Adaptive Agent-based Self-organization for Robust Hierarchical Topologies

Evangelos Pournaras Martijn Warnier Frances M.T. Brazier
Intelligent Interactive Distributed Systems*
VU University, Amsterdam
The Netherlands
{E.Pournaras,M.Warnier,FMT.Brazier}@few.vu.nl

Abstract

Virtual organizations in large-scale distributed environments can organize their communication in a hierarchical topology (i.e., trees). However, such topologies can be unreliable as local failures have a global impact in the organization. Hierarchical topologies need to adapt continuously to changes of the underlying environment. Pro-active and re-active self-organization can make such topologies highly robust.

This paper proposes AETOS, the Adaptive Epidemic Tree Overlay Service. AETOS is a new agent-based approach for building and maintaining on-demand robust tree topologies that structure communication. Agents are pro-actively (self)-organized appropriately in a tree to minimize the effect of failures. In addition, they re-actively rewire their connections to reflect changes in the environment. The self-organization model, the control of the system and an illustrative example are discussed in this paper.

1 Introduction

Hierarchical virtual topologies that define the communication structures over unreliable large-scale distributed infrastructures, require robustness. These topologies can be used in a wide range of applications [5, 9, 13, 14, 15]. Robustness refers to the fact that the hierarchical topology must be *dynamic*, *adaptive*, *reconfigurable* and finally *self-organized* to (i) handle the changes of the underlying distributed environment and (ii) reflect the application requirements.

Software agents have been proposed as a paradigm for management of distributed systems [12]. Self-healing of hierarchical structures [3] and agent-based self-organization [16] appear in various approaches of distributed self-management.

*Affiliated with Delft University of Technology from the 1st of September 2009

Adaptive central control of hierarchical topologies for communication is not always an option nor a scalable solution. This is especially the case in the aforementioned large-scale distributed environments. Building and maintaining robust hierarchical topologies in a distributed manner is the challenge this paper addresses. Local intelligent software agents play a central role in acquiring a global hierarchical topology using their ability to *cooperate*, *adapt* [4] and *reconfigure*. Local agent behavior can make the topology self-organizing.

This paper focuses on tree topologies as an instance of hierarchical structures. In trees, local failures have a global impact in the topology as the removal of a node disconnects the branches underneath from the main body of the tree. Creating self-organized tree topologies that are resilient to failures requires: (i) to pro-actively organize (sort) agents over the tree in such a way that a potential failure has minimum impact on the tree structure, (ii) to re-actively adapt to changes in the environment by reconnecting, in case of failures, or rewiring connections to improve the robustness of the topology.

These are the two main concepts of *AETOS, the Adaptive Epidemic Tree Overlay Service*. AETOS is the mechanism that this paper proposes to build and maintain robust tree topologies in distributed environments. It is based on a 3-layer architecture as the core of self-organization. These three layers are facilitated and managed by a local software agent, the *AETOS agent*. Applications can be build on top of this agent-based dynamic tree overlay. The interaction between the AETOS agent and the application is managed through another local agent, the *AETOS proxy*. This agent provides the tree overlay on-demand to the application, and is responsible for bootstrapping and terminating the self-organization process.

The contribution of this work is three-fold. This paper proposes the following:

1. The *myopic competitive agent model*, a highly reconfigurable self-organization model. It enables the use of dynamic proximity criteria. This model is inspired by

the conceptually similar Topology MANagement protocol (T-MAN [6]) that uses static proximity criteria. Section 4 discusses this model.

2. The *AETOS agent* that realizes the self-organization behavior of the node participating in the tree topology. Section 5.1 outlines the core of the AETOS agent.
3. The *AETOS proxy* that enables the applications to use tree overlays on-demand. Section 5.2 illustrates how the AETOS service interacts with the application through this proxy.

AETOS can provide robust tree topologies independent of the application type. This generic approach allows more flexibility for exploring and using hierarchical topologies in different application types compared to the related work illustrated in Section 3.

2 Motivation and Problem Overview

Structure and robustness are difficult properties to achieve in large-scale distributed systems. This section outlines how hierarchical topologies can benefit an application. It also reveals how hierarchical topologies are affected globally by local failures.

