Department for Informatics
University of Fribourg (Switzerland)

Exploiting self-organization for the autonomic management of distributed systems

THESIS

presented to the Faculty of Science of the University of Fribourg (Switzerland)
in consideration for the award of the academic grade of
Doctor scientiarum informaticarum

by

Amos BROCCO

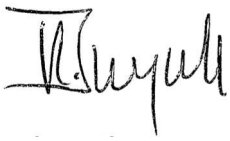
from
Sementina, Switzerland

Thesis N° 1687
UniPrint, Fribourg
2010

Accepted by the Faculty of Science of the University of Fribourg (Switzerland) upon the recommendation of:

- Prof. Heinz GRÖFLIN, University of Fribourg, Switzerland, President of the Committee;
- Prof. Béat HIRSBRUNNER, University of Fribourg, Switzerland, Thesis Director;
- Prof. Pascal FELBER, University of Neuchâtel, Switzerland, Examinator;
- Prof. Alcherio MARTINOLI, École Polytechnique Fédérale de Lausanne, Switzerland, Examinator.

Fribourg, October 1, 2010

The Thesis Director:	The Dean:
	
Prof. Béat HIRSBRUNNER	Prof. Rolf INGOLD

Exploiting self-organization for the autonomic management of distributed systems

Abstract: This thesis focuses on algorithms to support different aspects of distributed systems management and their implementation using self-organized, adaptive, and bio-inspired techniques. Three main topics are covered in the thesis: peer-to-peer overlay management, efficient resource discovery, and decentralized task allocation. Concerning peer-to-peer overlay management, a bio-inspired algorithm called BLÅTANT was developed. BLÅTANT is used to build and maintain an optimized peer-to-peer overlay through the collaborative behavior of different species of mobile ant agents. Connections between peers are modified to bound the diameter of the overlay and to remove redundant links that may result in excessive communication traffic. The initial idea has evolved into two fully distributed and fault resilient solutions: BLÅTANT-R and BLÅTANT-S. In order to support efficient resource discovery, a decentralized search protocol that improves probabilistic flooding by means of a proactive caching mechanism was introduced. The proposed solution is based on epidemic information sharing, and provides important improvements in the recall rate with minimal network overhead. Finally, to support decentralized task allocation, and provide intelligent scheduling decisions across multiple grid nodes, a collaborative community scheduling algorithm named ARiA, which aims at serving the grid as a whole has been implemented and evaluated. Based on the research performed in this thesis, it is our opinion that self-organization can bring a decisive improvement in the performance, reliability, and robustness of distributed systems.

Keywords: Distributed Systems, Peer-to-Peer Systems, Self-Organization, Overlay Management, Resource Discovery, Bio-inspired Computing, Grid Scheduling

Utilizzo di metodologie auto-organizzate per la gestione autonoma di sistemi distribuiti

Sommario: Questa tesi vuole assumere come oggetto d'analisi degli algoritmi per la gestione di diversi aspetti dei sistemi distribuiti, nonchè la loro implementazione tramite l'impiego di meccanismi auto-organizzati, adattivi e bio-ispirati. Tre sono i temi trattati nella tesi: la gestione di una rete *overlay* basata sulla tecnologia *peer-to-peer* ($P2P$), la ricerca d'informazione attraverso metodi completamente distribuiti, e l'allocazione decentralizzata di compiti su un insieme di sistemi distribuiti. Per quello che riguarda il primo tema, ovvero la gestione di un *overlay* P2P, viene descritto un algoritmo che mira all'ottimizzazione delle connessioni tra sistemi ispirato al comportamento delle colonie di formiche. Le connessioni vengono modificate dall'algoritmo riducendo sia il diametro della rete sia il numero di collegamenti ridondanti, al fine di limitare il traffico generato dalla comunicazione tra sistemi. L'idea iniziale si evolve in due implementazioni completamente distribuite, chiamate BlåtAnt-R e BlåtAnt-S. Successivamente, per permettere la ricerca d'informazione sulla rete, presentiamo un protocollo decentralizzato che migliora l'efficienza dei metodi di ricerca esistenti. La soluzione proposta è basata sullo scambio di informazioni tra i nodi della rete attraverso un protocollo epidemico. Infine, per offrire un'ottimale allocazione di compiti sulle risorse disponibili (per esempio in una griglia computazionale, o *grid*), discutiamo un algoritmo di schedulazione completamente distribuito chiamato ARiA. Le soluzioni proposte nella tesi sono valutate dettagliatamente, discutendone i pregi e i difetti. Sulla base della ricerca presentata in questa tesi, è nostra opinione che i metodi auto-organizzati possano apportare importanti benefici per ciò che concerne l'efficienza, la robustezza e l'affidabilità dei sistemi distribuiti.

Parole chiave: Sistemi distribuiti, P2P, Auto-Organizzazione, Ricerca d'informazione, Sistemi Bio-ispirati, Allocazione sul grid

Dedication

To Rossana
Thank you for loving me and always making me feel good about myself :)

Acknowledgments

I am heartily thankful to my supervisor, Béat Hirsbrunner, for giving me the opportunity to study for a PhD, supervising my work and providing useful guidelines. I would also like to thank all the members of my Ph.D. committee, and gratefully acknowledge the financial support of the Swiss Hasler Foundation.

My deepest gratitude and thanks go to my friends and colleagues at the Department of Informatics of the University of Fribourg, who have supported me in many different ways during these years. In no particular order, my gratitude to Apostolos Malatras, Fulvio Frapolli, Agnes Lisowska, Maurizio Rigamonti, Ye Huang, Denis Lalanne, Vincenzo Pallotta, Philippe Froidevaux, Oliver Schmid, Daniel Ostojic, Muriel Bowie, Pascal Bruegger, Isabelle Colin, Kevin Cristiano, Momouh Khadraoui, Fei Peng, Daniel Fasel, Dominique Guinard, and Nicolas Juillerat. I am also grateful to the secretaries of the Department of Informatics of the University of Fribourg, for helping the department to run smoothly and for assisting me in many different ways.

Sincere thanks are extended to all the anonymous reviewers of my published papers for their valuable suggestions which indeed helped improve my research.

I am grateful to all my friends from the best damn forum on the web, in particular to Federico “Raistlin”, Ilaria, and Marco “Motolo”.

I also wish to thank Hayley, Joshua, Jeremy, Zachary, Taylor, Matthew, Christopher, Dominic, Avril, Anna, Dido, and Bethany.

My special appreciation goes to Isabella, Franco, and Marco for their kindness and affection. I want to thank my family for supporting me in every step of the way, for always being there for me, for their love and in general for everything. Finally, I would like to express special thanks to Rossana. She helped me to concentrate on completing this thesis, and supported me with her love and encouragement.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of this research work.

Contents

Chapter 1

Introduction

Contents

Distributed systems describe a collection of independent computers that appear to the user as a single entity [276]. Distributed systems can be classified in different categories according to their structure and purpose: peer-to-peer systems [58], ad-hoc networks [222], mesh networks [18], grid systems [119], etc. Despite some differences, the common goal driving the development and deployment of distributed systems is the ability and willingness of the participating entities to share local resources with the community, such as files in peer-to-peer systems or computing resources in grids. The benefits of distributed solutions are functional separation, improved reliability, higher scalability, and reduced costs [276]. On the downside, distributed systems are inherently more complex to manage, difficult to secure and to fully exploit than their centralized counterparts. A number of techniques for managing the issues raised by distributed applications (such as coordination and synchronization) have already been proposed in the past; unfortunately, the scale, dynamism, volatility, and security concerns of current scenarios often require rethinking of new solutions [245]. Moreover, the endowments of distributed solutions are often hindered by the complexity required for the management of the underlying infrastructure. Consequently, solutions that simplify administration and lessen the burden of configuring and optimizing distributed systems are required.

Self-organization and adaptiveness can be considered as desirable features for improving in this area and establish reliable, efficient, and scalable distributed solutions. In particular, self-organizing systems have been put forward as a way to overcome the *complexity bottleneck* [107], by replacing complex centralized control with fully distributed operation emerging from the interaction and coordination between a multitude of simple components. In accordance with the definitions proposed in [107, 126], we understand the term *organization* as *structure with function*. Whereas structure concerns the arrangement (or order) of components in the system (for example, the topology of a network), function is related to its purpose. Under this definition, a *self-organizing system* is able to spontaneously create and maintain a functional structure without central control, and self-organization refers to the spontanous emergence of a global order [171] (as opposed to chaos and entropy) resulting from distributed control and local interactions [145]. The distributed nature of

self-organizing systems provides a number of advantages over their centralized analogues, such as resilience, robustness, graceful degradation and recovery from errors [145]. Relying on the interrelationships among a large number of different elements composing the system as well as positive or negative feedback mechanisms, self-organized solutions are intrinsically dynamic; nonetheless, such dynamicity is not arbitrary, but rather convergent toward preferable configurations that optimally fulfill the purpose of the system. In this regard, the adaptiveness of a system refers to its ability to modify its organization in order to optimally fit its purpose to the conditions of the environment [145].

Our research spans over different concerns, such as communication, information retrieval, and resource allocation. Accordingly, this thesis addresses major critical aspects of the design, implementation and deployment of distributed systems, and offers autonomic solutions for building an optimized peer-to-peer overlay, supporting efficient resource discovery, and promoting fully distributed task allocation. Following the separation of concerns proposed in [151], we divide our system into different functional components, namely overlay management, resource discovery, and task allocation. The peer-to-peer overlay represents the foundation of our work, as it provides an adaptive communication infrastructure on top of which high-level applications have been implemented. More specifically, a resource discovery mechanism that exploits the characteristics of the overlay and local shortcut caches to efficiently forward queries has been implemented. Additionally, a task allocation protocol has been designed to enable dynamic and optimal distribution of computing tasks across all available resources. In the rest of this chapter, we explore the motivation behind our research, discuss the considered evaluation scenario, and highlight the benefits introduced by our solution.

