# Topology Cache aided Self-Organization for Ad-hoc Mobile Networks

Márton Legény, Borbála Katalin Benkő

***Abstract*—In this paper we examine an approach where, in order to reduce the communication overhead in a fully distributed, dynamic self-organizing network, nodes maintain a cache about the topology of their vicinity, and keep this data structure up-to-date throughout minor and rapid changes in the network. We plotted six complex caching strategies, defining when to establish the cache and when to update its content, and evaluated them with two different network dynamics profiles along a clustering and load-balancing scenario through simulation. Smart caching strategies beat the original non-cached approach both in speed and communication overhead. The difference between strategies is more visible with the moderate network changes, as the rapid change acts as a natural performance booster by bringing new potential load-balancing partners into the vicinity at no cost. The winner of the examined strategies uses a change sensitive cache update where the amount of change in the two-hop neighborhood is approximated by the observed change among direct neighbors.**

***Index Terms*—Ambient Intelligence, Clustering methods, Protocols, Topology.**

## I. Introduction

In Ambient Intelligence and Pervasive Computing scenarios [1], [2], multitudes of networked elements provide the user with services in a way that makes use of the distributed yet networked setting. Elements of the network themselves and so their services may be very diverse, and may also be highly dynamic in time. The ability of automatic self-organization is becoming an important requirement in these scenarios, as networked elements need to cooperate, share resources, communicate, or discover each other effectively, without help of an external 'network manager'.

While human network managers possess information about the state of the whole network, in case of self-organization, nodes need to rely on *locally available information* - even if it is incomplete and non-objective - when making their autonomous decisions.

Self-organization often uses *simple algorithms* that have *emergent* properties, i.e. the multitude of executions result in a complex, "intelligent" high-level behavior which is of a different quality than the simple building blocks themselves. Emergent algorithms are often inspired by biological, chemical or physical phenomena like swarms, insect colonies, human brain or the immune system. These paradigms were used in numerous ways to solve problems in computer networks, for example load balancing [3], [4], [5], [6]. Self-organization in overlay networks [7], [8] is also widely used for deploying distributed applications (mostly used in P2P data sharing systems) without the need for a supervising entity such as in [9], [10] and [11].

*Clustering* - in a self-organizing network - means that entities of the network search for other entities that meet a certain criterion (e.g. similarity in case of normal clustering or complementariness in inverse clustering) and establish connections with them in form of an overlay network. *Load balancing* is a use case of clustering when entities distribute their local load with members of the cluster. While the efficient creation of clusters is a prerequisite for good load balancing, it is not a sufficient condition: a load balancing algorithm must also answer the questions when and with which cluster member to share the load, and what information to use (collect, store and update) when making these decisions.

Our work focuses on biologically inspired, fully distributed self-organization algorithms for large overlay networks, with an emphasis on clustering and load balancing.

In this paper we describe an approach when the entities participating in the clustering and load balancing procedure maintain a cache about the topology of its vicinity in order to speed up self-organization and also to decrease the number of messages sent during the process. Section II describes basic self-organization algorithms that will be extended with topology cache. Section III tackles with considerations regarding the topology cache: establishment strategies and approaches to keep it up-to-date. Section IV elaborates on the load balancing problem used during the evaluation. We evaluated various caching strategies along different network dynamism characteristics; the results are summarized in Section V. Finally, in Section VI we conclude the work.

The novelty of this paper is the integration of topology caching with emergent self-organization algorithms.

## II. Clustering algorithms in mobile networks

While numerous clustering algorithms are known today, we chose a specific algorithm family for this paper, known as On-Demand clustering.

Manuscript received March 07, 2011.

Márton Legény was with the Budapest University of Technology and Economics, Hungary, H1117 Budapest, Magyar tudosok krt 2. (e-mail: legeny@hit.bme.hu). Now he works as a Software Engineer at Nokia Siemens Networks in Hungary.

Borbála Katalin Benkő is with the Budapest University of Technology and Economics, Hungary, H1117 Budapest, Magyar tudosok krt 2. (phone: 36-1-4633219; fax: 36-1-4633263; e-mail: bbenko@hit.bme.hu).

### A. Basic On-Demand Clustering

On-Demand Clustering (ODC) [12], [13], [14] is an emergent self-organization algorithm, developed at BT Labs, with beneficial properties on the node degree. The algorithm can be summarized as follows.