2.1 Application Examples

The coordination of energy consumption among thermostatically controlled appliances, illustrated in [14], is a representative example of a resource allocation problem based on a hierarchical topology and software agents. In this system, local agents aggregate information about their energy utilization and reason about it over the hierarchical topology towards satisfying the global goal of the system, that is the stabilization of the total energy consumed in the network.

Hierarchical topologies also appear in peer-to-peer systems in which *super-peers* are placed at a higher level due to their additional duties or privileges they have. For example, in [13], peers with a high performance profile are responsible for performing dynamic load-balancing in an overloaded network. These 'virtual servers' are interconnected and forward the load-balancing requests between themselves until they satisfy each request. This layer creates another level in the hierarchy, in a similar manner to the Gnutella II peer-to-peer network.

Hierarchies can also be useful for other types of applications, such as video streaming [15], security [9] and distributed databases [5]. The main challenge in using hierarchical topologies is how the uncertainties of distributed environments influence the topology and the application that uses it. The robustness and the maintenance of these topologies enables the effective utilization by the applications.

2.2 Failures and Robustness

This paper focuses on virtual tree topologies that are built on an underlying network infrastructure. In this paper, these virtual topologies are referred to as *overlay* networks. The *nodes* in the overlay can be controlled by one or more *software agents*, i.e., the nodes are the local environment of the agents. This paper examines how the failures of nodes influence the tree overlay and how the local software agents can cooperate to make the topology robust to failures.

Failures can occur in any deployed system. In this paper, a failure is defined as an abstract state in which the overlay communication of a node is interrupted. The interruption refers to the fact that a node (and its agents) in this state cannot reach and cannot be reached by other nodes. It is an abstract condition because it can be caused by a wide range of reasons, such as disconnections in the underlying network, firewalls, security attacks etc. A tree overlay is robust if it is capable of retaining its structure under the effect of failures.

The main problem of trees is that if a node fails, the branch under the failed node disconnects from the main body of the tree. Only a few failures can be enough to turn the structured environment in an unstructured one. This paper approaches the problem of minimizing the influence that individual failures cause to the whole tree structure. The main goal of the AETOS service is to guarantee that failures have a minimum global impact on the tree overlay. This means that disconnected branches should contain as few nodes as possible.

3 Related Work

Related work focuses on (i) hierarchical self-healing frameworks based on agents and (ii) peer-to-peer self-organization of robust tree overlays.

Hierarchical agent-based approaches provide a multi-level local control, recovery and healing in distributed systems. The hierarchical framework proposed by [10], includes a number of hierarchical control algorithms to maintain Quality of Service (QoS) and Service Level Agreements (SLAs), and mechanisms for tuning, load-balancing and provisioning. In another hierarchical autonomic management system for grid applications [1], autonomic managers are controlled by software agents and reason about the overall system behavior. The orchestrating managers are organized in a hierarchy and pursue their goals using QoS contracts. The AHIT^{AC} model [9] is based on a hierarchical, 3-layer architecture for intrusion tolerance. The 'executing', 'tolerance' and 'evaluation' layers achieve self-recovery and self-optimization on object networks that finally become more stable and tolerant.

The above methods incorporate the notion of adaptation as re-active behavior and not as pro-active. It is unclear how these methods can be applied in scalable distributed environments and in different types of applications.

As far as robust tree overlays are concerned, video streaming applications build and maintain the topology based on various performance metrics as outlined in [11, 15, 17]. For example, in [15], the broadcast tree is built based on the bandwidth and the up-time of the participating nodes. These two metrics lead to two different versions of a tree, one bandwidth-ordered and one time-ordered. The proposed algorithm performs some shifting operations that combine these two versions to a minimum-depth broadcast tree overlay. However, the algorithm does not consider the optimum number of children that each node should retain for controlling the processing cost.

Furthermore, some of the existing work has investigated the support of the tree overlay by another underlying overlay that facilitates the hierarchy and supports the dynamic states of the nodes. mTreebone [17] is based on stable nodes that are self-organized through an auxiliary mesh overlay. This mechanism considers the bandwidth of the nodes that perform video multicast. However, it only utilizes stable nodes, excluding others from the system. Similarly, in [5], the tree overlay is based on skip graphs and is able to perform complex range queries over multidimensional data. This approach requires additional complex mechanisms for load-balancing and healing.

