Self-optimised Tree Overlays using Proximity-driven Self-organised Agents

Evangelos Pournaras, Martijn Warnier and Frances M.T. Brazier

Abstract Hierarchical structures are often deployed in large scale distributed systems to structure communication. Building and maintaining such structures in dynamic environments is challenging. Self-organisation is the approach taken in this chapter. AETOS, the Adaptive Epidemic Tree Overlay Service, provides tree overlays on demand. AETOS uses three local agents to this purpose (i) to translate application requirements to self-organisation requirements, (ii) to self-organise nodes into optimised tree topologies based on these requirements, (iii) to control bootstrapping and termination of self-organisation. The evaluation of AETOS in different simulation settings shows that it provides high connectivity in tree overlays optimised according to application requirements.

1 Introduction

Complex, intelligent, distributed systems in dynamic environments need to adapt continuously. Management is a challenge. Central management of such systems is not often an option: distributed management is required.

Self-management relies on local management at the level of individual systems, and virtual topologies (overlays) to regulate communication between systems, for example to aggregate global knowledge about the state of a system. Hierarchies often provide the structure upon which distributed management is based. Examples of domains of applications for which this holds include DNS, multimedia multicasting [32], energy management [29] and distributed databases [16].

Building and maintaining robust and application-independent hierarchical topologies designed to this purpose is the challenge this chapter addresses, in particular for tree structures. Connectivity in a tree overlay is of key importance. If a node is

Evangelos Pournaras, Martijn Warnier and Frances M.T. Brazier
Delft University of Technology, Dept. of Multi-actor Systems, Section Systems Engineering
e-mail: {e.pournaras,m.e.warnier,f.m.brazier}@tudelft.nl

(temporarily) disconnected, the branches underneath the node are also (temporarily) disconnected from the rest of the system, affecting global performance.

AETOS, the Adaptive Epidemic Tree Overlay Service, is the approach proposed in this chapter. AETOS makes it possible to create self-organised tree topologies that are proactively resilient to failures, and reactively self-heal [9] the structure built. AETOS [30] builds and maintains application-independent robust tree topologies in dynamic distributed environments.

Intelligent software agents are used (i) to translate application requirements to self-organisation requirements, (ii) to self-organise nodes in optimised tree topologies based on these requirements, i.e., reactively reconnecting or rewiring connections to improve robustness, (iii) to control bootstrapping and termination of self-organisation.

Experimental evaluation of the AETOS self-organisation based on connectivity convergence is presented.

This book chapter is outlined as follows: Section 2 outlines application domains in which hierarchical topologies are used. It also illustrates the problem and summarises the contributions of AETOS. Section 3 illustrates related work on robust tree overlays. Section 4 provides a high-level overview of the agent-based approach of AETOS. Sections 5-7 present the three agents of AETOS: the 'application agent', the 'self-organisation agent' and the 'system control agent' respectively. Section 8 illustrates the experimental evaluation of the approach that this book chapter proposes. Finally, Section 9 concludes this chapter and outlines future work.

2 Objectives and contributions

This section discusses the importance of tree topologies for various application domains and identifies the problem of managing application-independent self-organised trees. It also provides an overview of the proposed solution.

2.1 Applications

Tree structures are often used in information management for aggregation, search, dissemination and decision-making. Their complexity is usually bounded to a logarithmic function, or to the number of nodes in the tree structure. They are also used for many other purposes, such as knowledge extraction and visual information systems.

Although the use of trees in centralised systems is typical and has been extensively studied, using and maintaining a tree structure in a decentralised system is the challenge this chapter addresses. Introducing a dynamic tree structure for distributed systems potentially enables effective self-management. As an example, EPOS, the Energy Plan Overlay Self-stabilisation system [29], performs stabilisation in the

global energy utilisation of thermostatically controlled devices. These devices are interconnected and organised in a tree overlay. Based on this structure, they perform local aggregation and decision-making of the local allocated energy they consume for a period of time. EPOS achieves the minimisation or the reverse of the deviations in the global energy utilisation making it possible (in theory) for power systems to become more robust and flexible to dynamic environments.

IP multicast appears to have many limitations in its adoption and deployment [11], especially concerning the average end user. These limitations are related to its routing complexity and scalability. Application-level multicast has emerged as a new approach for distributing multimedia content. The majority of methods based on application-level multicast use tree overlays. Organising nodes in a loop-free structure can make distribution of content effective and potentially scalable compared to mesh-based overlays. Extensive comparisons of various application-level multicast approaches are illustrated in [6, 24, 32].

Tree structures integrated with skip lists [31] in skip tree graphs benefit distributed database operations such as range queries [16]. In the same domain, tree overlays, introduced as a distributed indexing scheme, enhance resource searching and sharing [37]. Finally, super-peer topologies model distributed systems in a hierarchical fashion that can reflect the heterogeneity of different node capabilities, such as storing capacity, processing power, connectivity or bandwidth. This provides the potential for various application optimisations, such as load-balancing. Such an option is explored in ERGO, the Enhanced Reconfigurable Gnutella Overlay [28]. ERGO rewires nodes with high outgoing load to nodes with low incoming load. This is achieved through the interaction of lower-level nodes with higher-level *virtual server* nodes responsible for load-balancing.

2.2 Problem statement

As stated above building and especially maintaining tree overlays, optimised for different applications, is the problem this chapter addresses. The main aspects of this problem are: self-organisation, self-optimisation, and application independence.

Self-organisation: Nodes should be able to self-organise themselves in a tree overlay using local knowledge. Often, as explained in Section 6, this knowledge is a partial view of the distributed environment. Nodes should be able to connect to other nodes and potentially rewire connections without introducing loops or violating restrictions such as their capacity.

Self-optimisation: The satisfaction of application requirements when using tree overlays is usually an optimisation problem, as described in detail in Section 3. Nodes should connect to the appropriate neighbours to maximise their applications' utilities. Note that applications most often use topologically different tree overlays as they are based on different performance metrics. For example, in EPOS [29] availability of nodes, is the metric used to identify disconnected nodes. Note that

availability is a metric measurable in many applications, such as Overnet [5]. Similarly, application-level multicast tree overlays are based on metrics such as latency, bandwidth, node degrees and other. Section 3 discusses related approaches.

Application independence: Providing a dedicated self-organisation mechanism for each application can be costly. Distributed systems are dynamic, and support applications that interact with each other.

Figure 1 illustrates the concept of different tree overlays on the same physical network. Each overlay is used by a different application. A physical host corresponds to one (or more) overlay host in an overlay network. Note that the position of an overlay host in a tree overlay is different for each application overlay. This is because the mapping between a physical host and the respective overlay hosts depends on the application requirements and optimisation metrics.

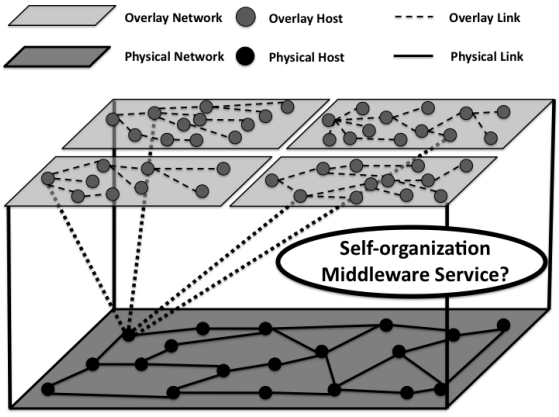

Fig. 1 Each overlay application requires a different optimised tree topology. A physical node corresponds to overlay nodes placed in different position in the tree. In this case, the building and maintenance is application-dependent. A self-organisation middleware service for tree overlays could solve this problem.