1.1 Motivation

In recent years, the availability of a large number of networked computers, high-bandwidth connections, as well as the general adoption of broadband access to the Internet, enabled the deployment of distributed systems achieving unprecedented scale and popularity. In this context, grids have emerged as infrastructures for high performance computing, that serve a number of scientific communities [72] and leverage a large pool of geographically dispersed resources to solve large and complex tasks. Conversely, peer-to-peer file-sharing networks [3, 8] have become an accessible mass-phenomenon used by millions of users worldwide. Despite the progress made in simplifying end-user interaction with such distributed systems, managing large scale deployments, enabling efficient access to their resources, and dealing with security aspects remain active domains of research [289, 221].

Based on the review of the most common unjustified assumptions of distributed computing presented in [245] we can identify several characteristics that should be accounted for in order to create robust and reliable systems: fault tolerance, asynchronous operation, efficient network usage, security management, and support for dynamic and heterogeneous networks. More specifically, distributed systems must be able to cope with failures of both hosts and the underlying network infrastructure: failures should be detected and an appropriate response should be triggered to ensure proper operation of the system. Distributed applications must also not depend on synchronous operations, as communication latency between different hosts may significantly differ [307], and avoid assuming that the system

is composed of homogeneous resources.

From a structural perspective, different architectures have been proposed. Depending on the scale, constraints, and goals, centralized, hierarchical or fully distributed management is employed. On one end, centralized approaches represent simple solutions to serve a large number of geographically sparse clients with minimal bandwidth consumption. Central systems are easier to secure, and data consistency can be easily ensured. Meanwhile, fully distributed designs reduce maintenance costs and increase robustness by avoiding dependency on central systems that represent potential single point of failures. Unfortunately, such a design introduces coordination challenges and increases traffic.

Peer-to-peer applications represent a widely known example of fully distributed systems, while grids traditionally rely on centralized control and service provisioning. These opposite approaches mainly reflect the differences between peer-to-peer systems and grids. Peer-to-peer systems are highly dynamic systems, with less engagement from each participant, while grids are relatively stable, persistent and reliable [275].

At the functional level, to solve the aforementioned management problems, research has turned to autonomic computing [172] as potential solution for automated and adaptive systems [211]. Autonomic systems promote self-configuration, self-repair, and autonomous optimization of the quality of service. In this regard, information about the environment can be coupled with specific management policies to enable autonomic operation of the system in a fully decentralized way [95, 177]. Moreover, bio-inspired solutions [48] represent a suitable approach to the problem, because they inherently support all the important self-$\star$ features of autonomic computing.

Bio-inspired computing replicates natural phenomenons, such as genetics [176] or emergent behaviors [98], to solve complex computational problems. In contrast to traditional approaches, bio-inspired solutions are generally geared toward decentralized problem solving, with techniques resulting from the collaboration of several entities governed by simple rules. In the context of network management, a number of bio-inspired methods have been proposed to tackle problems such as routing in complex topologies [63] and load balancing [210]. Emergence is of particular interest for distributed systems; in systems with emergent properties the behavior is not a property of an individual entity, but rather the result of collaborative interactions between all components. This advantage of emergent approaches also affects the robustness of the system: whereas centralized approaches might suffer a complete breakdown in case of failure, emergent solutions do not depend on single entities and thus represent reliable solutions capable of surviving unexpected situations and problems.

Accordingly, the aim of this thesis is to investigate the implementation of novel self-organized solutions to ease the deployment of distributed systems. In particular, we aim at employing bio-inspired techniques to provide unsupervised adaptation to changes in the environment. Furthermore, in order to address all issues related to centralization, we propose to base our system on fully distributed mechanisms, and employ peer-to-peer communication between the components of our solution. Security aspects are out of the scope of this thesis; a review and analysis of security in peer-to-peer system is available in [289, 30].

1.2 Evaluation Scenario

The work presented in this thesis has evolved in the context of a novel middleware for grids named SmartGRID [150] [1], the goal of which is to provide an abstraction layer for the deployment of robust and reliable grid services on top of a multitude of loosely connected, heterogeneous and volatile resources. The proposed solution thus aims at filling the gap between grid applications and the computing resources.

Whereas currently deployed grid systems are relatively stable and comprised of a limited number of nodes, the vision for next-generation grids foresees large-scale networks with a highly dynamic and evolving behavior, with nodes joining and leaving the system in real time and with the number of nodes increasing over time. SmartGRID envisions a grid computing environment that is open to a larger group of contributors than traditional grids, and that requires less management effort. To increase robustness and avoid single points of failure, SmartGRID promotes loosely coupled peer-to-peer interaction between the engaged entities. Each contributing site is independently managed according to local usage policies; moreover, SmartGRID is designed to integrate with, and make use of, existing platforms and infrastructures, thus creating a complementary, instead of an alternative, technology to increase efficiency. In this context, interoperativity with traditional resource management systems is an essential feature of the platform.

The proposed architecture is composed of two independent layers that communicate through an intermediate datawarehouse, as depicted in Figure 1.1.

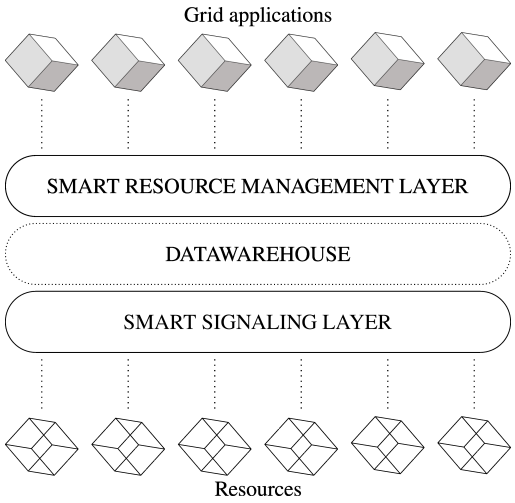


Figure 1.1: SmartGRID architecture

Smart Signaling Layer The Smart Signaling Layer (SSL) enables low-level communication between resources, and provides resource discovery services. The primary goal of the SSL is to abstract from the heterogeneous and volatile nature of the underlying resources and network infrastructure to provide a robust communication and service provisioning

[1] SmartGRID is supported by the Swiss Hasler Foundation, in the framework of the ManCom Initiative (ManCom for Managing Complexity of Information and Communication Systems), project Nr. 2122

framework. Operation and services offered by the SSL degrade gracefully in the event of network or site failure; for this, the system is based on a fully decentralized and distributed design that avoids bottlenecks and single points of failure. Furthermore, research on the SSL focused on the implementation of a self-organized and adaptive peer-to-peer solution by employing bio-inspired methods. In this context, of particular interest are ant-inspired approaches, as they have already proven to provide robust means for network-related problems such as routing [63] or load-balancing [210].

Smart Resource Management Layer The Smart Resource Management Layer (SRML) is in charge of supervisioning the usage of resources and mediating interaction between the user and the system by providing an interface for task submission and tracking. The SRML exploits information from the SSL to efficiently schedule tasks either on local resources or on remote nodes. Accordingly, the SRML interoperates with the existing scheduling infrastructure, and obeys to local and remote resource usage policies.

Datawarehouse The SSL and SRML communicate through a datawarehouse, which provides both an asynchronous communication channel and a temporary storage. In the context of the SmartGRID middleware, the datawarehouse also helps maintaining clear separation of concerns between the two functional layers.

Although the main contribution of this thesis falls within the Smart Signaling Layer, research has spanned over all layers. In particular, the overlay management algorithm introduced in Chapter 3 and the resource discovery protocol in Chapter 4 concern the SSL, while the meta-scheduling framework presented in Chapter 5 concerns the SRML.

1.3 Research Problem

Different questions arise from the previously described scenario.

- "Can we exploit self-organization and bio-inspired solutions to provide an optimized peer-to-peer communication and service provisioning framework?"
- "Can we improve existing resource discovery mechanisms using fully distributed bio-inspired solutions?"
- "Can we provide efficient task allocation to optimally exploit a large number of resources by means of a fully distributed scheduling mechanism?"

These questions summarize the research problem addressed by this thesis.

1.4 Contributions of this thesis

According to the requirements that arise from the research problem and the considered evaluation scenario, the contributions of this thesis are threefold. First, it proposes an algorithm for managing *a self-structured adaptive peer-to-peer overlay*, that is optimized

to support efficient communication between nodes while avoiding single points of failures and bottlenecks. Next, it introduces a generic method for *improving resource discovery effectiveness* by exploiting local caches. Finally, it addresses the problem of *efficiently allocating tasks across heterogeneous resources* by means of a lightweight meta-scheduling protocol to fully exploit the benefits of distributed computing without imposing limitations on local resource management. To achieve our goals, we introduce novel techniques and make use of self-organized and bio-inspired methods in order to ensure robust and adaptive behaviors.

In the context of the SmartGRID project, this thesis also introduces a software platform for the deployment of fully distributed bio-inspired solutions, in particular those involving ant-like mobile agents. The generic nature of the latter enables its integration into different projects that aim at implementing and exploiting distributed swarm intelligence solutions. Furthermore, along with the aforementioned algorithms and protocols, this platform represents a fully functional framework for supporting robust and adaptive grid services in a heterogeneous environment, as envisaged by the SmartGRID project.

1.5 Structure

This thesis covers the aforementioned topics as follows.

Chapter 1 provides an overview of the thesis, and discusses the motivations behind the research topic as well as the open issues and challenges to be addressed. Furthermore it defines the research problem and highlights the contributions of the thesis.