- The clustering process is initiated on demand, i.e. when a node is in need of expanding its cluster.
- The node where the demand for clustering raises, called *initiator*, selects one of its neighbors to serve as *match maker*.
- The match maker looks for a matching node, one that meets the initiator's clustering criterion, among its own neighbors.
- When a *match* is found, the initiator and the match establish a link, while, in order to keep the total number of links under control, the match maker removes its own link towards the match. This process is called *rewiring*.

It has been shown that ODC results in an emergent self-organization behavior, i.e. clusters are formed and expanded when local demand for that rises. However, ODC may not perform equally well in all cases. While the strict locality principle in the match search guarantees that the communication overhead remains under control; it sometimes prevents the fast formation of clusters, especially in case of sparser or type-wise highly diverse networks, where a suitable match if often not present amongst the match maker's one-hub neighbors. The locality principle in these cases causes not large enough clusters or not fast enough clustering.

### B. Spyglass

Spyglass [15] [16] is a match-centric extension of ODC, where the match maker is able look one hop farther than in the original algorithm. Spyglass differs from ODC in the last two bullet points, so, in the behavior of the match maker and in the details of rewiring.

- The match maker looks for a matching node, one that meets the initiator's clustering criterion, among its own neighbors. When no match is found, the match maker continues with checking its two-hop neighbors (the neighbors of its neighbors) for a match.
- When a *match* is found, the initiator and the match establish a link, while, in order to keep the total number of links under control, the match maker removes its own link towards the match (in case of it being a direct neighbor) or towards the neighbor that links to the match (in case of a two-hub neighbor).

The motivation behind Spyglass was to overcome the very strict locality principle of ODC without losing the beneficial properties of the original algorithm. However, looking at second-hop neighbors is an operation of exponential cost, so some kind of optimization – such as caching – is inevitable.

We decided to maintain a cache about the vicinity's topology, called Neighbor Cache (NC).

In Spyglass, just like in ODC, the communication is message based. Messages are used for the clustering itself (for example match search request, link establishment) and also for the purpose of gathering information about neighbors for the Neighbor Cache.

When a NC is available at the match maker, containing the neighbors of its neighbors, the match maker can use this data structure for finding a suitable match for the initiator without needing to send a single message to its neighbors.

## III. TOPOLOGY CACHING STRATEGIES

In [16] we showed that caching the overlay topology is an efficient method for reducing the communication overhead, i.e. the number of administrative messages sent between nodes, when the topology of the network is quasi-static.

In this paper we tackle with cache handling strategies for networks where the topology keeps changing from time to time, including highly dynamic topologies such as mobile or ad-hoc networks.

Building or rebuilding the neighbor cache from scratch requires a vast amount of communication messages, so the cache maintenance strategy, defining when and what to build/update, may have a significantly influence on the number of messages sent. Having a full neighbor cache at all nodes may also be unnecessary, as, only nodes playing the match maker role utilize this data structure.

On the other hand, the validity and freshness of the cached content is also of high importance, as the NC should support and not hinder the formation of clusters. To keep the cache up-to-date is especially vital when the underlying topology changes, introducing new potential matches or removing existing ones.

Ideally, a good tradeoff between the communication overhead and the freshness of the cache should be found.

The cache management should, in our point of view, include the following aspects: (i) Cache Building Strategy: describing when to establish the local Neighbor Cache, (ii) Cache Update Strategy: defining when to rebuild or refresh the contents of the cache and/or check its validity, (iii) Fallback Policy: what to do when no match is found in the neighbor cache or the cache-based result is corrupt.

### A. General Directions

We considered the following caching directions:

- **No caching.** Reference algorithm, no cache is used.
- **Static pre-caching.** The cache is built up for all nodes before the simulation starts. Reference algorithm, helps in separating caching overhead from the actual clustering traffic.
- **On demand cache establishment** (and its variants)**.** The cache is built 'on demand', that is, when the node becomes match maker. The main advantage of this method is that it is concentrating the efforts on nodes where the cache is actually required. Variants include:
  - *Conditional on demand caching*, where the initiator or the neighborhood needs to meet a certain criterion in order to establish the cache, and
  - *Random on demand caching*, where the choice whether or not to build the cache when the request arrives is made randomly.

While at first site cache building and cache update seems to be distinct, moreover, fairly independent aspects, some analysis leads to the conclusion that the longer the system's life time

the more important the refresh strategy becomes, and basically, *with time, the refresh strategy overtakes the role of cache establishment*. Hence, as the original cache establishment method will be suppressed by the cache update strategy on the long term anyway, we think it is sufficiently justified to use the same strategy for cache establishment as for cache update.