Each application, for every overlay, is responsible for building and maintaining the tree structure. A generic self-organisation middleware service can decouple the building and maintenance from the application. This chapter focuses on the problem of how such a service can be modelled and how it can function in large-scale distributed systems, such as virtual networks over physical infrastructures or large-scale multi-agent systems.

2.3 System Overview

The contribution of this chapter is to propose a self-organisation service for tree overlays, named AETOS, the Adaptive Epidemic Tree Overlay Service. AETOS is an agent-based system positioned between the overlay applications and the physical network. Figure 2 illustrates the position and the interactions of AETOS in a distributed environment.

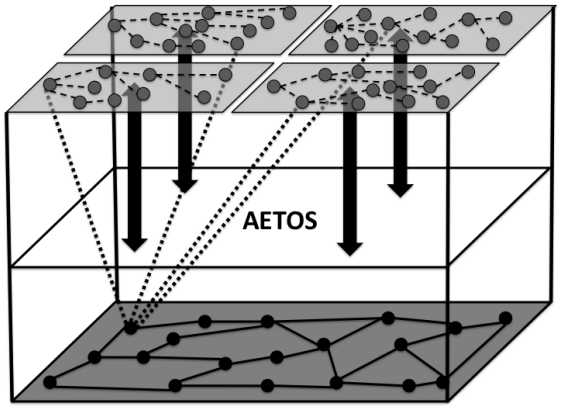

Fig. 2 AETOS is placed as a middleware service in a distributed environment. It undertakes the role of building and maintaining different tree overlays for different applications.

Overlay nodes have direct access to information from the physical network. This information, together with other application requirements, is passed to the AETOS layer. Based on this information, AETOS builds and maintains on-demand different tree overlays for each application.

AETOS achieves the abstraction of local application requirements to local self-organisation requirements. Nodes are dynamically self-organised to tree topologies on-demand based on their proximity derived from the local application requirements. Bootstrapping and termination of self-organisation is managed locally.

The experimental evaluation of Section 8 reveals that AETOS achieves high connectivity of tree overlays in various experimental settings. This chapter also investigates the influence of various factors in the cost-effectiveness of AETOS.

3 Related work

This section presents related literature on self-organised and robust tree overlays, focusing in particular on: (i) *application domain*, (ii) *optimisation metrics*, (iii) *complementary overlays*, (iv) *build and maintenance*, (v) *decentralisation level*, and (vi) *proactiveness vs. reactiveness*. Open issues are discussed and outlined, illustrating the need for a self-organisation service, such as AETOS.

3.1 Literature review

This section provides an overview of related work in the area of robust self-organised tree overlays, on the basis of the six areas distinguished above.

Application domain: The majority of the methods concerns tree overlays for application-level multicast, video streaming and real-time applications, as underlined in Section 2.1. Multimedia applications require effective broadcast for guaranteeing high QoS. The root is usually the provider of the multimedia content and the rest of the nodes are end-users that receive this content. They contribute resources in the system by forwarding the content they receive from their parents to their children. In database systems, complex queries can be performed over peer-to-peer tree overlays [17, 21]. Maintaining a robust and reliable topology is crucial for data consistency and knowledge extraction from the network. Publish-subscribe systems [10, 14] also benefit from tree overlays as they can be used to minimise the changes in the event routing. Other domains in which tree overlays are deployed are grid environments for task allocation and scheduling [8, 12] and sensor networks for data collection [12]. Note that, although these applications vary significantly and have different requirements, the common goal of all of them is shared: to maximise its utility by performing operations over application specific optimised tree overlays.

Optimisation metrics: Robustness in tree overlays can be achieved by single or multi-metric objective optimisations in the self-organisation process. Various optimisation metrics, related to the application type, are used to organise nodes in an appropriate tree overlay for the application. Some of the most common optimisation metrics include delay, bandwidth, node degree, uptime and other related optimisation metrics. Note that these metrics are usually related to the underlying physical network, in order for applications to maximise the utilisation of available network resources. In [12], trees are optimised by considering the number of hops and the eccentricity, both metrics related to the experienced delay in the underlying physical network. Bandwidth and node degree are associated in [13]. The number of children influences the bandwidth consumed from their parents in multicasting applications. In contrast, these two metrics are assumed independent in mTreebone [36]. This assumption is valid when other applications consume part of the available bandwidth. Node degree influences the topology and the optimisation of the application. Trees can be balanced, fat (wide) or long ones. Tree topologies, such as the latter two ones, can be integrated by exploiting trade-offs between opposing performance metrics, i.e., uptime and bandwidth [12, 32]. Similar trade-offs are explored in [22] as well. In these cases, multi-metric objective optimisations are applied by combining or weighting two or more metrics. For example, in [32], bandwidth and uptime are combined by computing their product, the 'service capability contribution'. Weighting schemes between 'path weight'-'hop count' and 'delay penalty'-'resource usage' are proposed in [12] and [22] respectively. Finally, the sojourn probability [19] and the joining times of nodes [23] can be used for the optimisation of tree overlays.

Complementary overlays: Some multicast applications maintain tree overlays over mesh ones. RESMO [22] is a minimum delay, minimum resource usage spanning tree over a mesh overlay. RESMO selects links from the mesh overlay with sufficient bandwidth. mTreeBone [36] is based on the similar concept of selecting stable nodes from the mesh overlay to build a backbone tree overlay. MeshTree [33] is a combination of a tree and mesh overlay by inserting shortcut links between

the nodes of the tree overlay. Such a link redundancy is used in other approaches as well. For example, BATON* [17] additionally inserts adjacent and neighbour links between nodes of the tree for acquiring additional robustness. TAG [23] and PRM [4] use gossiping and random links respectively to deal with data loss and discontinuous playback in real-time applications. Gossiping is used to support trees in GoCast [34] as well. Other underlying complementary overlays that appear in literature are DHTs [10]. However, DHTs, are not resilient to failures and require maintenance.

Build and maintenance: In the investigated approaches in this section, the building process of tree overlays is either integrated with their maintenance, for example in [20], or it serves as a bootstrapping mechanism for the maintenance that follows, e.g. [3]. The main method used for building a tree, or an initial version of it, is the consecutive joins to candidate parents and children [33] or to the leaves of the tree [32]. These candidates are derived randomly [19] or from their proximity to the local node [23]. After the initial joins, nodes either aim to improve their position in the tree or they cooperate to optimise the tree topology. In the first case, nodes perform shift-up operations [32] by moving to an upper level in the tree, whereas, in the second case, a parent and one of its children swap their positions [1, 17]. Plumtree [20] combines eager and lazy push gossiping strategies to build and maintain a tree overlay. In [10], the node-key mapping of the underlying DHT is used to form the tree overlay. Alternative methods for the distributed building of a tree overlay include the top-down approach proposed in [22], the Bellman Ford [12] and Prim's algorithm [13]. Furthermore, nodes can monitor the connectivity of their neighbours by sending heartbeats [21, 23]. In case of a failure, they try to connect with another node. TreeOpt [27] improves the tree connectivity by performing two types of children moves as an evolutionary optimisation of the tree overlay. In [14], a candidate parent is selected by applying and combining different repair strategies related to the application requirements. Similarly in [8], ancestor lists are retained in case of failures. In contrast, the proposed approach in [13] defines a 'parent-to-be' for every node (besides the root) before a failure occurs. Thus, repair is faster. Other techniques propose link redundancy in order to satisfy alternative connectivity in case of failures [17, 36]. Load-balancing also supports the maintenance of tree overlays by aiming to retain the load in the nodes between root and leaves equal [17, 21].