Some slightly different approaches are based on epidemic protocols. For example, TAG [11] builds a robust tree overlay with low delay. An integrated pull gossiping protocol performs switching among multiple paths to improve the bandwidth utilization. Finally, the Plumtree [8] overlay combines eager and lazy push gossiping strategies to build a dynamic tree. However, the fault tolerance and tree repair is not part of the main construction protocol, thus manual repair actions are performed in these approaches.

The *main gap* that this paper identifies in almost all of the illustrated approaches *is the lack of a generic mechanism that could optimize a tree overlay for any type of application*. For tree overlays, the related approaches *do not manage to keep the self-organization process independent of the application requirements*.

However, three main critical characteristics are identified in the illustrated approaches:

1. The utilization of local performance and robustness metrics that are related to the application, e.g. bandwidth, delay, time, availability etc.

2. The underlying support of other overlays, e.g. mesh overlays, skip graphs.

3. Protocols that exhibit high robustness in dynamic environments, e.g. gossiping protocols.

AETOS is based on the three above characteristics for cost-effective building and maintenance of robust tree overlays.

4 The Myopic Competitive Agent Model

The AETOS service is facilitated by an agent with the following knowledge: the *robustness* r, the *random view* $\boldsymbol{R}$, the *myopic view* $\boldsymbol{M}$ and the *tree view* $\boldsymbol{T}$.

The robustness r is attributed and defined by the application and may represent metrics such as availability, bandwidth, reputation etc. It is used as a ranking scheme for the robustness of the nodes. Robustness values are assumed to be independent among the agents, an assumption that holds in various distributed systems such as the Overnet peer-to-peer system [2]. Furthermore, the robustness r may be related to more than one metric. In this case, either a function combines different metrics to one or they are weighted according to the requirements of the application [15].

The random view $\boldsymbol{R}$ is the primary neighboring list. It contains a number of other random agents from the distributed environment. Based on the random view, the myopic and the tree view are formed.

The myopic view $\boldsymbol{M}$ of an agent expresses the proximity of an agent to its neighbors. It is filled by selecting neighboring agents with the lower *robustness distance* d from the agent's local robustness r. For example, the robustness distance between an agent x and an agent y is $d = |r_x - r_y|$. The search space of an agent that is used to fill the myopic view is the random view. Close proximity agents can also exchange neighbors (gossip) and further discover each other faster.

The tree view $\boldsymbol{T}$ is a list with the parent-children neighbors of an agent. Similarly, the search space of an agent to fill the tree view is the myopic view.

The aforementioned views are dynamic and neighbors can change over time. The length of the random view depends on the protocol that selects the random neighbors. For example, the Peer Sampling Service [7], discussed in Section 5.1, defines the length of the view equals to 20-30 in large-scale environments. The myopic view can have any length in the range $0 - |\boldsymbol{R}|$. Its length varies during the runtime as it is dynamic and reconfigurable. The myopic view reconfiguration is illustrated in Section 4.2. The length of the tree view is $n + 1$, with n the number of children that the local AETOS agent retains.

An agents in AETOS is *competitive*. This means that it aims to continuously improve its position in the tree by choosing to connect with more robust agents. This competitive behavior leads an agent to reconfigure its myopic view $\boldsymbol{M}$ by changing the proximity criteria. The next subsections illustrate the role of the myopic view and the reconfigurations performed.

4.1 Myopic Agent View

An AETOS agent must find its parent and its children from its myopic view that expresses the proximity with other agents in the environment. For this reason, the myopic view $\boldsymbol{M}$ of an agent is split in two parts, the sorted set of *candidate parents* $\boldsymbol{P}$ and the sorted set of *candidate children* $\boldsymbol{C}$ such that $\boldsymbol{M} = \boldsymbol{P} \cup \boldsymbol{C}$. Agents with higher robustness r than a local agent belong to the set of candidate parents and agents with lower values belong to the set of candidate children.