Chapter 2 studies the related work in the field of peer-to-peer systems. Our discussion covers the two main classes of existing peer-to-peer solutions, namely *structured* and *unstructured* ones, and draws some comparisons of the corresponding drawbacks and advantages with help from a detailed analysis of noteworthy projects. Some theoretical foundations about graph theory are also presented.

Chapter 3 consolidates the knowledge gained through this literature review in a list of requirements that constitute the guidelines for validating our solution for the management of a peer-to-peer overlay. Consequently, the fundamental block of our research, namely that of a self-organized optimized peer-to-peer overlay is presented. Our novel overlay management algorithm, called BlåtAnt, represents a fully distributed solution based on bio-inspired techniques. The logical foundations of our approach are firstly validated by an analytical construction and subsequently by empirical experiments based on two implementations that attest the qualities of the solution.

In Chapter 4 we deal with the problem of providing an efficient resource discovery mechanism by means of a fully decentralized proactive caching system. The proposed solution improves existing flooding protocols by exploiting local caches on each node, that are updated using an epidemic protocol, to direct queries toward nodes that are more likely to provide the required service. Extensive evaluation assesses the improvements in the recall rate and the increased efficiency provided by the proposed search scheme.

Chapter 5 concerns distributed task allocation, thus putting more focus on the considered grid computing scenario. In this view, a review of existing scheduling solutions is presented. Subsequently our fully distributed meta-scheduling protocol is introduced and evaluated.

Chapter 6 reviews the proposed solutions in the context of the SmartGRID project. In particular, a prototype platform for the development and deployment of some of the aforementioned bio-inspired algorithms is presented and relevant details are demonstrated.

Chapter 7 draws the conclusions of this thesis, and summarizes the results and achievements of the work presented in the preceding chapters. Moreover, food for thought and pointers for future research based on the current work are provided.

Chapter 2

Peer-to-Peer Systems

Contents

Peer-to-Peer systems refer to a distributed computing paradigm that is built upon network architectures where nodes act as both servers and consumers. In contrast, traditional *client-server* models exhibit a clear separation between nodes providing services and those making use of it.

A *network overlay* is a logical topology maintained on top of another network, either a physical one (built of wired or wireless communication links) or a virtual one (another overlay). Peer-to-Peer systems are typically managed by application-level protocols that depend on the underlying network facilities (such as TCP or UDP communication): each node of the overlay needs to be able to communicate with possibly all other nodes, provided that the end address is known.

Peer-to-peer overlays are characterized by direct communication between the participating members, aimed at improving scalability, fault-tolerance and robustness compared to centralized approaches. In contrast to physical networks, connections in an overlay are logical, and depend on the ability of each node to address communication towards arbitrarily any other node by exploiting the routing capabilities of the underlying network. Physical and overlay networks can be abstracted to graphs, the topology of which is defined by the (logical) connections between the nodes. In this respect, an advantage of overlay networks over physical ones is the ability to easily modify or adapt the topology to meet user-defined requirements.

Nodes connected to a particular overlay may be referred to as a *community*: nodes within the same community typically share some resources amongst each other, such as services or data. Building and maintaining such communities, namely providing mechanisms to allow nodes to join the overlay and to contact other nodes, is achieved by means of *membership management* protocols [285, 123]. These protocols also aim at providing an efficient communication channel to spread information, to implement anonymous and secure communication, or to overcome censorship barriers.

The focus of peer-to-peer communities is to enable collaboration amongst a large number of systems and ease the sharing of information between them. Accordingly, research in this field is concerned with both the problem of maintaining an overlay as well as that of retrieving information. This chapter focuses on both issues, as a review of existing solutions cannot neglect the existence of a close relation between overlay management and resource discovery. More specifically, while all peer-to-peer systems are characterized by their lack of central authority, different peer-to-peer architectures follow diverse design principles depending on the strategies developed to retrieve information. An important step in the understanding of these systems is identifying the major differences between approaches, and highlighting their benefits and drawbacks. We survey existing research work aiming at tracing the principles of peer-to-peer systems and presenting some noteworthy solutions, restricting our discussion to information storage and retrieval peer-to-peer system on a fixed network infrastructure. We thus omit topics such as overlays for streaming (P2PTV) [143, 190], voice communication (VOIP) [11, 264], or mobile and ad-hoc communication [228, 89]. The purpose of this review is to briefly highlight key design concepts of existing systems. A thorough analysis and taxonomy of the extensive literature available on this topic is outside the scope of this thesis. Nonetheless, an in-depth analysis of the current

state-of-the art of peer-to-peer approaches is available in [58].

The foundations for understanding the characteristics of an overlay network lie in their mathematical properties. In this regard, an introduction to graph theory concepts that will be used throughout the rest of this thesis are presented in Section 2.1. Our discussion will then focus on the structural differences between existing solutions, with Section 2.2 introducing peer-to-peer information systems and highlighting the main differences between current approaches, namely structured and unstructured designs. An in-depth look of structured systems is provided in Section 2.3, with a discussion of noteworthy implementations and a detailed review of the mechanisms for storing and retrieving information in the overlay. Conversely, Section 2.4 presents unstructured solutions, while Section 2.5 analyzes the problem of churn in peer-to-peer overlays. Section 2.6 elaborates on the application of peer-to-peer technologies in the field of grid resource discovery. Finally, this chapter concludes in Section 2.7, with a discussion on findings that enables a better understanding of the goals and issues that are addressed by our solution.

2.1 Graph theory fundamentals

The characteristics of a computer network can be analyzed by considering it as a directed graph. In this respect, to better understand the terms used throughout the rest of this thesis, a brief but precise definition is required. Further analysis on the topic can be found in [294].

A *graph* is defined by a triple $\langle V, E, \rightarrow \rangle$, where V is a set of *vertices*, E a set of *edges*, and $\rightarrow$ a relation associating two vertices (called *endpoints*) and an edge. A graph can be used to represent computer *networks* consisting of *nodes* and un-directed *links*: accordingly nodes are the vertices, and links are the edges of the graph. Similarly, network overlays can be mapped to graphs where edges are defined by logical connections between nodes. For simplicity, in the rest we overlook the differences between the terms *graph*, *network*, *topology*, and *overlay*, and use either one of them interchangeably. In the same spirit, the terms *node* and *vertex*, as well as *edge* and *link* will also be used interchangeably.

For a graph $G = \langle V, E \rangle$, we define $n \in G \Leftrightarrow n \in V$. A node n_i in a graph $G = \langle V, E \rangle$ is *adjacent* to another node n_j if there exist an edge $(n_i, n_j) \in E$. The *neighborhood* set N_i of node n_i is the set of nodes adjacent to n_i, the size of of which determines the *degree* of a node. A graph is said to be *undirected* iff : $\forall n_i \in V : \forall n_j \in N_i \rightarrow n_i \in N_j$, making the adjacency property commutative; otherwise the graph is said to be *directed*. In a computer network we consider a node n_i as *connected* to another node n_j iff n_i is adjacent to n_j, i.e. $n_i \in N_j \ \wedge \ n_j \in N_i$. Figure 2.1 illustrates an example graph where $V = \{a, b, c, x, y\}$, $E = \{(b, a), (c, b), (b, x), (x, b), (c, y), (y, c)\}$. Arrow lines depict edges connecting two endpoint vertices, whereas double arrow lines indicate undirected links. A *path* in G is a succession of nodes $n_k \in G$, such that there exists a link between every node in this succession.

In a physical network, each link is a physical connection (either wired or wireless) between two nodes. In an overlay network, a (logical) link between two nodes exists if both have knowledge of each other and an active communication takes place. A graph G is *connected* if for each pair of nodes $n_i, n_j \in G$, there exists at least a path between

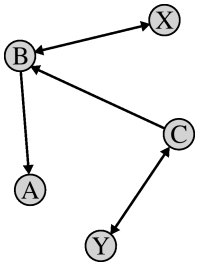


Figure 2.1: Example graph

them. Furthermore, a graph is said *fully connected* if there exists a link between each pair of nodes. Conversely, if a graph is not connected, it is said to be *partitioned*.

Several measurements are useful to describe high-level properties of a graph. In this regard the *degree* of a node is its number of neighbors; if the graph is directed it is possible to distinguish between *in-degree* and *out-degree*, for links originating from a node, respectively ending on a node. Related to the degree, important measures that help defining the complexity of a graph are the *average degree*, which represents the mean degree across all nodes, and the *degree distribution*, which expresses the probability that a node has exactly a given number of neighbors.

The *eccentricity* of a node n in a connected graph G is the the greatest distance between n_i and any other node $n_j \in G$. Accordingly, the *diameter* of a graph G is the maximum eccentricity of any node $n_i \in G$, the *radius* is minimum eccentricity of all nodes $n_i \in G$, while the *average path length* represents the average eccentricity.

The *clustering coefficient* of a graph measures the degree to which nodes share common neighbors. For a node n_i with neighbors degree k, the local clustering coefficient C_i is computed as the quotient of the number n of existing links between n_i's neighbors and the number of all possible links between them ($\frac{k(k-1)}{2}$):

$$C_i = \frac{2n}{k(k-1)}$$

The clustering coefficient C of a graph G is the average of the local clustering coefficients of all vertices:

$$C = \frac{1}{|G|} \sum_{n_i \in G} C_{n_i}.$$

In a social network, the clustering coefficient is the probability that two friends of a person are also mutual friends. Finally, the *girth* of a graph is the length of the shortest cycle. If the graph does not contain any cycles (for example, a tree graph), its girth is infinite.