Decentralisation level: Among the illustrated approaches, there are some hybrid schemes for topology management. DPOCS [1] is based on the 'overlay control server (OCS)' that assists nodes to join the multicast groups. OMNI [3] and TAG [23] follow a similar concept by introducing the 'multicast server nodes (MSNs)' and a 'content server' respectively. mTreebone [36] utilises only stable nodes for video multicasting. BulkTree [2] groups the nodes to 'super-nodes' in order to increase the stability of the tree. Finally, the approach of [19] is based on a video broadcasting source node that centrally collects and calculates statistics. This information is used during for the self-organisation process.

Proactiveness vs. reactiveness: Methods that apply a sorting of the nodes, within the tree overlay, for application optimisation are considered proactive. For example, the use of 'service capability contribution' [32] as a metric for combining a bandwidth-ordered and a time-ordered tree makes the multicasting proactively more robust and efficient. Methods that use complementary overlays [20, 33, 34, 36], link and data redundancy [4, 17, 21, 23] are also regarded as proactive approaches. In this case, proactiveness is applied indirectly and externally, by other overlay support. In [13], a highly proactive approach is proposed. Nodes calculate the new parents for their children before a failure occurs and without violating the node degrees. In contrast, reactive nodes monitor their neighbours [21] and perform reconnections to other nodes when a failure occurs. Usually the selection of the nodes is based on various strategies [14] that balance performance trade-offs. TAG [23] can be considered to be a reactive system as it operates in highly dynamic environments with real-time constraints. Proactive approaches benefit from the fact that they aim to decrease the complexity and time of the repair actions or the impact of failures. However, proactive approaches introduce: (i) a usually constant but (ii) significant communication and processing cost.

3.2 Open Issues

The conclusions from the literature review are in line with the AETOS motivation discussed in Section 2. Robust and self-organised tree overlays depend on the application domain. Most optimisations consider metrics related to physical networks. It is unclear how other higher-level application-related metrics could influence and change the proposed self-organisation methods. In addition, related work reveals that different applications have different trade-offs. Therefore, combining or weighting multiple optimisation metrics, in an application-independent way, is challenging.

Dynamic protocols, i.e., gossiping protocols, and complementary overlays are effective in many cases. Usually, they are not required to be dedicated for the self-organisation of trees but rather can be reused as existing services in distributed environments. The role of such complementary overlays should be further studied and clarified. The same holds for the proactive or reactive approaches of self-organised systems. Although high proactiveness results in high robustness and resilience to failures, the required cost can be significant with relatively low benefits for the application. Future work should explore the level of proactiveness and reactiveness required for building robust tree overlays for a wide range of applications.

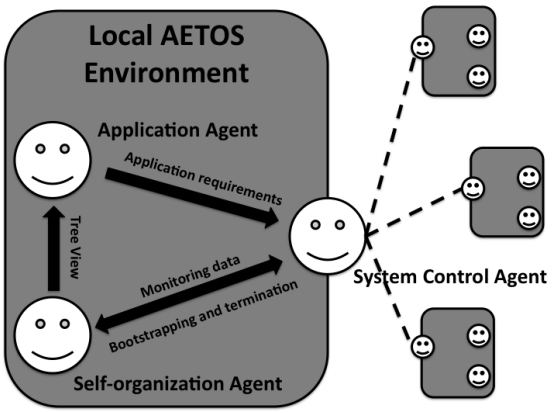

Fig. 3 AETOS is based on three agents that interact locally. The 'application agent' provides the application requirements to the 'system control agent'. The latter bootstraps self-organisation, monitors the 'self-organisation agent' and finally terminates self-organisation. When the 'self-organisation agent' is terminated, it makes the parent and the children neighbours available to the 'application agent'.

4 Approach

The multi-agent systems paradigm, in which individual autonomous agents interact with each other to accomplish their goals, has been successfully applied to management and self-organisation of distributed systems [7, 25, 35]. AETOS, a service for building and maintaining on-demand and application-independent robust tree overlays, deploys agents for the purpose of self-organisation.

Overlay hosts (nodes) are the local environment of AETOS agents. These agents act solely within their local environment (and do not migrate).

AETOS agents have (i) *local knowledge*, (ii) *local components* that manage the local knowledge and execute *local tasks* and (iii) *local layers* of components that create a hierarchy in the information flow. The AETOS service is provided by these agents (and their interaction).

Three local agents participate in AETOS: (i) *the application agent*, (ii) *the self-organisation agent* and (iii) *the system control agent*. Figure 3 illustrates how they interact in the local AETOS environment.

The principle interactions among AETOS agents are outlined as follows:
The 'application agent' abstracts the application-specific requirements to application-independent self-organisation requirements by providing a common interface between applications and AETOS. The 'system control agent' turns the self-organisation requirements to self-organisation parameters that the 'self-organisation agent' understands. It then bootstraps, monitors and finally terminates the self-organisation process. Upon termination, the 'self-organisation agent' makes the parent and children neighbours available to the 'application agent' which makes them accessible to the application.

Note that, Figure 3 depicts interaction between the 'system control agent' and other 'system control agents' outside its local environment. Such interaction is optional and beyond the focus of this paper.

5 Application agent

The 'application agent' provides a generic interface for managing *application requirements* between AETOS and different applications. Note that, these requirements are parametrisation settings that make an application work effectively. There is one 'application agent' per application instance. The set of application requirements, denoted by **A**, managed by the 'application agent' are the following:

Robustness (r): This is the abstraction of the optimisation metric on which the self-organisation is based. It can concern any of the previously identified metrics mentioned in Section 2.2 and 3.1. If the application utilises more than one optimisation metrics, the application itself must apply a weighting scheme, or function to derive the abstract robustness r. Robustness is assumed to be a decimal number.

Node degree (n): The node degree concerns the number of neighbours for each application instance. It denotes the available resources the application reserves for the tree overlay.

Expected response time (t_r): This is the time period in which AETOS should return the tree neighbours to the application instance. Higher response times allow better topology optimisations. Section 7 explains the use of this parameter by the 'system control agent'.

Note that the above application requirements are the local knowledge of the 'application agent'. The executed tasks are the following:

Register: The 'application agent' contacts the 'system control agent' and sends (i) its identifier and (ii) a new tree overlay identifier, to register a new tree overlay in the AETOS service. This information is finally stored in the 'self-organisation agent' together with the reserved space for the tree neighbours.

Build: This task concerns the creation and maintenance of a tree overlay. It enables on-demand self-organisation. The 'application agent' sends (i) its identifier, (ii) the tree overlay identifier and (iii) the set of application requirements **A** to the 'system control agent'. If the utilised tree overlay does not meet the expectations of the application, this task is executed again.

Connect: When the set of tree neighbours is received from the self-organisation agent, the set is delivered to the application that finally establishes the connections.

Unregister: The 'application agent' contacts the 'system control agent' and sends a tree overlay identifier. The self-organisation for this overlay terminates and all the information related to this overlay is removed from the 'self-organisation agent'.

By implementing an 'application agent' that incorporates the knowledge and the tasks above, applications have access to the AETOS service.

6 Self-organisation agent

In AETOS, each node has one local 'self-organisation agent'. The 'self-organisation agent' forms the core of the AETOS system. The self-organisation agent's knowledge, components and 3-layered service architecture are presented in more detail below.