Furthermore, each agent in the tree, excluding the leaves, is assumed to have a number of children n. The ratio of the length of the candidate children set over the length of candidate parents set ($\frac{|\boldsymbol{C}|}{|\boldsymbol{P}|}$) is proportional to the number of children n. For example, if $|\boldsymbol{M}| = 12$ and $n = 3$ then $|\boldsymbol{C}| = 9$ and $|\boldsymbol{P}| = 3$. This guarantees that the search space for children and the parent is proportional. Figure 1 illustrates how the myopic view is formed from the random view and how it is split and sorted in the two sets of candidate tree neighbors.

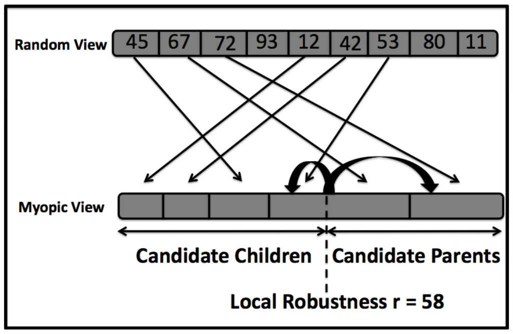

Figure 1. The process of filling and sorting the myopic view with the appropriate agent neighbors. The two bold curved arrows illustrate that the local agent by default chooses the more robust candidates.

Finally, a locally competitive agent selects the candidates with the highest values from the candidate parents and children respectively. The two bold curved arrows in Figure 1 depict this selection of the more robust candidates.

4.2 Myopic View Reconfigurations

The proposed myopic competitive agent model has a major drawback. The myopic view is filled based on the proximity criteria of robustness. However the myopic view is a partial list, thus it does not necessarily contain the final best neighbors. This is a scenario that can appear easily as the values r of all of the agents may not follow a uniform distribution. Note that the length of the views is assumed to be equal for all the agents. For this reason, AETOS updates the proximity criteria dynamically until the best neighbors are found. The myopic view must converge to the tree view, thus the clustering of agents must be dynamic.

The idea of introducing dynamic proximity criteria is realized by reconfiguring and adapting the myopic view. In other words, the myopic view is reconfigured by including or excluding areas.

The ranges of robustness values for candidate parents and children are examined below. All of the indexes refer to the robustness values in the myopic view. A potential parent p belongs to the candidate parents range P such that $p \in P = [p_{min}, p_{max}]$. Similarly, the potential children $c_1 < c_2 < ... < c_n$, with n the number of children, point to the candidate children range C such that $\{c_1, c_2, ..., c_n\} \in C = [c_{min}, c_{max}]$. Figure 2a illustrates the initial ranges of candidate neighboring sets. The agents perform the following reconfigurations:

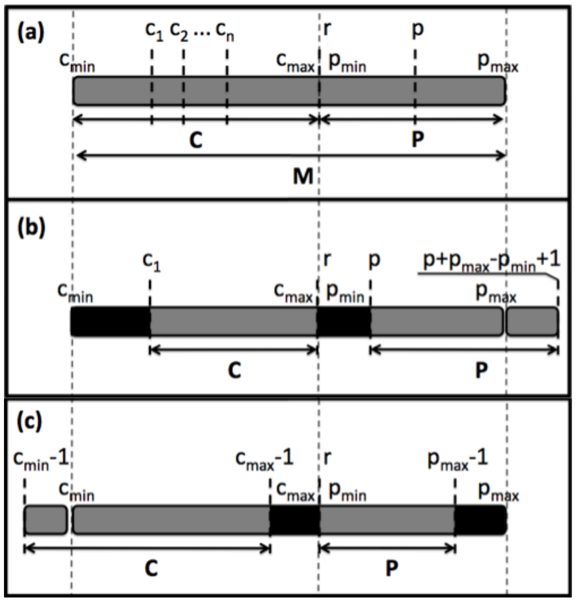

Figure 2. The parent and children candidates in the myopic view. (a) initial myopic view, (b) after an upgrade reconfiguration, (c) after a downgrade reconfiguration.