2.2 Peer-to-peer information systems

The main challenge of peer-to-peer system is efficient information retrieval mechanisms that provide satisfactory results while being scalable and consuming a reasonable amount of bandwidth. We measure this level of satisfaction by computing the *hit rate* (also known

as *recall rate*), namely the ratio between retrieved results out of all possible ones. Because retrieval queries typically generate less traffic than actual content transfer, some peer-to-peer systems, such as Napster [9] and BitTorrent [1], implement dedicated *centralized indices* and rely on peer-to-peer interaction only for data exchange. Centralized indexing schemes are simpler to design and provide efficient (network traffic-wise) search, but create single points of failure, as well as robustness and scalability issues. On the other hand, *pure peer-to-peer solutions* remove the bottleneck of central servers, and make use of fully distributed search mechanisms across equipotent nodes at the expense of longer response times. Moreover, fully decentralized search in pure peer-to-peer overlays involves an additional trade-off between the quality of results and the generated traffic. As a consequence, *hybrid* [300] (hierarchical) solutions have been developed: some of the peers, typically the ones with greater computing capabilities or better connectivity, are used to mediate requests of other peers and cache information for later usage. Hybrid solutions recognize and exploit the heterogeneity of many peer-to-peer networks, with great variations in the capacity of each peer (both in terms of computational resources and connectivity).

2.2.1 Classes of Peer-to-Peer Systems

Because of the drawbacks of centralized indexing architectures, research on peer-to-peer systems mainly focuses on pure and hybrid solutions. In this context, the challenges raised by the dynamic and distributed nature of peer-to-peer systems has led to the development of different solutions for both the membership management problem, and the data management one. Two main classes are generally recognized [193, 58]: *structured* and *unstructured* overlay networks. In the former there exists a tight relation between the information shared on the overlay and the topology (structure) of the overlay itself, while in the latter freedom is given in both the construction of the overlay as well as in the storage of the data.

Structured solutions can be compared to a well maintained library, where books are classified by topic and alphabetically sorted. The location of a book can thus be precisely determined given that its title is known. While such precise organization enables very efficient search *by title*, it still fails to support more complex queries, such as *"All books with a butterfly on the cover"*. Moreover, an effort is required to keep order within the library: when a new topic is added or a shelf is full, books may be moved from one shelf to another. Conversely, unstructured solutions can be compared to a room with a lot of books laid on the floor: while searching for a book by title becomes more challenging, no particular care is required when adding or removing a book.

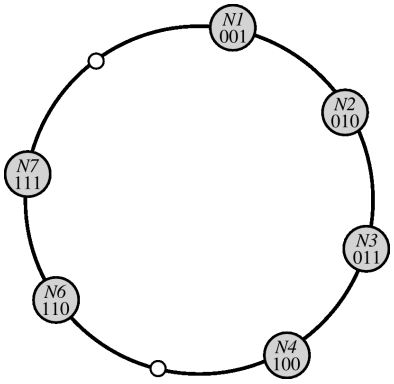
Both approaches inherit the benefits of distributed systems such as fault resilience and a lack of centralized control, however they bear important differences that need to be considered and discussed. Accordingly, the remainder of this chapter aims at reviewing the main characteristics of a number of structured and unstructured systems, highlighting their advantages and weaknesses. Our goal is to identify the requirements, the open challenges, as well as possible solutions that will drive the implementation of a novel peer-to-peer overlay to support resource discovery in our grid validation scenario.

2.3 Structured Solutions

Structured solutions, also known as Distributed Hash Tables (DHT), maintain topologies using deterministic algorithms in order to enable network efficient resource discovery and bounded delay performance. Contents shared by nodes are associated with an identifier, and all identifiers are then associated to nodes according to specific hash functions. There is thus a strong correlation between the content and the node that will store it. In order to locate content on the overlay, a lookup function is used to resolve the routing path to the node associated to the content's hash. Accordingly, the query can be successfully routed through one or multiple steps to the node storing the content. In this section several examples of DHT systems are presented and discussed.

2.3.1 Chord

Chord [269] assigns to each node an identifier of m bits within a circular space of size 2^m, so that the network is organized as a logical ring. Nodes know their successor in the ring, i.e. the node whose identifier follows in the identifiers' space. Content shared by nodes is also assigned an identifier (or key), which is typically a hash of the content's data, modulo m bits, generated using a consistent hashing function [169]. Consistent hashing functions ensure that adding or removing buckets in the table does not significantly concern the remaining ones; in this context, their use minimizes the number of nodes and keys affected by the addition or removal of a node (i.e. a bucket in the DHT), and helps spreading keys evenly across available nodes. A key K is published on the node referred to as $successor(K)$ whose identifier matches K or follows it.



K	successor(K)
0 (000)	N1 (001)
1 (001)	N1 (001)
2 (010)	N2 (010)
3 (011)	N3 (011)
4 (100)	N4 (100)
5 (101)	N6 (110)
6 (110)	N6 (110)
7 (111)	N7 (111)

Figure 2.2: Example Chord ring (m=3)

Figure 2.2 illustrates an example of a ring topology with $m = 3$: not all of the $2^3 = 8$ available identifiers are allocated to nodes. For all possible keys that can be mapped on the ring, the values for $successor(K)$ (i.e. the node to which a key K is assigned) are indicated. Keys 0 (000) and 5 (101) are assigned to nodes $N1$ (001), respectively $N6$ (110) because a node, the key of which exactly matches them, does not exist in the overlay.

To speed up the lookup operation, each node n maintains a routing table (called *finger table*) of size m that contains the identifiers of other peers in the ring: the entry at the i^{th} position ($1 \leq i \leq m$) in the finger table corresponds to the first node that succeeds n

by at least 2^{i-1} hops in the ring, i.e. $successor(s)$ with $s = n + 2^{i-1}$. Table 2.1 lists the contents of the finger table corresponding to node $N3$.

i	s	$successor(s)$
1	$3 + 2^0 = 4$	N4
2	$3 + 2^1 = 5$	N6
3	$3 + 2^2 = 7$	N7

Table 2.1: Example CHORD finger table for N3

Lookup procedure In order to find $successor(K)$ in the overlay, either to build up the finger table or to lookup for a key, a node starts by querying known nodes starting from the one that appears closer to K, and repeats the process until the target peer has been found. The routing cost in a CHORD overlay of N peers is of $O(logN)$ hops.

Joining and leaving A node joins the overlay by contacting one of the existing peers and finding its position in the ring by querying for the key associated with its identifier. When a node joins the overlay or leaves the system, the successors pointer and finger tables of nearby nodes in the ring need to be updated. CHORD solves this by periodically executing a *stabilization* procedure on each node to rearrange keys and update the finger table. To provide resilience in the event of successor's crash, each node maintains a list of nodes that succeed it in the ring: if a successor fails to respond to a query, one of the known backup successors is contacted. The cost of node join or leave is $O(log^2N)$ messages.

Further research A number of applications use CHORD as the underlying peer-to-peer overlay. Notable examples are the COOPERATIVE FILE SYSTEM (CFS) [88] which employs a CHORD overlay to locate data blocks on servers, project SPOVNET [43] which aims at creating a communication infrastructure over heterogeneous technologies and uses CHORD to implement its routing scheme, and a CHORD based DNS service [82]. Additionally, a self organized approach, named SELF-CHORD [113], proposes the use of bio-inspired mobile agents on a CHORD overlay to self-organize keys by clustering them on nodes. The overlay is constructed and maintained as in CHORD, but content's keys and node's keys are independent, as there is no need to assign a key to a precisely specified peer. Data is instead grouped into different classes, with each element in the same class sharing the same key value. Mobile agents reorganize the keys in the overlay using a clustering approach similar to [114]. Each node computes an average value called *centroid* based on the numerical value of the stored keys: agents move keys on the overlay in order to minimize the distance of each key to the centroid of the current node. Routing of queries is based on an estimation of the key distribution over the overlay that allows for jumping to nodes that are close to the target.

2.3.2 KOORDE

KOORDE [166] builds on the principles of CHORD by using a ring topology augmented with *de Bruijn* graphs [57] links instead of a finger table. De Bruijn graphs are composed

of 2^b nodes, for a given number of bits b, where each node is assigned one of the available numerical values in $[0, 2^b[$. Each node m is connected with nodes $2m \ mod \ 2^b$ and $(2m + 1) \ mod \ 2^b$.

Lookup procedure To route a message from x to y in a de Brujin graph each hop is resolved by progressively replacing x's low-order bits with y's high-order ones, i.e. by shifting x to the left and introducing y's high-order bits on the right. For example, in the de Bruijn graph shown in Figure 2.3, to route a message from node 110 to 001, the sequence of traversed nodes is $110 \rightarrow 10\underline{0} \rightarrow 0\underline{00} \rightarrow \underline{001}$.

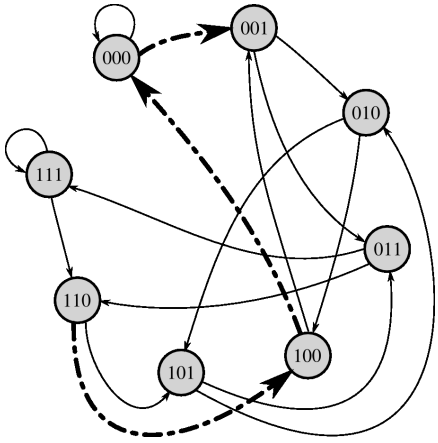


Figure 2.3: Example de Bruijn graph for $b = 3$. The highlighted links illustrate the routing path from node 110 to 001.

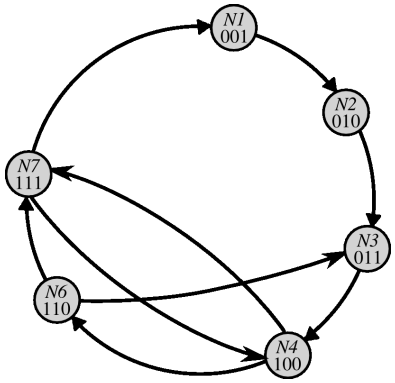
Because in a real network not all available identifiers 2^b are used (typically $b = 160$), KOORDE uses an adaptation of a de Bruijn graph to overcome the problem of missing (*imaginary*) nodes: each node m is connected with both the first node succeeding it on the ring, and the first existing predecessor of $2m \ mod \ 2^b$. Figure 2.4 shows a simple KOORDE topology with $b = 3$, detailing the shortcuts employed by each node. To provide fault tolerance in the event of a node failure, not only the first predecessor of $2m \ mod \ 2^b$ is known, but also the $O(logN)$ predecessors of it.