6.1 Knowledge

The 'self-organisation agent' has different *partial views* of its distributed environment. A partial view is a list of a finite number of other *node descriptors*. A node descriptor contains information related to the node and its applications, such as its address, connection port, overlay identifier and robustness r. A node descriptor gives the fundamental knowledge which forms the basis for communication between 'self-organisation agents'. The overlay identifier that belongs to a node descriptor received is used by the 'self-organisation agent' to match and extract the respective overlay knowledge that holds locally. Each 'self-organisation agent' has 3 partial views: the *random view*, the *proximity view* and the *tree view*, each described below.

Random View (R): The random view contains the primary knowledge and search space of the 'self-organisation agent'. It consists of a collection of random node descriptors from the distributed environment. Note that, the random view is dynamic and changes continuously. This local knowledge creates a global random graph for all overlay hosts. The maintenance and the dynamic changes of the random view are explained in Section 6.2.

Proximity View (M): The proximity view contains nodes with close proximity to the local node. Proximity is derived by calculating the ranking distance between two nodes. In AETOS, rank values refer to the robustness values r. Therefore, the *robustness distance* between an agent x and an agent y is $d = |r_x - r_y|$. The search space for filling the proximity view is the random view. However, it is also filled by enabling close proximity nodes to exchange neighbours (gossip) and further discover each other faster. Section 6.2 illustrates this option. Finally, the proximity view is dynamic and reconfigurable. This means that the ranking function can potentially change by reconfiguring the view appropriately. This aspect is explained in detail in Section 6.2.

The neighbours of a node in the tree hierarchy are split in two levels, the parent and the children. This concept is applied in the proximity view as well. Two sets of neighbours are defined: (i) the *candidate parents* (**P**) and (ii) the *candidate children* (**C**) such that $\mathbf{M} = \mathbf{P} \cup \mathbf{C}$. Note that the sets are sorted according to robustness r of the node descriptors.

Tree View (T): The tree view is a sorted set with the parent and the children of the local node in the tree overlay. The search space for filling the tree view is the prox-

imity view. The tree view is the one that is provided at the end of self-organisation process to the 'application agent'.

The above views are partial. Their length is a predefined system parameter and depends on the capacity of nodes and on the size of the whole system. For large-scale systems with thousands of nodes, $|\mathbf{R}| \approx 50$ [26]. For the proximity view, a similar scheme is proposed with $|\mathbf{M}| \leq |\mathbf{R}|$. The length of the tree view is $|\mathbf{T}| = n$.

The ratio of the length of the candidate children set over the length of candidate parents set ($\frac{|\mathbf{C}|}{|\mathbf{P}|}$) is proportional to the number of children $c = n - 1$. For example, if $|\mathbf{M}| = 12$ and $c = 3$ then $|\mathbf{C}| = 9$ and $|\mathbf{P}| = 3$. This guarantees that the search space for children and the parent is proportional.

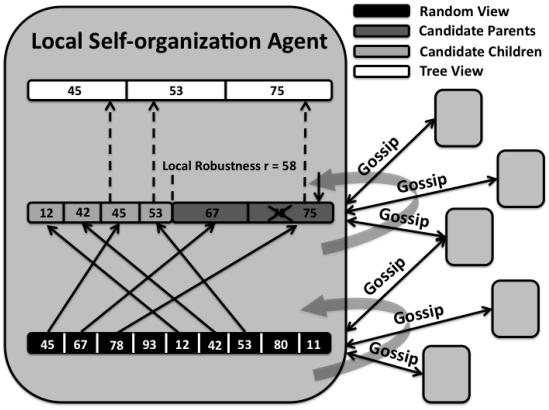

Fig. 4 The fundamental knowledge of a 'self-organisation agent' is based on 3 views: (i) the random view, (ii) the proximity view and (iii) the tree view. The proximity view is filled by random samples and close-proximity neighbours discovered through gossiping. The nodes with the highest robustness in the proximity view are the potential neighbours in the final tree view.

Figure 4 illustrates an example of information flow among the views in a self-organisation agent. The proximity of the local random samples from the random view is calculated and the closest neighbours are inserted in the proximity view. Other close-proximity neighbours are discovered through gossiping. Finally, the candidate neighbours with the highest robustness are acquired for tree neighbours. Upon success, they are inserted in the tree view. Section 6.2 provides detailed information about the local interactions and tasks executed by the 'self-organisation agent'.

6.2 Components

The local knowledge and tasks of the 'self-organisation agent' are facilitated in the following components. Figure 6 outlines these components and their interactions.

Proximity Manager: It holds the proximity view. It interacts with the 'proximity sampling' component and the 'reconfiguration manager' component to update

and improve the proximity view. Periodically, it informs the tree manager about the candidate neighbours with the higher robustness r in its proximity view.

Random Sampling: This component maintains the random view. This view is updated through a gossiping protocol, that is the peer sampling service [26]. With the peer sampling service, nodes continuously have random samples of the whole distributed environment and refresh old nodes with new ones. Gossiping creates a dynamic robust overlay on which the tree overlay is based. Readers are referred to [26] for details concerning the peer sampling service.

Proximity Sampling: This is the component that realises the gossiping among close-proximity nodes as Figure 4 illustrates. 'Random sampling' discovers close-proximity nodes from random samples. In contrast, 'proximity sampling' further discovers candidate neighbours by exchanging node descriptors between close-proximity nodes. The process of such a gossiping protocol is described in detail in [18]. 'Proximity sampling' interacts with the 'proximity manager' to update the proximity view with new candidate parents or children. Note that, this component is used to make the system converge faster to the required tree topology.

Reconfiguration Manager: The proximity view is not static but rather dynamic and reconfigurable. This means that the ranking function is defined in a dynamic range of robustness values which form a subset of the whole range of values in the proximity view. The 'reconfiguration manager' accesses the 'proximity manager' and is responsible for triggering a number of reconfigurations to the proximity view.

The ranges of robustness values for candidate parents and children are examined below. Let M be the range of the whole set of robustness values that node descriptors contain. All of the indexes refer to robustness values in the proximity view: (i) l points to robustness value of the local node descriptor. (ii) A potential parent p belongs to the candidate parents range P such that $p \in P = [l+1, p_{max}]$. Similarly, (iii) the potential children $c_1 < c_2 < ... < c_n$, with n the number of children, point to the candidate children range C such that $\{c_1, c_2, ..., c_n\} \in C = [c_{min}, l-1]$. Figure 5a illustrates the initial ranges of candidate neighbouring sets. The 'reconfiguration manager' can perform the following reconfigurations:

1. **Initialising Configuration**: the ranges of the candidate neighbours are configured as $P = [l+1, p_{max}]$ and $C = [c_{min}, l-1]$ respectively. The node descriptor with the higher robustness r in each candidate set is the potential child or parent respectively. In this case $p = p_{max}$ and $c_i = l-1$, for the ith potential child.
2. **Upgrade Reconfiguration**: the 'self-organisation agent' has already found a parent or its children and it seeks to connect with more robust nodes. To achieve this, it binds the starting point of its view to the robustness values of the selected nodes and fills the view with more robust node descriptors. The candidate parents range is reconfigured as $P = [p+1, p_{max}]$ and the children candidate range as $C = [c_1+1, l-1]$. Figure 5b depicts the upgrade reconfiguration.
3. **Downgrade Reconfiguration**: if a previously selected candidate neighbour has rejected the connection, the view is updated with less robust nodes. In this case, the candidate ranges are updated as $P = [l+1, p_{max}-1]$ and $C = [c_{min}, l-2]$ re-