1. **Initial Configuration:** the agent with the higher robustness r in each candidate set is the potential child or parent respectively. In this case $p = p_{max}$ and $c_i = c_{max}$, for the ith potential child. The two bold curved arrows in Figure 1 illustrate the concept of the initial configuration.
2. **Upgrade Reconfiguration**: the agent has already found a parent or its children and it seeks to connect

with more robust agents (competitive behavior). In order to achieve this, it binds the starting point of its view to the robustness values of the selected agents and fills the view with more robust agents. The candidate parents range is reconfigured as $P = [p+1, p+p_{max} - p_{min} + 1]$ and the children candidate range as $C = [c_1 + 1, c_{max}]$. Figure 2b depicts the upgrade reconfiguration.

3. **Downgrade Reconfiguration**: if a previously selected candidate agent has rejected the connection (see Section 4.3), the view is updated with less robust agents. In this case, the candidate ranges are updated as $P = [p_{min}, p_{max} - 1]$ and $C = [c_{min} - 1, c_{max} - 1]$ respectively. Figure 2c illustrates how the view is updated in this case. Note that, the downgrade reconfiguration is performed step-by-step, decrementing the positions by one for every rejected parent or child connection respectively.

Agents have the option to switch from a downgrade or upgrade configuration back to the initial one. Furthermore, the view of an agent can be a result of both an upgrade and a downgrade reconfiguration. Any applied reconfiguration keeps the length of the view equal or lower than the initial maximum length.

4.3 Triggering the Reconfigurations

The random and myopic views create link overlays that are unidirectional. However, the tree overlay requires bidirectional links, thus the tree views must be consistent with respect to the fact that the neighbors should be mutually acceptable. For this reason, agents exchange four basic messages based on which: (i) they configure their tree views, and (ii) the myopic view reconfigurations are triggered.

The four exchanged messages are (i) the *request* of a parent or child connection, (ii) the *acknowledgment* of a request, (iii) the *rejection* of a request and (iv) the *removal* of a parent or child connection.

In their active state, AETOS agents send a number of *parent and child requests* to the agent selected from the myopic view. The passive state of the agents defines the appropriate reactions to the messages received. The local AETOS agent reacts to a *parent or child request* as follows:

1. It checks if the metrics of the two communicating agents are consistent. This means that the value of the parent should be higher than the value of the child. If inconsistencies occur due to changes in the values of robustness, the local agent sends a *rejection* message to the requesting agent with information about the value of local robustness.

2. If there are no inconsistencies, the local agent either
   (a) updates and inserts the agent that sent the parent/child *request* in its tree view. In this case, the requested agent replies with an *acknowledgment*. If the update of the tree view is performed by replacing an existing agent with a better one, then a *removal* message is sent to the agent to be replaced. Or,
   (b) it rejects the request and a *rejection* message is sent. In this case, the existing parent or children agents are more robust than the one that sent the request.

   In both cases the reply-messages contain information that reflects the more recent values of the robustness metric.

3. The myopic view is potentially adapted by an upgrade reconfiguration.

If the local agent receives an *acknowledgment* of its request it performs:

1. an update of its tree view by inserting the new neighbor. If the update is a replacement, it sends a *removal* messages to the replaced agent.
2. an upgrade reconfiguration in the myopic view.

The *rejection* message triggers the following:

1. a downgrade reconfiguration of the myopic view.

Finally, the *removal* message is treated similarly to a *rejection* and thus the agent performs:

1. removal of the parent or one of the children.
2. a reverse to the initial configuration of the myopic view.
3. a downgrade reconfiguration in the myopic view.

These messages form the basic interactions among the agents to configure and maintain the tree overlay links.

5 Architecture and System Control

This section outlines the design of the myopic competitive agent model. It is realized in a 3-layer architecture on which the AETOS agent is based. It also discusses how the AETOS service can be bootstrapped and terminated in a distributed environment.

