j ∈ heads and jstate = ESCAPING and |heads| > k then
18     remove j from heads
19   else if jstate = HEAD then
20     add j to heads if not present
21   queue up a forward of jstate and jheads with one lower TTL if TTL is not already 1
```

**Fig. 1.** Simplified pseudocode of the clustering algorithm, for a node $p_i$ with id $i$

its $r$-hop neighborhood $G_i^r$. We achieve a local minimum, i.e. a set from which no cluster head can be removed without violating the aforementioned goal.

Space does not allow us to present the algorithm it its entirety. Full description with all details and everything that is needed to get self-stabilization can be found in [6]. The basic idea of the algorithm is presented in Figure 1. A node can have state HEAD or ESCAPING, in which it is a cluster head, or have state SLAVE, in which it is not a cluster head. In every round each node $p_i$ sends out its state and its cluster heads (line 11). Forwarding (line 21) with a time-to-live (TTL) mechanism makes sure that this information reaches all nodes in $G_i^r$ within $r$ rounds. A node $p_i$ that does not have $k$ cluster heads elects new ones by adding them to $heads$ (lines 9-10), which will be broadcasted as a join signal (line 11). A node that receives such a join signal becomes a cluster head if not already so (lines 14-16). A node that gets to know about a new cluster head adds it to $heads$ (lines 19-20).

This procedure might overshoot and establish too many cluster heads in the network, i.e. there is some cluster head node $p_j$ for which all nodes that has $p_j$ as a cluster head also have at least $k$ other cluster heads. To reach a local minimum, a cluster head node $p_i$ tries to escape at random points in time. This is done by setting the state to ESCAPING (lines 2-6) and ignoring incoming join signals for $2r$ rounds (line 14). If a node can allow a cluster head to go, it removes it from $heads$ (lines 17-18). Otherwise it disallows the escape attempt by continuing to send join signals. If no nodes disallow an escape attempt, the node can become SLAVE (line 7-8). If two cluster heads of which any one, but only one, can be allowed escape, tries to escape concurrently, both might fail. Therefore, the cluster heads repeatedly starts escape attempts at random points in time, within periods of a constant $T$ rounds, to resolve such conflicts. Eventually their attempts will not overlap.

## 4 Discussion and Conclusions

The full algorithm merges information received by different sources and with different TTL values. This reduces the total amount of information that needs to be transmitted. The flooding of messages makes sure that if there exist multiple paths within $r$ hops between a pair of nodes they will all be used. This provides higher fault tolerance. It can also give us higher security if nodes in the network can be compromised.

In [6] we prove that enough cluster heads are assigned within $O(r)$ rounds. Under the assumption of synchronized timers, we prove that the set of cluster heads convergences to a local minimum within $O(g \cdot r \cdot \log n)$ rounds, with high probability. We also show experimentally that synchronized timers are not needed and that $g$ does not have to be known very accurately. In addition, we present some preliminary experimental results on how far from optima our sets of cluster heads come, and provide more thorough discussions on security and discussions on complexity versus fault tolerance.

To conclude, 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.

## References

1. Caron, E., Datta, A.K., Depardon, B., Larmore, L.L.: A self-stabilizing k-clustering algorithm using an arbitrary metric. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 602–614. Springer, Heidelberg (2009)
2. Chen, Y.P., Liestman, A.L., Liu, J.: Clustering Algorithms for Ad Hoc Wireless Networks, vol. 2, ch. 7, pp. 154–164. Nova Science Publishers, Bombay (2004)
3. Dolev, S.: Self-Stabilization. MIT Press, Cambridge (March 2000)
4. Dolev, S., Tzachar, N.: Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. Theoretical Computer Science 410(6-7), 514–532 (2009)
5. Johnen, C., Nguyen, L.H.: Robust self-stabilizing weight-based clustering algorithm. Theoretical Computer Science 410(6-7), 581–594 (2009)
6. Larsson, A., Philippas, T.: Self-stabilizing (k, r)-clustering in wireless ad-hoc networks with multiple paths. Technical report, Chalmers University of Technology (2010)
7. Spohn, M.A., Garcia-Luna-Aceves, J.J.: Bounded-distance multi-clusterhead formation in wireless ad hoc networks. Ad Hoc Networks 5(4), 504–530 (2007)
8. Sun, K., Peng, P., Ning, P., Wang, C.: Secure distributed cluster formation in wireless sensor networks. In: ACSAC 2006, Washington, DC, USA, pp. 131–140. IEEE Computer Society, Los Alamitos (2006)
9. Wu, Y., Li, Y.: Construction algorithms for k-connected m-dominating sets in wireless sensor networks. In: MobiHoc 2008, pp. 83–90. ACM, New York (2008)