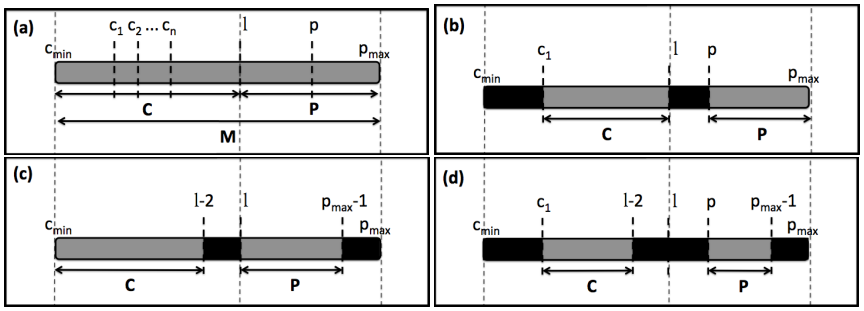

Fig. 5 The parent and children candidates in the proximity view. (a) initial proximity view, (b) after an upgrade reconfiguration, (c) after a downgrade reconfiguration, (d) applying an upgrade and a downgrade reconfiguration.

spectively. Figure 5c 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.

The 'reconfiguration manager' has the option to switch from a downgrade or upgrade configuration back to the initial one. Furthermore, the proximity view can be a result of both an upgrade and a downgrade reconfiguration. Figure 5d illustrates an example of this case. Any applied reconfiguration keeps the length of the proximity view equal or lower than the initial maximum length.

Tree Manager: The Tree Manager manages the connectivity of the tree overlay and it interacts with other nodes to establish the parent and children connections. The interactions are based on the exchange of 4 messages: (i) the *request* of a parent or child connection, (ii) the *acknowledgement* of a request, (iii) the *rejection* of a request and (iv) the *removal* of a parent or child connection.

In its active state, the 'tree manager' periodically accesses the 'proximity manager' and receives the candidate parent and child with the highest robustness r. It sends a 'parent and child request' to each of them respectively. If the 'proximity manager' cannot provide candidate neighbours to the 'tree manager' for a prespecified period of time, it reports this information to the 'reaction manager'.

The passive state of the 'tree manager' defines the appropriate reactions to the messages received. For a 'parent or child request', the reactions are the following:

1. It checks if the robustness r of the two communicating nodes 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 'tree manager' sends a 'rejection' message to the requesting agent with information about the value of local robustness.
2. If there are no inconsistencies, the 'tree manager' either

   a. updates and inserts the node that sent the 'parent/child request' in its tree view. In this case, the 'tree manager' replies with an 'acknowledgement'. If

the update of the tree view is performed by replacing an existing node descriptor with one with higher robustness, then a 'removal' message is sent to the replaced node. Or,

b. it rejects the request and a 'rejection' message is sent. In this case, the existing parent or children are more robust than the node that sent the request.

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

3. a report is sent to the local 'reaction manager'.

If the 'tree manager' receives an 'acknowledgement' of its request it performs:

1. an update of its tree view by inserting the new neighbour. If the update is a replacement, it sends a 'removal' messages to the replaced node.
2. a report to the local 'reaction manager'.

The 'rejection' message triggers the following:

1. a report to the local 'reaction manager'.

Finally, in case of a 'removal' message, 'tree manager' performs:

1. removal of the parent or one of the children.
2. a report to the local 'reaction manager'.

These messages form the basic interactions among the AETOS agents to configure the tree overlay connections.

Reaction Manager: It receives reports from the 'tree manager' concerning the configuration of the tree connections. Based on these reports, it triggers the appropriate reconfigurations in the 'reconfiguration manager'.

An upgrade reconfiguration is triggered when a new parent or the last child is added in the candidate parents or children respectively. A downgrade reconfiguration is applied when a 'parent of child request' is rejected or a removal is performed in a parent or child. Finally, the initialising reconfiguration is performed before the upgrade or downgrade reconfigurations to overwrite the old ones.

6.3 Service Layer architecture

The interactions of the components in the 'self-organisation agent' can be outlined in the following 3-layer hierarchy:

PAROS: The *ProActive Robust Overlay Sampling* is the underlying overlay that provides high robustness in the tree overlay. It guarantees that the network remains connected and it is not clustered due to node departures or failures.

ARMOS: The *Adaptive Rank-based Middleware Overlay Service* is a proximity-driven reconfigurable overlay. It incorporates the 'tree manager', the 'proximity sampling' and the 'reconfiguration manager'. It is based on PAROS and supports the connectivity of the tree overlay by providing candidate neighbours.

ATOM: The *Adaptive Tree Overlay Management* is responsible for configuring the tree connections and provides feedback to ARMOS for improving the candidate neighbours.

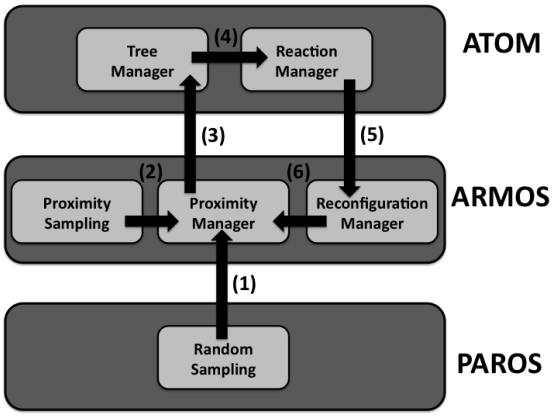

Fig. 6 The 3-layer hierarchical interactions of the components within the 'self-organisation agent'. The numbers denote the sequence of interactions between the components. The arrows (3), (4), (5) and (6) depict the feedback loop which forms the core of adaptivity in AETOS.

Figure 6 outlines the 3-layer hierarchy and the components of the 'self-organisation agent'. The sequence of interactions is as follows: (1) 'random sampling' provides periodically random samples to the 'proximity manager'. From these random samples, the ones with close proximity are selected and stored in the proximity view. (2) 'proximity sampling' exchanges node descriptors with close proximity nodes for improving the proximity view. (3) Periodically, the 'proximity manager provides the best candidate neighbours to the 'tree manager'. The latter interacts with these candidates to establish tree connections. (4) The result of these interactions is reported to the 'reaction manager' (5) that triggers the appropriate reconfigurations in the 'reconfiguration manager'. (6) Finally, the proximity view is reconfigured and new candidate neighbours can be provided to the 'tree manager'.

Note that the feedback loop between the 'proximity manager', 'tree manager', 'reaction manager' and 'reconfiguration manager' forms the core of adaptivity in AETOS.

7 System control agent

The 'system control agent' acts as a proxy between the 'application agent' and 'self-organisation agent'. It keeps information about the registered overlays and provides this information to the 'self-organisation agent'. It also receives the application requirements for each overlay and monitors the self-organisation process. With this information, it can control locally the bootstrapping and termination of the self-organisation.

Bootstrapping: The 'system control agent' initially guarantees that the robustness values are unique. This is achieved by assigning a unique comparable random number in the robustness value r. It then feeds the robustness r and the number of children $c = n - 1$ to the 'self-organisation agent'. Therefore, the 'self-organisation agent' is able to start executing its component tasks.

Termination: Termination is based on the expected response time t_r. The 'system control agent' monitors the 'self-organisation agent'. When the runtime exceeds the t_r, it terminates the self-organisation.

At this moment of local convergence the agent, (i) stops the participation of the agent in the self-organisation process and (ii) enables the 'tree manager' to provide the tree view to the application. Note that, the node can be still contacted when is not participating in the self-organisation. In this case, it notifies the node about its current terminated state.