### B. Cache Rebuild Strategies

We examined the following build and rebuild strategies (all of which were originally designed as update directions for the on-demand cache establishment):

- **Random (re)build.** When becoming match maker, the node makes a random choice whether or not to build/refresh the cache. When "no refresh" is decided, the existing cache - if any - is used for answering the initiator's request.
- **Size-sensitive random (re)build.** Variant of the previous one, the random choice depends on the size of the cache. The consideration behind this method is that in denser parts of the network the chance of experiencing a topology change is higher than in rare segments (supposing that the probability of a change is independent and is the same in all nodes). This approach models the density of the network segment with the size of the neighbor cache.
- **Success based (re)build.** The match maker maintains a (sliding window) history about the success of the last n queries; and, when the success rate goes below a limit, the neighbor cache gets rebuilt. This strategy was designed to better accommodate to the "real" local needs of the neighboring initiators: as long as the cache works successfully, no communication should be wasted on updates.
- **Change based (re)build.** The cache gets rebuilt when the match maker senses a high enough change in its environment (direct neighbors). Optimally, if a node could observe all changes at no cost, this information could lead to mathematically optimal update strategies. In case of change based refresh, instead of truly observing all changes of the two-hop neighborhood, we approximate that with observing the direct neighbors only (which can be done without cost, by watching the channels themselves).
- **Type-sensitive change based (re)build**. A variant of change based refresh, when, as the demand arrives, not only the amount of the observed change is considered for the decision, but also its class (color): only such change classes matter that match with the request. This way, it is possible to exclude the effect of practically irrelevant changes from the decision.

### C. Fallback Strategies

As for fallback strategies, two possibilities were considered.

- **Fallback when no cache**. The algorithm falls back to the original excessive (message based) lookup only when there is no cache available at the node. Whenever a cache is present - may it be up-to-date or very old - the cache based answer will be used in answering the initiator's query.
- **Fallback when lookup unsuccessful**. Any time when a serving a query is unsuccessful from the cache (no match is found), the algorithm falls back to the original excessive lookup. Please note that this strategy neutralizes the effects of a possibly poor cache refresh, while on the other hand, it also wastes a large amount of communication messages when the lookup is unsuccessful due to topologic causes (e.g. there is really no match in the neighborhood). Even worse, the excessive overhead may get accumulated in the effected network part, occupying the communication channels repeatedly and hindering the real traffic. That is why this fallback strategy is used for comparison purposes only in our analysis.

## IV. Clustering based Load Balancing

Load balancing task is a common application area of clustering. We use a model where a load balancing problem generates the demand for the clustering.
The model is the following.

- The overlay network consists of colored nodes and links between them.
- Each node is able to process only those jobs that match its color.
- Links are not colored.
- Jobs enter the overlay network via colored workload generators, each statically attached to a matching-color node.
- Workload generators generate jobs and put them on the queue of the attached node. The expected value of the generation rate is constant.
- Nodes consume jobs from their local queue.
- When a node feels to be overloaded, it shares the local workload with its matching neighbors by transferring jobs from the local queue to them over a link. The sharing decision is also bound to specific conditions on the remote queue length (the acceptor cannot be overloaded) and the capacity of the link.
- When a node cannot find enough appropriate neighbor to share the load with, a demand for clustering occurs.

Hence, clustering is aimed to reorganize the overlay topology and to create a new link to a suitable node on demand.

Note that the load balancing task introduces certain changes in the requirements towards the clustering algorithm. When talking about a single clustering algorithm, the resulting cluster size is the most important goodness metrics. However, when clustering serves load balancing purposes, the need for the cluster size is limited. We do not need to generate the possible largest clusters, instead, just large enough clusters for the local excess workload (the creation of clusters larger than that would not bring further advantages but would cost communication messages). On the other hand, the clustering speed becomes more vital for load balancing: the initiator needs the match urgently. Given that the job generation rate is constant, every unsuccessful search for a match just worsens the initiator's situation.

## V. EVALUATION

In this chapter we present our simulation results regarding the cache maintenance methods, with a focus on the communication overhead and the clustering / load balancing performance. After describing the test setting and the evaluation criteria, we elaborate on the details of the simulation results of the various algorithms and network change characteristics.

### A. Test Setting

#### 1) Simulation Setting

Measurements were conducted on a showcase network consisting of 10,000 nodes and 25,125 links initially, with random graph topology [17], and workload generators attached to 5% of the nodes. Nodes belong to 10 different classes (colors), and load balancing swings into action once the local queue length exceeds the static limit of 5 unprocessed jobs. The choice of network size and density was motivated by our previous work on static topologies [15], where this network size was found to be convenient for demonstrating the scalability of the emergent algorithms as well as for pointing out the differences between directions.