To route a message in the overlay, the previously described routing algorithm is adapted to support imaginary nodes by computing hops through them. Accordingly, in a KOORDE network with N nodes with a node degree of $O(log(N))$, it is possible to achieve $O(log(N)/loglog(N))$ hops routing.

Joining and leaving Because of the similarity between KOORDE and CHORD, the former uses the same procedures for joining the overlay, maintaining proper connectivity and recovering after abrupt disconnections by means of a stabilization process.

2.3.3 PASTRY

PASTRY [246] is a distributed peer-to-peer overlay infrastructure that bears similarities to Plaxton meshes [223] and makes use of prefix routing [28]. Nodes are assigned random



node	$s = 2m \ mod \ 2^b$	$predecessor(s)$
000	000	111
001	010	001
010	100	011
011	110	100
100	*000*	111
101	010	010
110	100	011
111	110	100

Figure 2.4: Example KOORDE overlay for $b = 3$. Values in *italic* in the table represent *imaginary* de Bruijn neighbors that do not exist in the overlay.

unique identifiers of length k bits ($k = 128$) that map uniformly into a circular space of size 2^k; nodes can simultaneously act as servers (storing objects), routers (addressing incoming messages to the next hop in the overlay), or clients (initiating lookups). Content shared in the network is assigned a key in the same namespace as node identifiers, meaning that a lookup operation is equivalent to routing a message towards a node.

Lookup procedure For the routing process, node and content identifiers are interpreted as digits with base 2^b, where b is a user-defined parameter typically set to 4. This value determines both the amount of information stored by each node, as well as the performance of the routing process: for an overlay of N peers, the routing algorithm is typically able to deliver a message to a destination in less than $O(log_{2^b} N)$ steps. The routing process itself makes use of the node's identifier or the content's key to forward incoming messages to the node the identifier of which is numerically closer to the target. More specifically, at each routing step, the message is forwarded to a known node whose identifier shares with the target a prefix that is at least one digit longer than the prefix that the target shares with the current node's identifier. If no such node is known, the message is forwarded to a node whose identifier shares a prefix with the key as long as the current node, but is numerically closer to the key than the current node's identifier.

To support routing, nodes have to maintain several data structures that actively compose their *node state*: a *routing table*, a *neighborhood set*, and a *leaf set*. The routing table contains $log_{2^b} N$ rows, with $2^b - 1$ entries each; for every node X, each entry in row n of the local routing table is a pointer (i.e. IP address) to a node whose identifier shares the first n digits (prefix of length n) with the identifier of X, but differs at least in the $(n+1)^{th}$ digit. The neighborhood set contains the addresses and identifiers of nodes that are in the proximity of X, and it is used to ensure that a message is forwarded to nodes with minimal distances. The leaf set is divided into two subsets, in order to store references to nodes the identifiers of which are either numerically larger, or smaller than that of the current node. An example of the state of a node is illustrated in Figure 2.5.

When a message is received, a node checks whether the identifier is within the bounds of the leaf set. If this is the case, the message is forwarded to the node in the leaf set the identifier of which has the minimal distance to the key in the message. Otherwise, the

Routing table

row	*entries*			
0	**0**212	⋆	**2**233	**3**322
1	1**0**12	1**1**31	⋆	1**3**33
2	12**0**3	12**1**1	12**2**3	⋆
3	123**0**	123**1**	⋆	123**3**

Leaf set

smaller	*larger*
1211	1321
1132	1312
1111	1333
1200	1321

Neighborhood Set

1231
1122
1322
1002

Figure 2.5: Example PASTRY node state for node 1232 ($b = 2$, and key size reduced to 8 bits for simplicity). Underlined text in the routing table highlights the prefix shared with the 1232 identifier, respectively **bold** the non-matching digits.

routing table is consulted and the message is forwarded to a node that shares a common prefix with the message's key by at least one more digit.

Joining and leaving A node Y can join the overlay by sending a special *join* message to a node within the overlay; the key of this message is the randomly generated identifier of Y. The join request is routed as a lookup request, and each traversed node sends its state to Y so that the latter can fill up its own data structures. More specifically, to fill the routing table, node Y will copy row n with the entries at row n from the node traversed at step n.

PASTRY nodes keep track of failed peers by means of heartbeat monitors and by detecting failures when forwarding messages. When a failed node is discovered, a recovery procedure is initiated by its neighbors in order to restore their state.

Further research Pastry is used to manage the overlay in SCRIBE [64], a decentralized multicast infrastructure: links between nodes are used to create multicast trees that enable efficient dissemination of messages. Furthermore, SPLITSTREAM [67] builds on SCRIBE to provide efficient high-bandwidth content distribution. Another project, PAST [100], implements a distributed storage solution with support for replication and load-balancing. Finally, SQUIRREL [155] implements a distributed web cache shared amongst a large number of machines.

2.3.4 TAPESTRY

TAPESTRY [309] (now called CHIMERA) uses a similar approach to PASTRY, but also deals with replication by means of multiple *roots* for each object. TAPESTRY uses a variation of Plaxton meshes where each peer is assigned a 160 bit identifier represented by a k digit key with base b. Nodes maintain a routing table that is used to forward messages by means of a prefix routing algorithm. The routing table is organized into rows with multiple levels: entries at the i^{th} row, j^{th} level, point to the closest nodes that share with X a common prefix of exactly $j-1$ digits, and whose j^{th} digit is equal to X's j^{th} digit *plus* 1. Figure 2.6 illustrates the routing table of an example TAPESTRY node with a digit identifier equal to 1232 ($b = 2$). To increase the resilience of the network, multiple references are kept for the same entry in the table. Nodes can publish new data in the DHT by determining the node that the content should be assigned to, and which will be referred to as the *root* of

the object. To achieve this, a lookup query with the identifier of the content is started. To improve both resilience and the lookup performance, each node along the routing path also stores a reference to the node where the request originated. Nodes that have shared contents in the overlay periodically renew their submissions by repeating the publication process.

Routing table

row ↓ / level →	1	2	3	4
1	**2**⋆⋆⋆	1**3**⋆⋆	12**0**⋆	123**3**
2	**3**⋆⋆⋆	1**0**⋆⋆	12**1**⋆	123**0**
3	**0**⋆⋆⋆	1**1**⋆⋆	12**2**⋆	123**1**

Figure 2.6: Example TAPESTRY routing table for node 1232 ($b = 2$, and key size reduced to 8 bits for simplicity). Underlined text in the routing table highlights the prefix shared with the 1232 identifier, respectively **bold** numbers the non-matching digits. Entries contain addresses of multiple nodes matching the given pattern: for example, entry 1**1**⋆⋆ may contain the pointers to 1123, 1102, etc.

Lookup procedure TAPESTRY lookup procedure uses a longest prefix matching routing algorithm. At each step nodes look in the table for the closest known node for the requested identifier: the message is progressively forwarded toward the node that is responsible for the content's key. Thanks to replication along the routing path, requests are most likely fulfilled before reaching the object's root, with the upper bound for number of hops being equal to $O(log(N))$.

Joining and leaving Nodes join at a position determined by a lookup of their own identifier in the network. The incoming node interacts with nodes on the routing path to retrieve information used to fill up the neighbors map. To finish the join process, nodes update their shared keys with adjacent peers. Finally, heartbeat messages are used to detect abrupt disconnections and ensure reliable operation of the overlay.

Further research OCEANSTORE [180] is a storage solution originally built on TAPESTRY (now based on BAMBOO [237]) that provides secure archiving on an overlay of untrusted servers. Another notable project that exploits a TAPESTRY overlay is BAYEUX [312], which implements a multicast infrastructure.

2.3.5 VICEROY

VICEROY [197] uses connected rings and an approximation of a butterfly network, while the mapping between content's keys and nodes resembles the principles of CHORD. An overlay of N peers is divided into $log(N)$ levels, with each level organized as a ring; all nodes are also connected in a global ring. Each node is assigned an *identity* that maps its discrete identifier to a real identifier in the $[0, 1]$ interval, and is randomly assigned to a level l. Beside from connections with its predecessor and successor in the corresponding ring, each peer is provided with additional connections to other nodes. In particular, five

long range contacts are created with peers located in different levels. A node with identity n in level l has connections with two peers at level $l+1$ (*down links* to a node in level $l+1$ at a distance of $1/2^l$, i.e. a node at level $l+1$ with identity at least $n+1/2^l$), one at short distance, one at long distance, and one connection with a peer at level $l-1$ (*up link* to a close-by node).

Lookup procedure To locate an item in the overlay, nodes forward the request following their up link until the identifier of the contacted node is lower than the item's key. Then, either ring or down links are used to continue routing up until the item has been found. As an example, consider the simple VICEROY overlay depicted in Figure 2.7. To route a message from node 7 to node 10, the forwarding path is: $7 \rightarrow 6$ (up link) $\rightarrow 5$ (up link) $\rightarrow$ 10 (down right link). The routing process in an overlay of N nodes requires $O(log(N))$ hops.

Joining and leaving Nodes have to select a level to connect to, and thus need an estimate of the size of the overlay. Instead of implementing a costly network size estimation algorithm, a node s estimates the size of the network as $N' = 1/distance(s, successor(s))$ (where $successor(s)$ is the successor in global ring). Node s selects its level uniformly at random in the interval $[1, N']$, and then contacts its successor in the ring of that level to complete the join process. When a node leaves, the remaining nodes reorganize their connections accordingly.