In this termination approach, the application is the one that defines, through its requirements, when the self-organisation 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 it must be the one that influences the termination of the self-organisation.

8 Evaluation of the proposed approach

AETOS is implemented and evaluated in ProtoPeer [15], an asynchronous simulation platform for large-scale distributed systems. ProtoPeer provides a generic interface for enabling the step from single-machine to multiple-machines simulation and finally to live deployment.

This section focuses on the evaluation of the 'self-organisation agent'. The goal of the evaluation is to reveal the cost-effectiveness of AETOS in the connectivity of two different tree topologies. In this section, connectivity *refers to the percentage of the total number of nodes connected to the main tree. The convergence of connectivity is investigated under varying length of the random view and two different network sizes.*

The input settings in the ProtoPeer simulation environment represent the 'application agent'. The 'self-organisation agent' is implemented as three services or 'peerlets' in ProtoPeer terminology. Each service corresponds to a layer in the ar-

chitecture of Figure 6. In the first layer, the peer sampling service [26] is the implementation of the 'random sampling' component. In the middle layer, the 'proximity manager' and the 'reconfiguration manager' are implemented. The implementation of 'proximity sampling' is part of ongoing work and is not part of AETOS in the results illustrated in this section. However, the implications of this missing component are discussed in this section. The two components of the ATOM layer, the 'tree manager' and 'reaction manager', are facilitated in a peerlet of the 'self-organisation agent'. Finally, the evaluation of the bootstrapping and termination by the 'system control agent' is part of future work.

8.1 Simulation settings

Three group of experiments are performed in two different simulation environments. Table 1 outlines the simulation parametrisation in these two environments. 'Simulation environment 1' has 121 nodes. 'Simulation environment 2', a larger-scale network, has 1093 nodes. The first two groups of experiments run for 2500 iterations and the third for 400. The latter group of experiments runs for fewer iterations due to restrictive memory scalability of the ProtoPeer measurement infrastructure. The ProtoPeer environment supports bootstrapping of the system in a ring topology in the first 6 iterations from which the peer sampling service and the components in the higher levels are initialised.

Table 1 Simulation environments

Parameter	Simulation Environment 1	Simulation Environment 2
Number of nodes ($\mathbf{N}$)	121	1093
Number of children (c)	3-5	3-5
View selection policy	swapper	swapper
Random view length ($\|\mathbf{R}\|$)	4-20	40
Candidate parents length ($\|\mathbf{P}\|$)	2	3
Candidate children length ($\|\mathbf{C}\|$)	4	5
Number of iterations	2500	400
Requests frequency	2 per iteration	2 per iteration

The *swapper* selection policy used within the peer sampling service [26] is used to increase randomness in the local node samples. In 'simulation environment 1' the length of the random view $\mathbf{R}$ is varied between 4-20. In 'simulation environment 2' the length of the random view $\mathbf{R}$ is fixed to 40. The length of the view of candidate parents is chosen to be smaller than the view of the candidate children and is $|\mathbf{P}| = 2$, $|\mathbf{P}| = 4$ for the first and $|\mathbf{P}| = 3$, $|\mathbf{P}| = 5$ for the second simulation environment.

Finally, nodes are organised in two tree topologies: (1) a tree for which the number of children to which the 'application agents' try to connect is 3 and (2) a tree for which the number of children to which the 'application agents' try to connect

is 5. As a result with a fixed number of nodes, the trees have different number of levels. The robustness r assigned to the 'self-organisation agent' is a unique random number between 0 and 100. Note that in every iteration the 'self-organisation agent' potentially sends one parent and one child request, thus the frequency of requests is 2 per iteration.

8.2 Results

The first group of experiments runs in 'simulation environment 1' in which the number of children to which agents aim to connect is equal to 3. Figure 7(a) illustrates the convergence of connectivity by varying the length of the random view. Figure 7(b)-(e) depicts the communication cost of AETOS expressed in the number of messages generated by the ATOM layer of the 'self-organisation agent'.

The second group of experiments also runs in 'simulation environment 1' but in this case the number of children to which agents aim to connect is equal to 5. Figure 8(a) illustrates the convergence of the connectivity by varying the length of the random view. Figure 8(b)-(e) depicts the communication cost of AETOS expressed in the number of messages generated by the ATOM layer of the 'self-organisation agent'.

Finally, the last group of experiments runs in 'simulation environment 2' for $c = 3$ and $c = 5$. Figure 9 illustrates the connectivity convergence in this settings.

In summary, the above results show that AETOS can achieve a high degree of connectivity in both simulation environments. AETOS converges to 90% connectivity in less than 150 iterations. An exception is the case of 'simulation environment 2' with $c = 3$, in which connectivity approaches 60% in the 400th iteration. Section 8.3 explains in detail the behaviour of AETOS in these simulation experiments.

8.3 Discussion of Experimental Results

The results reveal that a certain percentage of connectivity can be achieved within relatively few iterations. For example, 50% connectivity can be achieved in less than 100 iterations in 'simulation environment 1' and between 150-400 iterations in 'simulation environment 2'. In contrast, for connectivity higher than 98% AETOS convergence lasts much longer, requiring 436 iterations in 'simulation environment 1'. In this environment, increasing connectivity from 90% to 98% requires more than 250 additional iterations. This effect is more significant in 'simulation environment 2' in which connectivity seems to converge 10%-30% more slowly than 'simulation environment 1'. The peer sampling service provides a bounded random search space and thus convergence speed decreases as the size of the network or the topology complexity increases. The connectivity jump from 50% to about 80% in