5.1 Architecture

The myopic competitive agent model illustrated in Section 4 is based on three concepts: (i) the random view, that is, the local search space of the agent, (ii) the myopic view reconfigurations that enable self-organization based on dynamic proximity criteria and (iii) the final establishment of

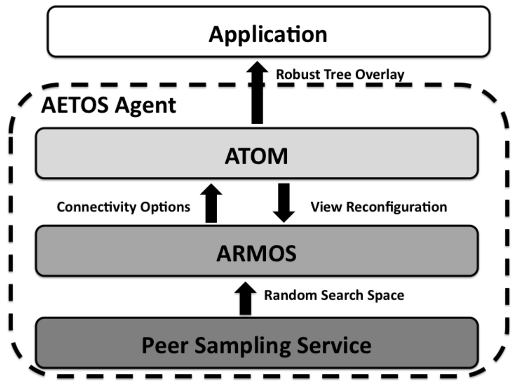

Figure 3. The building and maintenance architecture of a robust tree overlay.

the tree overlay links and triggering of the myopic view reconfigurations. The AETOS service incorporates these concepts in a 3-layer architecture outlined in Figure 3.

The bottom layer is the Peer Sampling Service [7]. It provides the random view to AETOS. The Peer Sampling Service is a gossiping framework that maintains a highly connected dynamic topology. In addition, it gradually removes the disconnected nodes from views of the agents and it can also reflect potential changes in the robustness values r of the agents.

The middle layer is ARMOS, A Rank-based Middleware Overlay Service. ARMOS clusters the agents similarly to T-MAN [6]. However, it uses dynamic and not static proximity criteria as outlined in Section 4.2. In the case of AETOS, the proximity corresponds to the robustness distance d. ARMOS is a realization of the myopic view reconfigurations that the myopic competitive agent model introduces.

The top-layer of AETOS is ATOM, the Adaptive Tree Overlay Management layer. ATOM manages the establishment of the tree overlay links from the connectivity options that ARMOS provides. It also triggers the myopic view reconfigurations in the ARMOS layer. Section 4.3 illustrates the role of ATOM.

Finally, the AETOS agent facilitates and manages these three layers. All come together to provide robust tree overlays to the applications on-demand.

5.2 Bootstrapping and Termination

The process of building and maintaining tree overlays is controlled by the application *on-demand*. This means that the local application instance decides if AETOS should *create* a new tree overlay or *improve* an existing one. Creating and improving tree overlays are the services provided by AETOS to one or more applications.

Controlling a tree overlay requires global knowledge of the topology. For example, AETOS should know when to stop the tree self-organization process upon completion or to improve the topology if nodes have been inserted or departed from the system. For this reason, bootstrapping and terminating AETOS in a distributed manner is challenging.

This paper proposes a fully distributed bootstrapping and termination scheme that is based on a local entity, the *AETOS proxy*. This proxy is an agent that (i) *initializes* the self-organization process, upon request by the local application instance, with the local application requirements, (ii) *monitors* if the local application requirements have been satisfied and (iii) *terminates* the self-organization process and allows AETOS to provide the tree overlay to the application. Figure 4 illustrates the AETOS proxy and how the application interacts locally with the AETOS service.

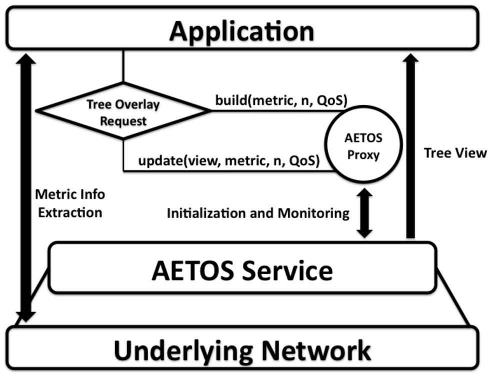

Figure 4. The AETOS proxy and the application interaction with the AETOS service.

At first, the AETOS proxy receives the requests from the application instances together with their requirements and initializes the participation of the local AETOS agent in the self-organization process. The application requirements concern the robustness metric, based on which AETOS sorts the nodes over the tree overlay, and the number of children n that the local application instance can support. There are also some QoS requirements. These include the trade-off between how fast the tree overlay is expected by the application and how optimized (robust) the tree overlay actually is.

In the next step, the proxy monitors this process. When all of the requirements have been fulfilled and there are no changes in the parent-children connections for a given period of time, the AETOS proxy assumes that the system has converged. This period of time is application specific and is determined by the QoS requirements mentioned above.