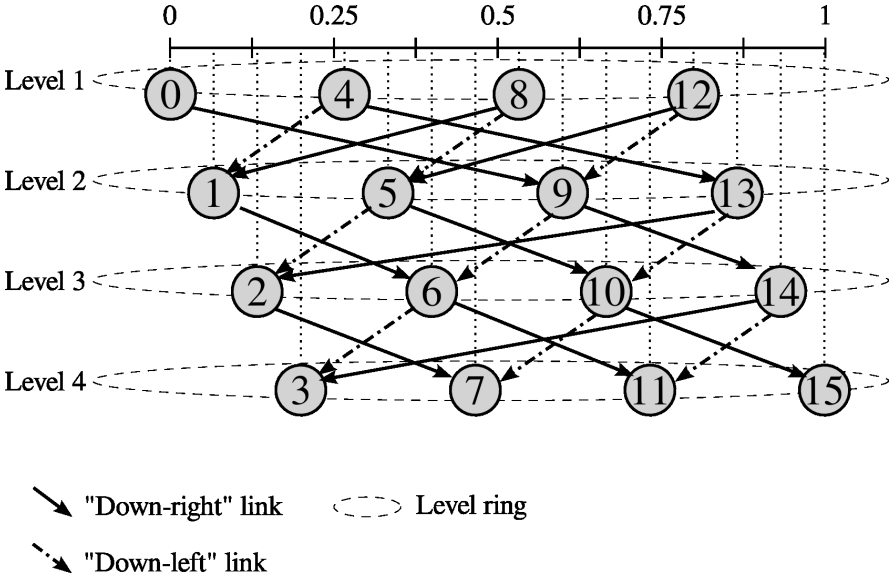


Figure 2.7: Example VICEROY network with 16 nodes. Dotted lines indicate the mapping between discrete identifiers and the real ones (identity) in the interval $[0, 1]$. Up links are omitted for simplicity.

2.3.6 CAN

The Content Addressable Network (CAN) [234] assigns to each node (and content's key) a portion of a d-dimensional toroidal key space (Figure 2.8). Keys for both nodes and contents are generated by means of a uniform hash functions that maps to a point in the space. Each peer stores the keys lying within its region; moreover, for every dimension, nodes are aware of close-by peers managing neighbor regions.

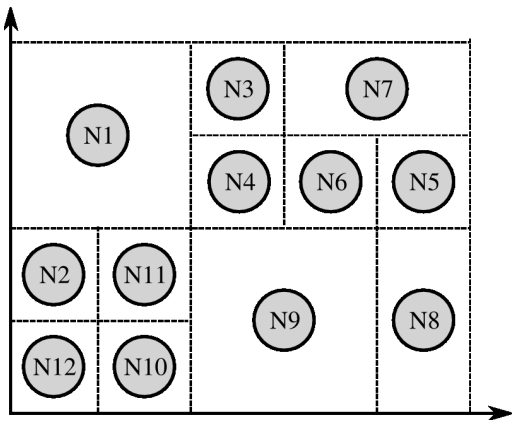


Figure 2.8: Example CAN 2-dimensional space with 12 nodes

Lookup procedure To lookup for a key, the hash function is applied to determine the associated point in the key space. Lookup messages are progressively forwarded, using a greedy routing algorithm, to nodes that are closer to the zone containing the point. The average path length in an overlay of size N with a uniform distribution of the keys in d dimensions is $O((d/4)/(N^{\frac{1}{d}}))$.

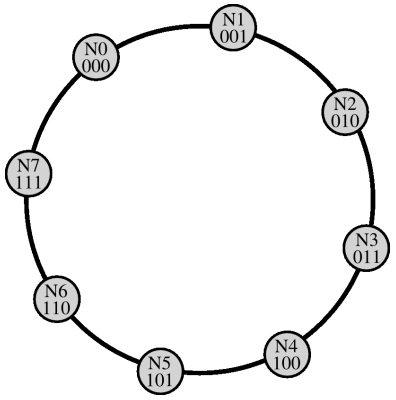
Joining and leaving To join the overlay, a node selects one of the existing peers and sends its request. The zone managed by the latter is split between the peers, and key-value pairs lying in the joining node's zone are transferred. Information about neighbor peers is fetched by the incoming node and all involved nodes in adjacent zones are contacted to update their neighbors' sets. A node that leaves the overlay will hand over the keys to a neighbor. To detect abrupt disconnections, nodes periodically exchange heartbeats with their neighborhood: if a failure is detected, close-by peers coordinate to assign the identifiers left over by the leaving node to the remaining peer that is currently responsible for the adjacent zone of smallest size.

Further research The work presented in [299], extends CAN overlays with additional shortcut paths, or *expressways*, that enable logarithmic routing and reduce latency. In a similar way, [274] augments a CAN overlay with long links to create a small-world network.

2.3.7 Kademlia

Kademlia [203] assigns to each peer and to each resource 160 bits key identifiers. Content keys are stored on nodes the identifier of which is close to the key by using a bitwise XOR metric. Each node maintains a list of $log(N)$ buckets, each of which contains k entries that refer to other nodes in the overlay. Entries in the i^{th} bucket refer to peers at a distance between $[2^i, 2^{i+1}[$. When for some key-value pair a node that is closer is detected, the pair is replicated instead of moved to improve fault tolerance; for the same reason, nodes periodically re-insert references to shared objects in the overlay.

Lookup procedure To route a message, peers compute the XOR distance $\oplus$ between their identifier and the destination, and use it to retrieve information from the buckets table: entries in the corresponding bucket are used to forward the request. In contrast to other DHT approaches, during a lookup Kademlia peers start parallel requests to other peers; moreover, peers exchange routing information during each lookup. This behavior minimizes the need for a separate exchange of information between peers.



Bucket table for 011 (N3)

row	range	entries
0	$[2^0 \dots 2^1[$	010 (N2)
1	$[2^1 \dots 2^2[$	001 (N1)
2	$[2^2 \dots 2^3[$	100 (N4), 101 (N5)

Bucket table for 101 (N5)

row	range	entries
0	$[2^0 \dots 2^1[$	001 (N4)
1	$[2^1 \dots 2^2[$	110 (N6), 111 (N7)
2	$[2^2 \dots 2^3[$	010 (N2), 011 (N3)

Figure 2.9: Example Kademlia overlay with 8 nodes and sample bucket tables for nodes 011 (N3) and 101 (N5).

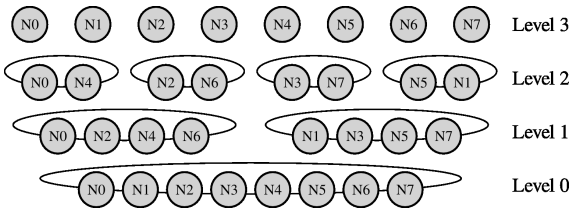
Figure 2.9 illustrates a simple Kademlia overlay with 8 nodes ($N0 \dots N7$), and the contents of the bucket table of nodes 011 and 101. To route a message from 011 ($N3$) to 111 ($N7$), $N3$ computes the XOR distance $011 \oplus 111 = 100$, and looks for entries in its table in the range distance of 4. Because 111 is not found, nodes 100 ($N4$) and 101 ($N5$) will be queried in parallel. Hopefully, $N5$ can return the address of 111, $N7$, so that $N3$ can successfully send its message as well as update its bucket table.

Joining and leaving To join the overlay, a node x contacts one of the existing nodes y and inserts it into the appropriate bucket. Successively, a node lookup is started on x to search for the key x: because y is the only available neighbor, x will start exchanging neighbors with it and thus gain knowledge of additional peers in the overlay. Liveness of nodes is monitored by checking the incoming messages and the successfulness of outgoing requests: references to nodes that stop communicating in the bucket table are removed following a *least recently seen* eviction algorithm.

Further research KADEMLIA is currently employed by several P2P file sharing [13] and content distribution [122, 12] architectures to support efficient keyword search.

2.3.8 SKIPNET

SKIPNET [140] uses a distributed approximation of *skip lists* [226] to implement a DHT with locality properties (a feature typically neglected in other systems). A skip list is a data structure composed of multiple levels of linked lists: at the base level all nodes of the list are present, at higher levels pointers enable to *skip* over elements at different granularity. SKIPNET allows for control of the location where the data is stored in the overlay, thus increasing availability and security. Nodes are referenced by their unique name identifier. Instead of using a list, the overlay is organized as a double-linked ordered ring of N nodes; each node stores a routing table containing $2log(N)$ pointers organized as $log(N)$ levels with 2 entries each. References at higher levels enable longer jumps (or skips) in the overlay. Pointers at level h in the table refer to nodes that are approximately 2^h hops to the left and right in the base ring. Instead of using a precise distance measure, these nodes are determined by splitting the ring at the lower level and probabilistically assigning nodes to the resulting rings. Figure 2.10 illustrates the resulting levels and rings in an example SKIPNET composed of 8 nodes. At each level, nodes are ordered by their name identifier in their corresponding ring. The latter determines the numerical identifier of the node, so that each node in a ring at level b shares the same high-order b bits of the numerical identifier.



Routing table N3

Level	Left	Right
0	N2	N4
1	N1	N5
2	N7	N7

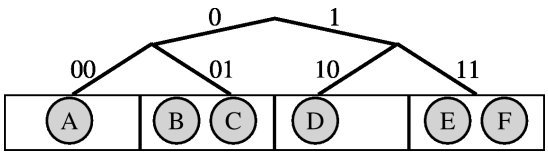
Figure 2.10: Example SKIPNET infrastructure and routing table for node $N3$.