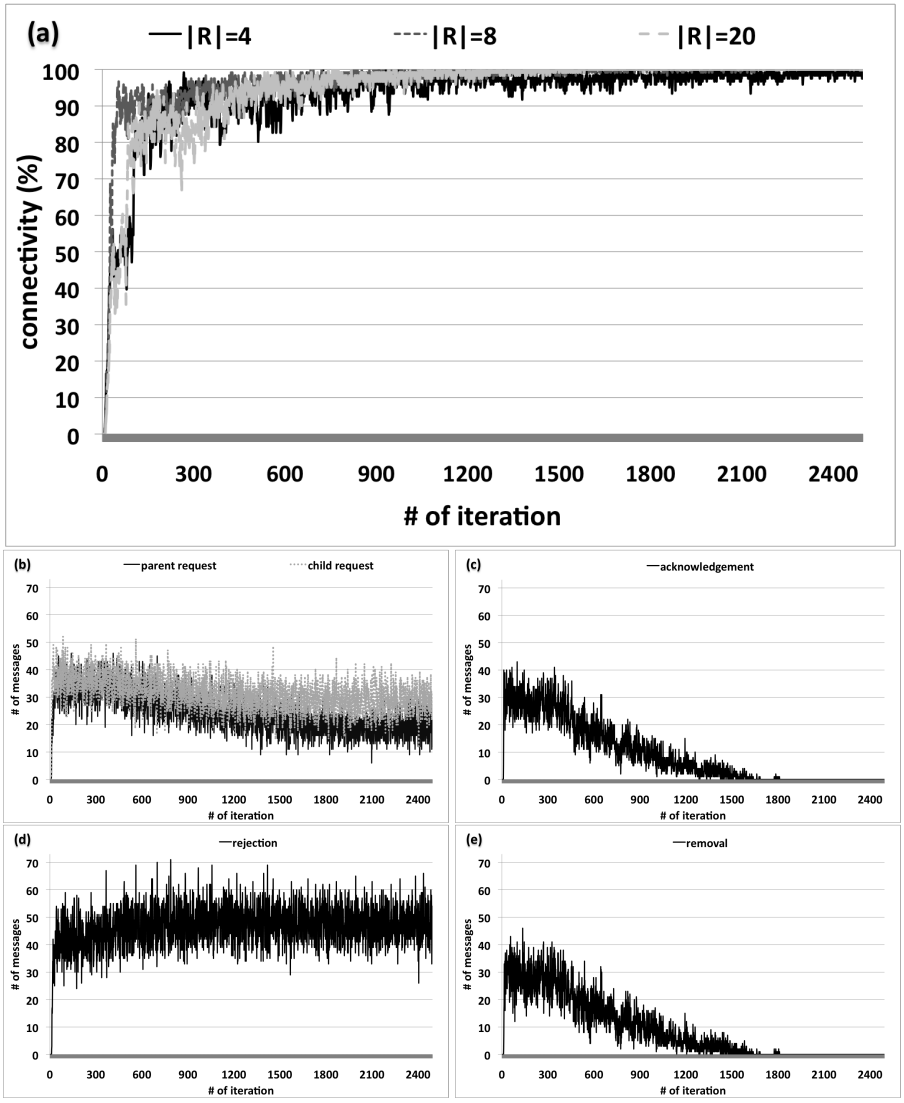

Fig. 7 Cost-effectiveness of AETOS in 'simulation environment 1' for $c = 3$. (a) Connectivity convergence for different length of random views. (b)-(e) Number of messages generated by the ATOM layer of the 'self-organisation agent' for $|\mathbf{R}| = 20$.

the 150th iteration in Figure 9 is explained by the connection of a large branch of nodes to the main body of the tree.

The communication cost of the ATOM layer is related to three things: (i) request frequency, (ii) convergence of the system and (iii) effectiveness in the termination of self-organisation. The parent and child requests decrease during convergence 40%-45% and 25%-35% respectively. This is caused by the effect of the reconfigurations

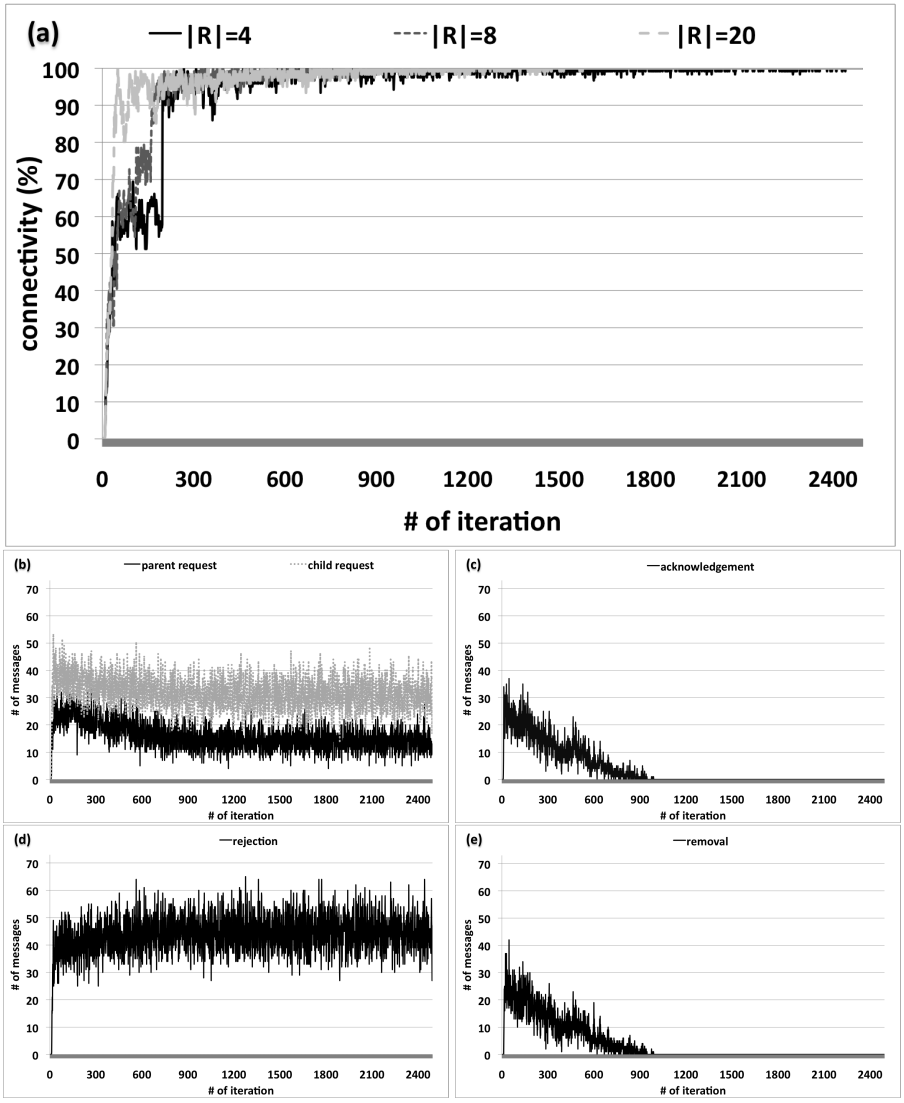

Fig. 8 Cost-effectiveness of AETOS in 'simulation environment 1' for $c = 5$. (a) Connectivity convergence for different lengths of random views. (b)-(e) Number of messages generated by the ATOM layer of the 'self-organisation agent' for $|\mathbf{R}| = 20$.

and the increase in the tree connectivity. In contrast, rejections increase 25%-30% as there are more nodes already connected that can potentially reject requests. After convergence, the number of messages is stabilised. At this point the system can be terminated and thus, alleviate the network from this constant communication overhead. Removal and acknowledgement messages decrease proportionally to the convergence time. This is expected, as nodes in a tree with 100% connectivity do

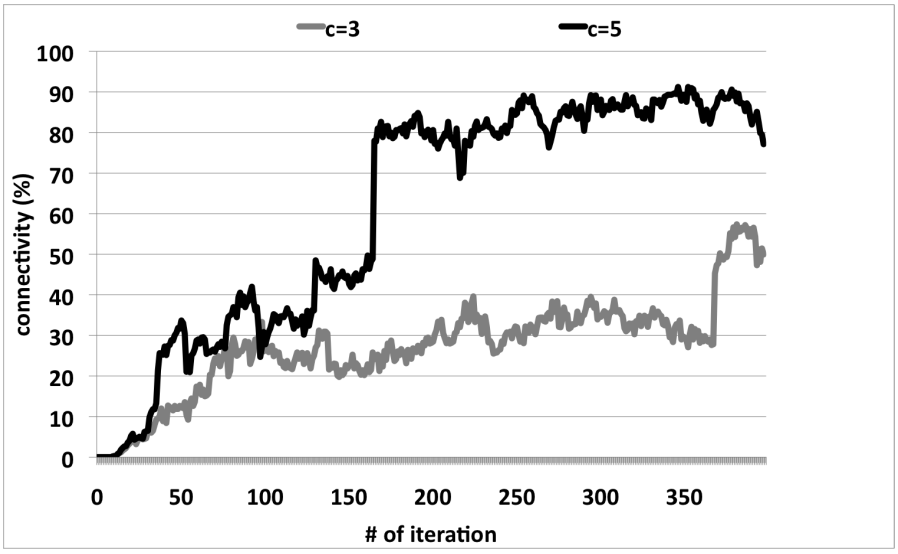

Fig. 9 Connectivity convergence in 'simulation environment 2' for $c = 3$ and $c = 5$.

not perform any removals or acknowledgements. Note that, the communication cost of the PAROS layer is constant and dependent on the network size and the gossiping period.