The simulation was limited to 500 rounds. Each workload generator had a static limit of generating 500 jobs at most. The generation rate changes between 1 and 10 jobs per round, as is constant for the life time of the generator.

We simulated the same scenario (excess workload triggers self-organization) for all examined caching strategies along two network dynamics profiles.

#### 2) Network Dynamics

The dynamics of the network was modeled by three factors: (i) Node Disappearance: nodes disappear from time to time. (ii) Node Appearance: new nodes keep appearing in each round. (iii) Node Movement: nodes change their location a bit in each round, resulting in new physical neighborhood relationships.

We used a physical proximity model for generating the initial overlay topology. Nodes were distributed randomly over a geographic area of 1000×1000 m. Nodes within a visibility distance (10m) 'see' each other, i.e. are connected by a link. Links generated by the self-organization process (overlay links) may exceed this visibility limit as long as the original initiator and match maker originating the link remain visible for each other. Node mobility was modeled with the random movement mobility pattern: a step into a randomly selected direction with the default speed. After mobility, when two nodes that used to see each other are now out of range, we took a worst-case view by removing all overlay links that originated from the two node's former neighborhood. This is a pessimistic view, as the overlay links in practice would not need to go down as long as alternative routes exist between them. However, we believe that a worst-case assumption is better to be used during evaluation than a too optimistic assumption.

Node appearance was modeled in form of creating new nodes at random locations. Node disappearance is modeled in form of removing randomly selected nodes from the current population (i.e. the properties of the node, such as age, connectedness or location, did not play a role when making the selection).

We defined two essentially different network change profiles.

- **Slow changing static sized network**. In each 50 steps long window 10% of the nodes disappear, 10% new nodes appear, and 50% of the nodes move with a speed of 1 m/movement.
- **Fast changing static sized network**. In each 10 steps long window 10% of the nodes disappear, 10% new nodes appear, and 50% of the nodes move with speed of 3 m/movement.

In both profiles, the size of the network remains quasi static (the ratio of disappearance equals to the ratio of new node generation), hence the network itself will neither expand nor shrink.

#### 3) Examined Strategies

The following cache strategies were examined.

- No Cache. No cache is used.
- On Demand Cache. Cache is established when the node becomes match maker. Cache is rebuilt when the request cannot be served from the current cache.
- On Demand Cache with PreCaching. All nodes are equipped with a cache at startup (at no cost). The cache is updated when the node becomes match maker and cannot serve the request from the current cache.
- Random Cache. When the node becomes match maker, a random decision is taken whether to build/update the cache. No fallback is used when the query is unsuccessful.
- Size Sensitive Random cache. The same as Random, but the probability of update is proportional to the number of direct neighbors.
- SuccessBased cache. Cache is built/rebuilt when 3 out of the last 4 queries were unsuccessful.
- ChangeBased cache. Cache is built/rebuilt when at least 10% of the neighbors changes.
- TypeSensitive ChangeBased Cache. Changes of the neighboring nodes are saved into a local registry. For each query, the change registry is evaluated. When at least 10% of the matching-color nodes are affected by the change, the cache is rebuilt, and the change history gets cleared.

#### 4) Simulation Environment

The simulation framework was written in Java 6 SE, and the experiments took place on a desktop PC with 2GHz dual core processor and 2 GByte RAM.

### B. Evaluation Criteria

The following metrics were applied for evaluation:

- Message count depicts the amount of ***communication overhead*** generated by the algorithm variant. The value includes self-organization and cache building / rebuilding related messages only. Small message counts are preferred over high message counts.
- ***Number of clustered nodes*** at the end of the simulation. Intuitively, the larger this number is the better the self-

organization performs. However, the demand for clusters is constrained by the amount of workload, so the cluster size cannot grow indefinitely.

- The ***unprocessed job curve*** drafts the processing dynamics of the system by plotting the number of injected but unprocessed jobs versus time. As the simulation includes a limited number of jobs, with time, the curve reaches zero (i.e. all jobs are processed). The area under the curve is also a good estimate for the goodness of the algorithm, smaller area suggests better performance (smaller waiting times). The tail of the curve is often very long. This may mean to things: either the local job generation rate is one (meaning that the node will process all jobs itself because of never getting overloaded); or that there are nodes in the system unable to establish clusters so they are left to process all incoming jobs on their own. That is why, when comparing curves, we do not necessarily locate the point when the curve reaches zero, instead, the point where the curve reaches a small but non-zero number.
- The ***number of overloaded nodes*** depicts the dynamics of the demand for clustering. This metrics can be used for two purposes: (i) to understand the characteristics of the demand that triggers clustering; and (ii) to understand how the demand is silenced by the clustering and load balancing algorithm.