Lookup procedure SKIPNET supports both routing by the name identifier or by the numerical identifier. To route a message by name, a node first checks the name identifier to see if the message has to be forwarded left or right according to the shared prefix. If the message and the node identifier share no common prefix, a random direction is chosen. Subsequently, at each hop, nodes forward the message to the farthest node whose identifier is not greater than the destination, by scanning the routing table starting from the highest level. In the example overlay shown in Figure 2.10, a message from $N3$ to $N6$ is forwarded to the left and to a referenced node at level 1, namely $N5$. To route a message by numerical identifier, the algorithm begins by looking for a node in level 0 whose numerical identifier's first digit matches the target's numeric identifier first digit. The algorithm then moves to the node's ring of level 1, and repeats the search by looking at the second digit. After a finite number of steps, the destination is found.

Joining and leaving When a node joins the overlay, it first needs to find the top-level ring that corresponds to its numeric identifier. This is achieved by routing a message to that numeric identifier. From this point on, the node retrieves its neighbors in the ring and in lower level rings by similarly looking for its name identifier. When a node leaves only the ring links at level 0 have to be repaired: this is performed by a repair process run either upon notification of the leaving node itself, or as soon as the departure has been detected by its neighbors.

2.3.9 P-Grid

P-Grid [14] builds upon a binary prefix tree (also known as *trie*) and uses prefix matching to resolve queries. Each peer is associated with a leaf in the tree, and is responsible for a set of keys composing its *key space partition*. The prefix of a binary representation of the data managed by a peer p determines its position in the tree, i.e. its path $\pi(p)$. In contrast to hierarchical solutions, the tree structure is not reflected in the actual topology connecting the peers. Accordingly, nodes have to maintain routing tables that point to nodes managing different subtrees thus different zones of the key space, and update them using an epidemic protocol. More specifically, each peer p stores references to other peers sharing a common path prefix of length l with p, but with the last bit inverted. To enable efficient range queries, content's keys are computed using an order-preserving hash function.



Routing table A

l	Prefix	Node
0	1	E
1	01	B

Figure 2.11: Example P-Grid and routing table for node A ($\pi(A) = 000$). Actual links are not determined by the structure of the binary tree but by the contents of each node's routing table.

An example of P-Grid is depicted in Figure 2.11: the key subspace is divided across peers according to the keys' prefix. Node A manages the 00 prefix, nodes B and C both manage the 01 path to increase fault-tolerance, node D is responsible for 10, while nodes E and F store data with prefix 11. The illustrated routing table of node A contains references to node E (with path 11, sharing no common prefix with $\pi(A)$), and B (with path 01 sharing a common prefix of length 1 with $\pi(A)$).

Lookup procedure To look up a key in the overlay a node first checks whether its path is included within the key bit string. In this case, the key is stored within the peer and the associated contents can be returned. Otherwise, the routing table is consulted and the request is forwarded to a node whose path better matches the key's prefix. The expected cost for a lookup operation is $O(log(N)$.

Joining and leaving When a peer connects to the overlay, the key space is divided and shared with the incoming peer. If two peers are responsible for the same partition of the

key space (i.e. they have the same path) they exchange references; if only a prefix of the path is shared, the peer with the shorter key path extends its key by taking over some of the keys.

2.3.10 Other approaches

Beside the systems presented in the previos subsection, we briefly review here other notable examples of structured overlays.

HyperCuP [251] employs a hypercube graph and guarantees that nodes are visited exactly once during a lookup operation (i.e. there is no retransmission of messages). Because HyperCuP merely proposes an overlay structure for efficient broadcast, it is not concerned with allocating data to nodes or routing requests to particular nodes as in a DHT. In this respect, efficient broadcasting in an overlay of N nodes can be achieved with $N-1$ message forwards. An extension of the protocol that uses semantic data to route messages and reduce network overhead is presented in [244].

Cycloid [261] employs a d-dimensional hypercube that forms a cube-connected cycle (CCC) [225] structure where each vertex is a cycle of d nodes. In comparison to other solutions, each Cycloid node has a small and constant degree in that it maintains exactly three connections: two cyclic connections and one cubical connection. This reduces maintenance costs in highly dynamic systems. An improvement over Cycloid that combines the CCC structure with a folded hypercube is presented in [187].

Kelips [137] divides nodes into k affinity groups $(0 \ldots k-1)$. Each node maintains references to nodes in its affinity group, in other affinity groups, and references to files shared by other nodes. This information is updated by means of periodically gossiping partial state information. In contrast to other structured approaches, Kelips is simpler, because there is no strict underlying topology (ring, hypercube, etc.) to be maintained. Lookup requests are progressively forwarded toward nodes that are closer to the target. Kelips has been used to implement a web caching mechanism, namely Kache [191].

Symphony [198] uses a ring structure mapping nodes onto the key space (like Chord). Nodes maintain a link to their successor and predecessor in the ring, and a number of long distance ring shortcuts. Shortcuts are chosen randomly according to a harmonic distribution, which results in large jumps in systems with few nodes, and short jumps in larger systems. The characteristics of the overlay reflect the *small world phenomenon* [206] (refer to Section 2.4.1), and its construction is based on the method proposed in [174].

2.3.11 Multi-attribute, range, and semantic queries

Distributed Hash Tables are very efficient in matching lookup queries, by finding the *value* univocally associated with a given *key*. A number of applications nonetheless depend on range or multi-attribute queries that look up for values that lie in an interval between two keys or are the union of different attributes. For example, in a distributed database of

geographical data, typical queries may involve finding *all points within a distance of* $100m$ *from a given location*. A possible solution to this problem is to divide the interval to be retrieved into a discrete number of points and initiate a query for each of those points. This nonetheless involves a trade-off between the efficiency of a search operation and the granularity of the results.

To support range searches in a Chord-like overlay, CHORD# [255], replaces consistent hashing with an *order-preserving* hashing. An order preserving hash function h ensures that if $a < b$, then $h(a) < h(b)$; range queries can thus be resolved by first locating nodes storing the values of a and b in the overlay, and then visiting all nodes between them. A routing scheme, named SONAR, that adds support for multi-dimensional range queries is presented in [254]. Other solutions enabling range-queries in CHORD are MAAN (Multi-Attribute Addressable Network) [60] and [138], which employ a locality sensitive hashing (LSH) [154] function to map similar data to nearby identifiers with high probability.

The work presented in [24] proposes an extension of CAN that enables efficient range queries for a grid middleware. The presented approach uses a space filling curve [247] (namely a Hilbert curve) as hash function: each peer is responsible for a subinterval of the domain $[0, 1]$, which represents the admissible attribute values that can be stored in the DHT. A range lookup first locates the zone containing the middle point of the requested interval, and then propagates the request to close-by zones until all points in the interval have been found.

As described in [235], prefix hash tries (PHT) can be adapted to address range queries; this technique has been successfully implemented in P-GRID by means of an order-preserving hash function to generate content's keys [90]. Skip graph based structures have also proved to be a viable solution for both range and multi-dimensional queries: beside the previously cited SKIPNET [140], examples include SKIPINDEX [308] and SKIP TREE GRAPH [132].

MERCURY [39] implements a query routing mechanism that supports multi-attribute and range requests. Nodes are grouped into hubs that cluster all the data related to a certain attribute, and queries are routed toward the hubs responsible for their contents. Furthermore, a ring topology is used to connect nodes within hubs, enabling efficient range query resolution.

Other solutions that enable range and/or multi-dimensional queries in DHTs are: [218], implementing range-queries over the BAMBOO DHT [237], [34], presenting an extension of PASTRY to supports range queries, [168], extending CHORD with support for range queries and load balancing, [91], proposing a recursive partition search method, [186], describing a distributed search scheme supporting both range and multi-attribute queries, and [192], introducing support for multi-dimensional complex queries by means of R*-trees [35]. Finally, a comparative analysis of common DHTs with support for multi-attribute and range search systems is presented in [260].

Semantic queries can be viewed as a natural extension of multi-attribute and range queries. A notable drawback of the latter is the lack of a notion of semantic similarity, thus words like *car* and *vehicle* are not considered as related. In [277] two solutions for semantic-based full-text searches based on CAN are proposed, employing vector space model (VSM) and latent semantic indexing (LSI). Documents are organized so that related documents are stored closer in the key space. In the VSM solution, each node is responsible for storing references to certain keywords, thus a document retrieval operation

is decomposed into several lookups for each of the search terms. In the LSI solution, the semantic vector is mapped into coordinates in the CAN space: document retrieval first locates the corresponding point, and then multicasts the request to (semantically) nearby nodes. Another solution [59] extends MAAN [60] to support RDF[1] meta-data storage and retrieval.

2.4 Unstructured solutions

Unstructured systems are not based on deterministic overlay topologies and do not enforce precise rules on the placement of data in the overlay. Accordingly, the information shared by nodes and the logical connections between them are unrelated. To some degree, unstructured systems are freely connected overlays that mimic social relationships between peers, and where information retrieval relies on multicast communication. While less efficient than DHTs, unstructured solutions are simpler to maintain and allow for more complex queries such as *free text* search. Because unstructured peer-to-peer topologies are not created and maintained by a deterministic process, it is often difficult to understand their dynamics in order to enable efficient communication and robust operation. Nonetheless, research has come up with models that replicate the characteristics observed in real-world *complex* networks and enable a deeper understanding of their features. Accordingly, before reviewing existing peer-to-peer solutions, we briefly present common models of complex networks; an in depth discussion on complex networks characteristics can be found in [87].

2.4.1 Complex network topologies

Unstructured networks are examples of complex networks where there is no apparent structure. While we refer to unstructured solutions as overlays constructed without relying on a deterministic algorithm, there exists some degree of control over the desired characteristics of the resulting topology. Thus, while it might not be possible to recognize regular patterns in the underlying graph, a coarse classification of unstructured topologies based on main features is nonetheless possible. The graph's degree distribution is one of such features, and it is the principal measure of analysis of complex networks.