An increase in the length of random views makes connectivity convergence faster in both simulation environments. This can be explained by the better global knowledge that the 'self-organisation agents' have of the system. Therefore, they can improve the quality of the candidate neighbours select to which they potentially connect. Finally, the number of children c influences the cost-effectiveness of AETOS significantly. A different number of children results in different topologies. In each simulation environment setting of this section, trees have the same network size with a different number of levels. Connectivity increases from 81% ($c = 3$) to 97% ($c = 5$) in 'simulation environment 1' and from 50% ($c = 3$) to 77% ($c = 5$) in 'simulation environment 2' at the 400th iteration. Furthermore, communication cost also decreases 17% by increasing c in 'simulation environment 1'.

The future addition of 'proximity sampling' is expected to enhance the quality of the proximity view. More specifically, it is expected to (i) decrease the connectivity convergence time as the self-organisation agent will update the proximity view faster after the performed reconfigurations and (ii) decrease the communication cost of the ATOM layer as it is related to convergence time.

9 Conclusions and future work

This chapter proposes AETOS, the Adaptive Epidemic Tree overlay Service. AETOS is an agent-based system that builds and maintains application-independent tree overlays, on demand. To this purpose three local agents are defined to (i) abstract application requirements to self-organisation requirements, (ii) self-organise nodes in various optimised tree topologies based on these requirements and (iii) control the bootstrapping and termination of self-organisation. Experiments show that a high level of connectivity can be acquired, and that cost-effectiveness of self-organisation is highly correlated to the available local knowledge, the tree topology, and the network size.

These results are promising. Further research will include extension of the current system with a 'proximity sampling' component, study the effects of a distributed simulation environment, and application of AETOS in a more realistic domain.

Acknowledgements The authors are grateful to the NLnet Foundation and Delft University of Technology, for their support. http://www.nlnet.nl.

References

1. B. Akbari, H. R. Rabiee, and M. Ghanbari. DPOCS: A Dynamic Proxy Architecture for Video Streaming Based on Overlay Networks. In *IEEE MICC & ICON '05*, volume 1, page 6, November 2005.
2. G. An, D. Gui-guang, D. Qiong-hai, and L. Chuang. BulkTree: An overlay network architecture for live media streaming. *Journal of Zhejiang University*, 7(1):125–130, 2006.
3. S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. In *INFOCOM*, volume 2, pages 1521–1531, 2003.
4. S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. *IEEE/ACM Trans. Netw.*, 14(2):237–248, 2006.
5. R. Bhagwan, S. Savage, and G. M. Voelker. Understanding Availability. In *IPTPS*, pages 256–267, 2003.
6. S. Birrer and F. E. Bustamante. A Comparison of Resilient Overlay Multicast Approaches. *IEEE Journal on Selected Areas in Communications*, 25(9):1695–1705, 2007.
7. F. M. T. Brazier, J. O. Kephart, M. Huhns, and H. Van Dyke Parunak. Agents and service-oriented computing for autonomic computing: A research agenda. *IEEE Internet Computing*, 13(3):82–87, May 2009.
8. A. J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 35(3):373–384, 2005.
9. 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.
10. P. Costa and D. Frey. Publish-Subscribe Tree Maintenance over a DHT. In *ICDCSW '05: Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05)*, pages 414–420, Washington, DC, USA, 2005. IEEE Computer Society.
11. C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, 2000.

12. D. England, B. Veeravalli, and J. B. Weissman. A Robust Spanning Tree Topology for Data Collection and Dissemination in Distributed Environments. *IEEE Transactions on Parallel and Distributed Systems*, 18(5):608–620, 2007.
13. Z. Fei and M. Yang. A proactive tree recovery mechanism for resilient overlay multicast. *IEEE/ACM Trans. Netw.*, 15(1):173–186, 2007.
14. D. Frey and A. L. Murphy. Failure-Tolerant Overlay Trees for Large-Scale Dynamic Networks. In *P2P '08: Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*, pages 351–361, Washington, DC, USA, 2008. IEEE Computer Society.
15. W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer. ProtoPeer: a P2P toolkit bridging the gap between simulation and live deployement. In *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–9, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
16. A. González-Beltrán, P. Milligan, and P. Sage. Range queries over skip tree graphs. *Comput. Commun.*, 31(2):358–374, 2008.
17. H. V. Jagadish, B. C. Ooi, K.-L. Tan, Q. H. Vu, and R. Zhang. Speeding up search in peer-to-peer networks with a multi-way tree structure. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2006. ACM.
18. M. Jelasity, A. Montresor, and O. Babaoglu. T-man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321 – 2339, 2009. Gossiping in Distributed Systems.
19. C. Y. Lee and H. Dong Kim. Reliable overlay multicast trees for private Internet broadcasting with multiple sessions. *Comput. Oper. Res.*, 34(9):2849–2864, 2007.
20. 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.
21. M. Li, W.-c. Lee, and A. Sivasubramaniam. DPTree: A Balanced Tree Based Indexing Framework for Peer-to-Peer Systems. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 12–21, Washington, DC, USA, 2006. IEEE Computer Society.
22. Y. Li and W. T. Ooi. Distributed construction of resource-efficient overlay tree by approximating MST. In *ICME*, pages 1507–1510, 2004.
23. J. Liu and M. Zhou. Tree-assisted gossiping for overlay video distribution. *Multimedia Tools Appl.*, 29(3):211–232, 2006.
24. Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008.
25. R. P. Lopes and J. L. Oliveira. Software agents in network management. In *ICEIS*, pages 674–681, 1999.
26. Márk Jelasity and Spyros Voulgaris and Rachid Guerraoui and Anne-Marie Kermarrec and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.
27. P. Merz and S. Wolf. TreeOpt: Self-Organizing, Evolving P2P Overlay Topologies Based On Spanning Trees. In *SAKS'07*, Bern, Switzerland, 2007.
28. E. Pournaras, G. Exarchakos, and N. Antonopoulos. Load-driven neighbourhood reconfiguration of Gnutella overlay. *Computer Communications*, 31(13):3030–3039, 2008.
29. 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.
30. E. Pournaras, M. Warnier, and F. M. T. Brazier. Adaptive Agent-based Self-organization for Robust Hierarchical Topologies. In *ICAIS '09: Proceedings of the International Conference on Adaptive and Intelligent Systems*. IEEE, September 2009.
31. W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.
32. 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.

33. S.-W. Tan, G. Waters, and J. Crawford. MeshTree: Reliable Low Delay Degree-bounded Multicast Overlays. *Parallel and Distributed Systems, International Conference on*, 2:565–569, 2005.
34. C. Tang and C. Ward. GoCast: Gossip-Enhanced Overlay Multicast for Fast and Dependable Group Communication. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 140–149, Washington, DC, USA, 2005. IEEE Computer Society.
35. H. Tianfield and R. Unland. Towards self-organization in multi-agent systems and grid computing. *Multiagent Grid Syst.*, 1(2):89–95, 2005.
36. F. Wang, Y. Xiong, and J. Liu. mTreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In *in IEEE ICDCS*, page 49, 2007.
37. H. Zhuge and L. Feng. Distributed Suffix Tree Overlay for Peer-to-Peer Search. *IEEE Trans. on Knowl. and Data Eng.*, 20(2):276–285, 2008.