### C. Results

#### 1) Communication Overhead

Figure 1 summarizes the measured total message count for each examined caching strategy along the slow and fast network dynamics profiles. In all cases, cache based directions significantly reduce the communication overhead, sometimes even by a magnitude, compared to the baseline Spyglass algorithm 'No Cache'.

From the examined strategies, blind On Demand caching produces the highest amount of messages. It is not only high compared to the conditionally caching strategies, but also in means of the reference algorithm No Cache, suggesting that a large portion of the nodes becomes match maker at least once in its lifetime (hence a demand for cache establishment occurs). In the variant when PreCaching is in place (OD with PreCaching), messages serve cache update purposes only.

Random caching, either size sensitive or pure random, produce around half as many messages as On Demand does. Size sensitive update, as its name suggests, tends to trigger cache updates more frequently in denser parts of the network, resulting in somewhat more messages then the pure random variant.

The success and change based strategies are surprisingly economic in terms of messages, the Type Sensitive Change Based strategy even beats OD with PreCaching in case of fast network change.

The effect of the network change profile (fast or slow) varies between strategies. One unexpected result is that for No Cache the fast network change profile produces significantly (31%) less messages than the slow network change. The explanation is that fast network change seems to be a natural booster for finding appropriate partner nodes, as the potential partner nodes, through their natural movements, have a chance of getting into the vicinity by themselves. That's why, there's less need for looking at two-hop neighbors, which is the message-consuming step.

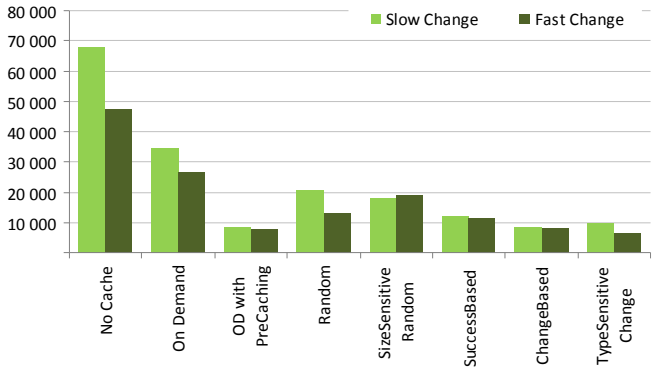


Figure 1: Message count with various network dynamics and caching strategies. The original Spyglass algorithm (No Cache) is easily beaten by all cache based strategies.

The same boosting effect of the fast network change can be observed in several other strategies, in various amounts. The only exception is the size sensitive random cache, where fast changes in the dense regions of the area cause extensive cache updates, producing somewhat more messages than in case of the slow network change profile.

#### 2) Clustered Nodes

Figure 2 shows the number of clustered nodes in the last round for the examined caching startegies along the two network dynamics profiles.

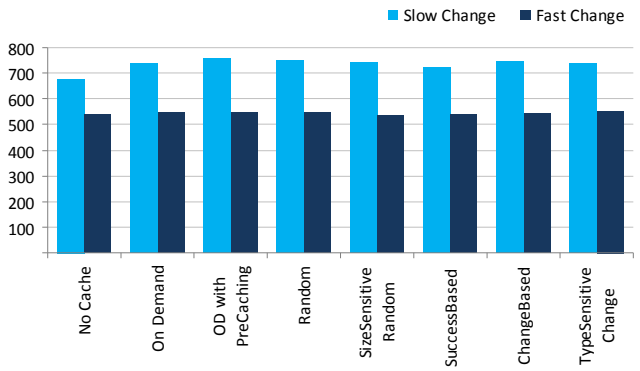


Figure 2: Number of clustered nodes at different network dynamics and caching approaches. While the difference is visible in case of slow changes, in fast changing networks none of the algorithms create clearly large clusters.