At the moment of convergence the proxy, (i) stops participation of the agent in the self-organization process by locking the existing parent-children connections and (ii) enables the AETOS service to provide the tree view to the application. Given an improve request, the proxy unlocks the connections and re-enables the participation of the agent in the self-organization process.

This termination approach is similar to the approach proposed in T-MAN [6]. However, in our approach, the application is the one that defines, through its requirements, when the self-organization terminates rather than the underlying AETOS system. The motivation for this decision is that the stability of the tree overlay is evaluated with respect to the application requirements and thus the application must be the one that calls or stops the service.

6 Example

As an illustrative example, measurements have been performed in a small-scale environment of 10 agents that are organized in a robust tree overlay with $n = 2$, i.e., each agent has two children. The agents receive random values in the range 0-100. Each of these local values is the robustness value r of the agent. The agent also retains the random view $\boldsymbol{R}$, the myopic view $|\boldsymbol{M}| = 6$ with $|\boldsymbol{P}| = 2$ and $|\boldsymbol{C}| = 4$ and the tree view $\boldsymbol{T}$.

AETOS runs in discrete rounds. In every round: (i) the Peer Sampling Service protocol updates the random view, (ii) ARMOS updates the myopic view by applying the appropriate view reconfigurations as well, and (iii) one parent and two children requests are sent.

After each round, the connectivity and the order of the agents over the tree are evaluated. The simulation reveals that a fully connected tree is built at the end of the first round. As the view reconfigurations are performed, the agents tend to find the appropriate neighbors. Finally, the system converges to the required hierarchy in the 5th round. Between rounds 2-4, agents in the lower levels of the tree change positions until they find the appropriate neighbors.

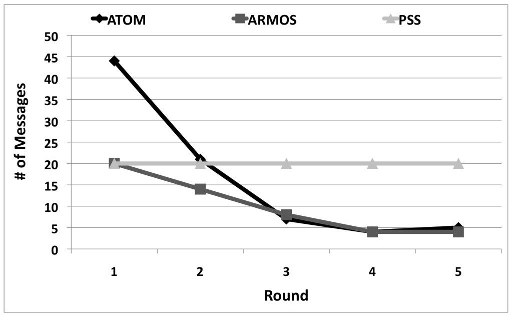

Figure 5. Communication overhead for each layer in the architecture of Figure 3.

One important aspect of AETOS is its communication overhead. Figure 5 illustrates the number of messages caused within each layer during the convergence runtime. The Peer Sampling Service is responsible for retaining the connectivity of the overlay, thus it works independently from the other two layers. For this reason, its communication overhead is constant at 20 messages per round (2 messages x 10 agents = 20 total messages).

In contrast, the two upper layers converge. ARMOS is a similar gossiping protocol with the same message complexity as the Peer Sampling Service and thus in the first round it also starts with the 20 generated messages. As the views are reconfigured and the agents find the appropriate neighbors the myopic views converge to empty ones.

Similarly, the ATOM layer continuously sends requests until it finds a better candidate in the myopic view. Consecutive upgrade and downgrade reconfigurations restrict the view and decrease the probability for finding better neighbors. For this reason, ATOM also converges.

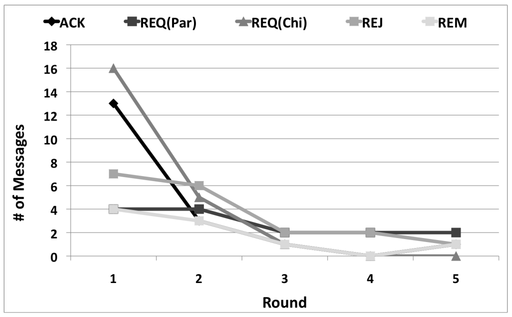

Figure 6. Communication overhead for each type of message in the ATOM layer.

The communication cost of configuring the tree connections has been evaluated. Figure 6 illustrates the number of messages per type in the ATOM layer. The ratio of the children requests over the parent requests appears proportional to the number of children per agent. Furthermore, in the first round, there are many acknowledgments and a significant number of removals but they both converge to the minimum number as the system converges.

Although definite conclusions cannot be reached by this small-scale example, they do provide a positive indication and motivation to further perform large-scale simulations in future work.