The network dynamics seems to clearly and consistently effect the number of clustered nodes. With fast network change, the total number of networked nodes is significantly less in all cases than with slow changes. Atfter detailed analysis of the logs, we identified two reasons for that. (i) Fast network changefacilitates natural clustering by moving potential neighbors into the vicinity for some little time and then moving it away; hence it produces smaller but more dynamic clusters

(i.e. the neighbors, after getting a share in the work, move away). With slow network change, neighbors don't move away so easily, get overleaded by the shared workload, hence, the initiator needs to expand the cluster. (ii) The pessimistic handling of overlay links also influences the result. With fast network change, a large amount of overlay links get removed in each step, shrinking the clusters.

### 3) *Unprocessed Job Curve*

When observing the unprocessed job curve [Figure 3] for the smart caching strategies and network change profiles, it is clear again that each of the algorithms perform better in the fast changing case. This, again, confirms the natural boosting property of the network change, which, by changing the overlay neighbors more frequently, tends to guarantee that neighbors are not overloaded, hence can be used for workload sharing. Under heavier overall load, where a higher percentage of nodes would be overloaded, the effect of this beneficial property would weaken.

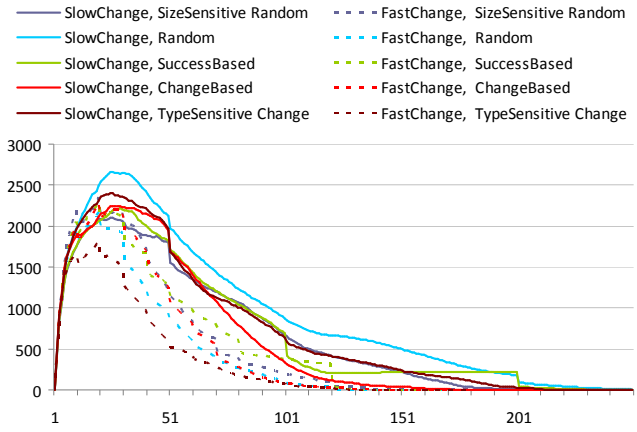


Figure 3: Unprocessed job curve at various change dynamics and caching approaches. Line style denotes the dynamics; line color shows the algorithm type. All algorithm variants perform better in the fast changing network.

Change based directions (ChangeBased, TypeSensitive Change) and the Success based strategy are among the top performers along both network change profiles.

Random caching seems to touch both extremities: it is the least effective in the slow scenario and among the bests in the fast changing one.

For comparison, Figure 4 shows the unprocessed job curve of the non-smart directions, i.e. No cache, and blind On Demand without and with pre-caching. The fast changing scenario also brings better results here. The shape of the pre-cached OD is good, however, the shape of the other curves, as well as the area under the curve, suggest that OD and No Cache provides weaker performance than smart strategies could bring; meaning longer waiting times and smaller throughput. This can be explained by the too simple fallback strategy used in these strategies: the cache is only updated when no match is found in the existing data base. In practice, this may result in returning useless matches to the initiator (matches that are already neighbors for it, or are out of range by now), wasting the time. The other reason is that updating the cache (or querying all neighbors) takes time, delaying the answer. In case the query cannot be served anyway, i.e. there's no match in the vicinity, these extra rounds get wasted again and again.

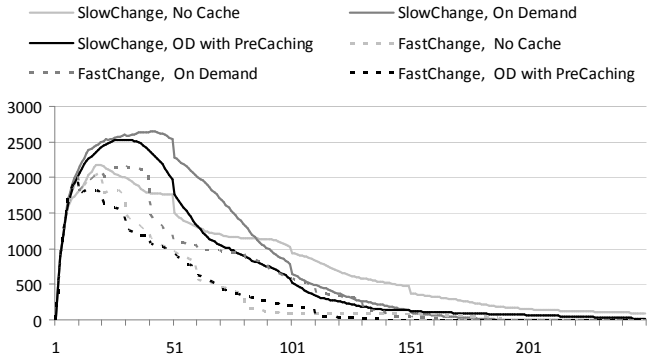


Figure 4: Unprocessed job curve without cache update.

### 4) *Number of Overloaded Nodes*

Figure 5 displays the number of unprocessed jobs for the slow changing network profile. Random cache produces the highest peak at 85 overloaded nodes at a time, meaning that this strategy may easily be the slowest to respond to the occurring demand. No Cache, SizeSensitive Random, and ChangeBased produce the lowest peaks (66-67 overloaded nodes), suggesting that these strategies will not hinder the algorithm in reacting with speed.

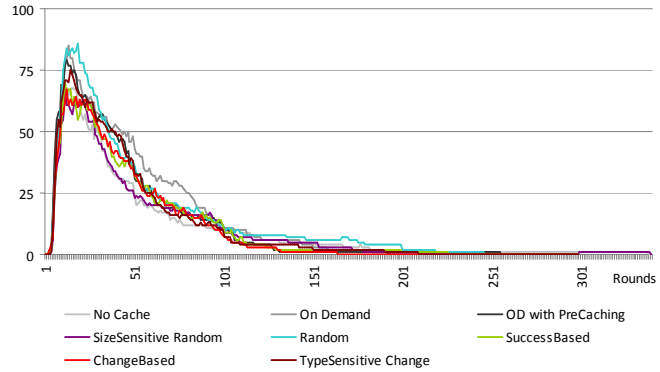


Figure 5: Number of overloaded nodes with the slow network change profile.

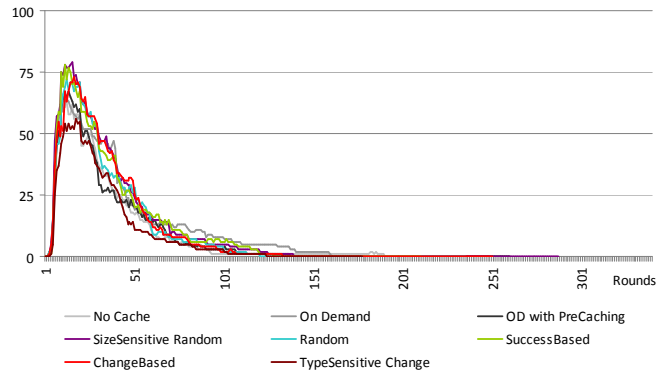


Figure 6: Number of overloaded nodes with the fast network change profile.

The same curves for the fast network change profile can be seen in Figure 6. Not only peaks are smaller but also the number of overloaded nodes decreases more rapidly, due to the more efficient workload sharing.

### 5) *Result Summary*

The performance of the examined caching strategies tends to be not as conclusive as one could expect; each strategy has its stronger and weaker points. Putting all together, we created a ranking in each evaluation category. Rank 1 denotes the best performance and Rank 8 means the weakest. Strategies with similar performance may share the same rank position. Table 1 shows the ranks and also the total 'rank' earned with uniform weighting.

| Strategy | Job Curve | Reaction Time | Comm. Over-head | Cluste-ring | SUM |
|---|---|---|---|---|---|
| **No Cache** | 4 | 2 | 8 | 2 | **16** |
| **On Demand** | 8 | 5 | 7 | 1 | **21** |
| **OD with PreCaching** | 1 | 4 | 2 | 1 | **8** |
| **Random** | 6 | 4 | 4 | 1 | **15** |
| **SizeSensitive Random** | 2 | 6 | 4 | 1 | **13** |
| **SuccessBased** | 4 | 5 | 3 | 1 | **13** |
| **ChangeBased** | 2 | 3 | 1 | 1 | **7** |
| **TypeSensitive Change** | 2 | 4 | 1 | 1 | **8** |

Table 1: Rank of algorithms per evaluation criteria. Results summarize the outcomes of the two network change profiles with equal weight.

When considering all criteria with equal weights, the top performers are the change based caching strategies, ChangeBased and TypeSenistive ChangeBased caching. (The other favorable algorithm, OnDemand with PreCache is just a theoretical toy, as the existence of a pre-built topology cache is not a realistic assumption in ad-hoc networks.)

However, the order of algorithms may change if the weights of the evaluation criteria are not equal, if a certain network change characteristics should be considered instead of the mixture of the two, or if some assumption made in the algorithms cannot be implemented in a specific problem case (e.g. monitoring the existence of the links to the neighbors at no cost). In each specific problem case, a tradeoff between communication overhead and the actual performance should be found.

## VI. SUMMARY

In this paper we examined various directions for a topology cache aided self-organization and load balancing algorithm. The topology cache aims to reduce the communication overhead of looking up matching nodes in the vicinity for load balancing. We defined six complex cache building and rebuilding strategies (in order to follow the changes in the network) and two reference algorithms; and evaluated each of them along a slow and a fast network changing profile through simulation. Experiments pointed out that fast topology change naturally stimulates self-organization, i.e. tends to bring matching, and not overloaded nodes into the direct neighborhood at no cost, resulting in possibly smaller clusters but better overall load-balancing and job processing curves. In case of a moderate network change profile, the difference between caching strategies was more visibe. All examined smart caching strategies beat the reference algorithms in terms of overall performance: (i) the communication overhead was significantly smaller, sometimes even by a magnitude, (ii) no significant difference was observed in the size of the clusters produced, , and (iii) the resulting load balancing dynamics, after a short initial delay, tended to outperform the non-cached version due to the O(1) cost of a cache lookup instead of the larger cost of a real neighbor lookup. The winner of the examined caching algorithms is a change based approach, rebuilding the cache only if the amount of approximated change in the vicinity exceeds a certain limit; where we approximate the total change in the two-hop neighbor-hood by the observable change amount among direct neighbors.