7 Conclusions and Future Work

This paper shows that local adaptive and reconfigurable agents can cooperate towards self-organizing themselves in robust hierarchical topologies using AETOS, the Adaptive Epidemic Tree Overlay Service. AETOS builds and maintains robust tree overlays on-demand. It is based on the

AETOS agent that facilitates the myopic competitive agent model. This paper also introduces the AETOS proxy that enables the application to bootstrap and terminate the AETOS service.

In contrast to the related work illustrated in Section 3, AETOS is able to build and maintain robust tree overlays for different application types. AETOS also facilitates the three main characteristics identified in the related work in a generic and effective manner: (i) an abstract robustness metric provided by the application. It can represent one or more possibly conflicting performance metrics. (ii) An underlying support by a proximity-based overlay (ARMOS) and a random overlay (Peer Sampling Service). (iii) Dynamic robust protocols. The Peer Sampling Service retains the connectivity in the lowest level of AETOS in case of failures and ARMOS uses dynamic proximity criteria.

The illustrative example of Section 6 provides a promising positive indication for the effectiveness of AETOS. Further experimentation in large-scale networks has to confirm this. The quality of the tree overlays will be evaluated in certain application types, such as the agent-based energy utilization discussed in [14].

Acknowledgments

This project is supported by the NLnet Foundation `http://www.nlnet.nl`.

References

[1] M. Aldinucci, M. Danelutto, and P. Kilpatrick. Towards hierarchical management of autonomic components: a case study. In *Proc. of Intl. Euromicro PDP 2009: Parallel Distributed and network-based Processing*, Weimar, Germany, Feb. 2009. IEEE.

[2] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding Availability. In *IPTPS*, pages 256–267, 2003.

[3] J. A. Chaudhry and S. Park. Ahsen - autonomic healing-based self management engine for network management in hybrid networks. In *GPC*, pages 193–203, 2007.

[4] K. S. Decker and K. Sycara. Intelligent adaptive information agents. *J. Intell. Inf. Syst.*, 9(3):239–260, 1997.

[5] A. González-Beltrán, P. Milligan, and P. Sage. Range queries over skip tree graphs. *Comput. Commun.*, 31(2):358–374, 2008.

[6] M. Jelasity, A. Montresor, and O. Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Elsevier Computer Networks*, 2009. To appear.

[7] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.

[8] J. Leitao, J. Pereira, and L. Rodrigues. Epidemic Broadcast Trees. In *SRDS '07: Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, pages 301–310, Washington, DC, USA, 2007. IEEE Computer Society.

[9] B. Li, H. Wang, and G. Feng. Adaptive Hierarchical Intrusion Tolerant Model Based on Autonomic Computing. *Security Technology, International Conference on*, 0:137–141, 2008.

[10] M. Litoiu, M. Woodside, and T. Zheng. Hierarchical model-based autonomic control of software systems. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005.

[11] J. Liu and M. Zhou. Tree-assisted gossiping for overlay video distribution. *Multimedia Tools Appl.*, 29(3):211–232, 2006.

[12] R. P. Lopes and J. L. Oliveira. Software agents in network management. In *ICEIS*, pages 674–681, 1999.

[13] E. Pournaras, G. Exarchakos, and N. Antonopoulos. Load-driven neighbourhood reconfiguration of Gnutella overlay. *Computer Communications*, 31(13):3030–3039, 2008.

[14] E. Pournaras, M. Warnier, and F. M. T. Brazier. A Distributed Agent-based Approach to Stabilization of Global Resource Utilization. In *Proceedings of International Conference of Complex Intelligent and Software Intensive Systems (CISIS'09)*, March 2009.

[15] G. Tan, S. A. Jarvis, X. Chen, and D. P. Spooner. Performance Analysis and Improvement of Overlay Construction for Peer-to-Peer Live Streaming. *Simulation*, 82(2):93–106, 2006.

[16] H. Tianfield and R. Unland. Towards self-organization in multi-agent systems and grid computing. *Multiagent Grid Syst.*, 1(2):89–95, 2005.

[17] F. Wang, Y. Xiong, and J. Liu. mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 49, Washington, DC, USA, 2007. IEEE Computer Society.