## VII. ACKNOWLEDGEMENT

The support of the Hungarian Government through the TÁMOP-4.2.1/B-09/1/KMR-2010-0002 project at the Budapest University of Technology and Economics is acknowledged.

## REFERENCES

[1] D. J. Cook, J. C. Augusto, V. R. Jakkula. 2009. „Ambient Intelligence: Technologies, Applications, and Opportunities." in *Journal of Pervasive and Mobile Computing*, 5(4): 277-298.

[2] J. C. Augusto. 2007. „Ambient Intelligence: the Confluence of Ubiquitous / Pervasive Computing and Artificial Intelligence." in *Intelligent Computing Everywhere*, Springer Verlag, pp. 213-234.

[3] G. Di Caro, F. Ducatelle, L. M. Gambardella. 2005. „Swarm Intelligence For Routing In Mobile Ad Hoc Networks." in *Proceedings of the 2005 IEEE Swarm Intelligence Symposium (SIS)*, pp. 76-83.

[4] A. Montresor, H. Meling, Ö. Babaoglu. 2002. „Messor: Load-Balancing through a Swarm of Autonomous Agents." in *Proceedings of 1st Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)*, Springer Verlag, pp. 125-137.

[5] A. Montresor, H. Meling, Ö. Babaoglu. 2003. „Toward Self-Organizing, Self-Repairing and Resilient Distributed Systems." in *Number 2584 in Lecture Notes in Computer Science*, Springer Verlag, pp. 119-124.

[6] IST CASCADAS Deliverable D3.2: „Report on rule-based modules for unit differentiation using cross-inhibition and/or resource competition." IST CASCADAS Project, 2007.

[7] S. Jain, R. Mahajan, B. Niswonger. 2000. „Self-Organizing Overlays." Technical report, University of Washington, DC, USA.

[8] S. Apel, K. Böhm. 2005. „Self-Organization in Overlay Networks." in *Proceedings of CAISE'05 Workshops (Workshop on Adaptive and Self-Managing Enterprise Applications)*, Vol. 2, pp. 139-153.

[9] I. Clarke, O. Sandberg, B. Wiley, T. W. Hong. 2000. „Freenet: A Distributed Anonymous Information Storage and Retrieval System." in *Proceedings of Workshop on Design Issues in Autonymity and Unobservability, Berkeley*, CA, USA.

[10] P. Gburzynski, B. Kaminska, W. Olesinski. 2007. „A Tiny and Efficient Wireless Ad-hoc Protocol for Low-cost Sensor Networks." in *Proceedings of the conference on Design, Automation and Test in Europe (DATE'07)*, pp. 1557-1562.

[11] R. Krishnan. 2004. „Efficient Self-Organization of Large Wireless Sensor Networks". Ph.D. Dissertation, Boston University, College of Engineering.

[12] F. Saffre, R. Tateson, J. Halloy, M. Shackleton, J.L. Deneubourg. 2008. “Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications”. in *The Computer Journal*.
[13] E. Di Nitto, D. J. Dubois, R. Mirandola, F. Saffre, R. Tateson. 2008. „Self-Aggregation Techniques for Load Balancing in Distributed Systems.” in *Proceedings of SASO 2008*, pp 489-490.
[14] IST CASCADAS Deliverable D3.1: „Aggregation Algorithms, Overlay Dynamics and Implications for Self-Organised Distributed Systems.” IST CASCADAS Project, 2006.
[15] M. Legeny. 2010. “Distributed and hybrid algorithms in Self-Organizing Networks” (In Hungarian). Master’s thesis, Department of Telecommunications, Budapest University of Technology and Economics (unpublished).
[16] M. Legeny, B. K. Benko. 2010. “Design of Novel Self-Organization Algorithms through Simulation.” in *Proceedings of EUROSIS ISC’2010*, pp. 10-15.
[17] P. Erdős, A. Rényi. 1960. „The Evolution of Random Graphs". in *Magyar Tud. Akad. Mat. Kutató Int. Közl*. 5: 17–61.