Random graphs (Erdös-Rényi) Random networks [265, 105] can be considered as the simplest example of complex networks [87]. A generative model for constructing random networks by connecting all pair of nodes with uniform random probability has been introduced by Erdös and Rényi [105]. Random graphs exhibit small diameters [45, 46] (of logarithmic or polylogarithmic growth), and node degrees following a binomial distribution [19, 156]. Whereas initial studies on random networks aimed at developing mathematical models for studying real world phenomenons, it was later proved that most real-word networks cannot be satisfactorily represented by the Erdös-Rényi model. The major issues are related to the different degree distribution and smaller clustering coefficients to what can be observed in real networks. Research showed that it is nonetheless possible to algorithmically create models matching or approximating real networks degree distributions

[1] http://www.w3.org/RDF

[216] by employing different probability distributions [214] or construction models based on rewiring methods [208].

Small world networks (Watts-Strogatz) The "small-world phenomenon" [31] was first observed by Stanley Milgram during his social studies in the 1960's [206]. By means of a simple experiment, it was shown that people are typically linked by short chains of acquaintances; that observation gave birth to the myth of "six degrees of separation" [293]. Beside short average path lengths, small world networks are characterized by high connectivity within small group of nodes (high clustering coefficient). In particular, this latter condition differentiates graphs with small-world properties from other random graphs. Graph-theoretic analysis of small-world networks led to the development of generative models [213, 215], such as the popular one proposed by Watts and Strogatz [292], as well as distributed algorithms to rewire an existing network into one with small-world characteristics [101]. Applications of small-world networks for computer networks have been analyzed in [174, 175, 249]; in this respect, the problem of *navigability* of such networks is of particular interest for routing information using the local information of each node. A small-world network of N nodes is said to be *navigable* if a decentralized routing algorithm, exploiting only local information and information about the target node, enables routing in a number of steps proportional to $log(N)$ [174].

Scale-free networks (Barabasi-Albert) Scale-free graphs [185] model networks with power-law degree distribution, where a large number of vertices have small degrees, and few vertices have very large degrees (*hubs*). As proposed in [32], scale-free graphs can be constructed by a random process where links are added between nodes using preferential attachment: the probability of creating a new connection to a node is proportional to its current degree. Scale-free networks inherit from random networks the characteristic of small diameters [80, 79], but the different degree distribution provides a better match for many examples of real-world networks, e.g the Internet, air traffic routes and airports, etc. Several research studies have analyzed the robustness of scale-free networks against random node failures and vulnerability to targeted attacks [20, 135, 85]. It has been shown that while scale-free graphs are more robust against random faults than random networks of comparable size, the latter better cope with targeted malicious attacks [47]. This issue is related to the presence of hubs that, if targeted by an attacker, quickly compromise the connectivity of the whole network.

2.4.2 Search in unstructured overlays

Search in early peer-to-peer systems such as NAPSTER [9] relied on centralized indexes that provided the requesting peer with addresses of nodes storing the desired content; actual transfer of data was carried out by peer-to-peer interaction between nodes. In contrast to DHTs, in unstructured overlays it is not possible to easily determine which peer shares the desired information, thus fully distributed search is typically performed with a *flooding protocol* [94]. Flooding involves sending the query, which describes the search parameters such as the name of a file, to some nodes in the overlay (typically the topological neighbors of the node initiating the request). Recipients locally determine

if the request can be fulfilled (based on their own content) and eventually respond to the initiating peer. Otherwise, the query is further forwarded to other nodes. To avoid forwarding queries for an undefined number of steps, each node can store a reference to recently processed messages, and avoid retransmission if the same message has already been received. Moreover, because the size of the network is not known, each message has an associated lifetime (Time-To-Live, or TTL, Hops-To-Live, or HTL) value which determines the maximum number of times it can be forwarded. While being simple to implement, basic flooding has a number of drawbacks. Because of its non-deterministic nature, flooding cannot guarantee that all nodes that own the queried objects will be contacted, likewise most of the nodes processing a query will likely not be able to fulfill it. Beyond that, because of the topology of the network, flooding results in an exponentially growing amount of messages, which is aggravated by retransmissions that may occur in topologies with many redundant links [195].

The number of nodes contacted can be limited by employing different traversal and broadcast policies. Because it is often not necessary to reach all nodes in the overlay to fulfill a query (the answer might just be found by contacting few close-by peers), it is worthwhile to avoid forwarding to all neighbors or for an extensive number of hops. Meanwhile, adaptive overlay networks [181, 124] or hybrid topologies (i.e. super-peers [302]) can be employed to limit the problem of message retransmission and reduce overhead.

In the following, some existing improvements will be reviewed: in this respect, it is possible to make a distinction between *uninformed* (*blind*, or *state-less*) methods and *informed* (*heuristic-based*, or *state-full*) ones [283]. Uniformed methods do not rely on semantic information and act upon the flooding mechanics (TTL, number of contacted neighbors, etc.). On the contrary, informed methods make use of heuristics based on semantic information about the query to direct the search toward peers that are most likely to provide an answer. An in-depth review and analysis of search methods for unstructured overlays can be found in [241, 306, 283].

Traversal techniques A traversal technique is the algorithm that defines the order in which nodes are visited during a query operation. Two common algorithms exist: *breadth-first* and *depth-first*. Breadth-first traversal visits nodes at progressively increasing distance, up to a predefined depth (TTL). In the example network depicted in Figure 2.12, a breadth-first query initiated at node A with TTL equal to 2, will first be forwarded to nodes Z, C, E (one hop distance from A), then to P, Q, F, S, R, B (two hops distance); nodes N, K will be omitted because they lay at a three hops distance from A. Depth-first traversal contacts nodes in one direction at time with backtracking. In the example in Figure 2.13 the visiting order for a depth-first query initiated by A might be $Z, F, Q, P, C, E, B, R, S$. Breadth-first traversal can achieve good response time because queries can be easily parallelized (nodes at each level can be visited at the same time), but is prone to generating more traffic [196]. Conversely, depth-first search is more efficient but can result in longer delays [167].

Iterative deepening If the requested information has a high probability of being found near the requesting node, flooding with increasing depth, also known as iterative deepening, can significantly reduce the overall traffic [195]. Queries are first broadcasted with

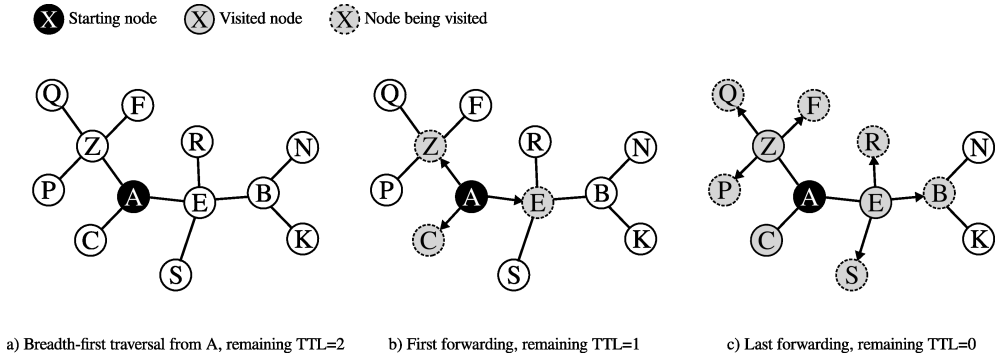


Figure 2.12: Example breadth-first traversal in an unstructured topology, with Time-To-Live equal to 2.

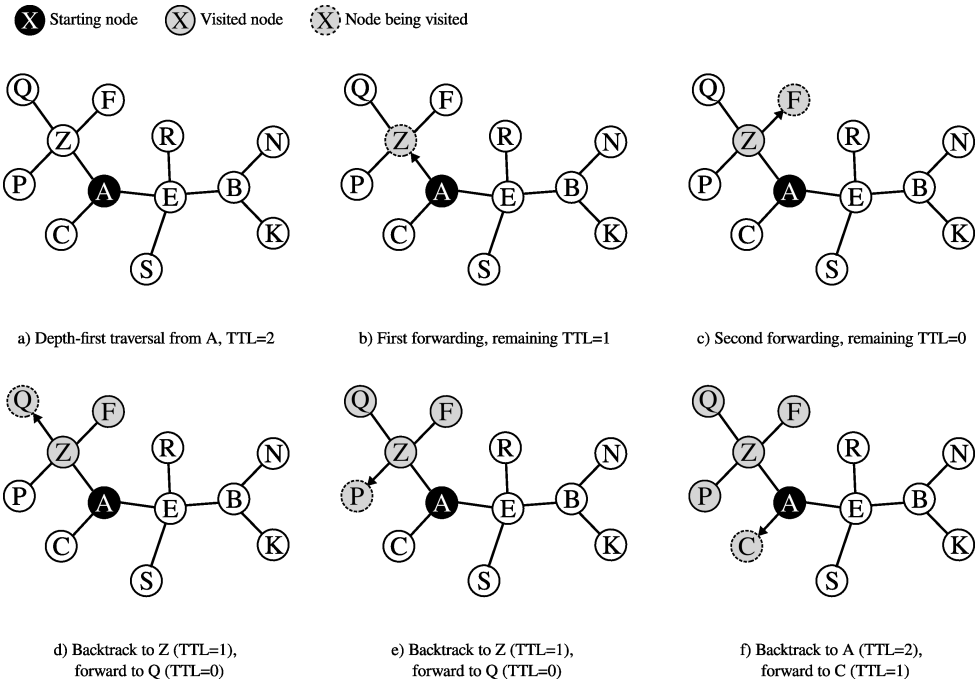


Figure 2.13: Example depth-first traversal in an unstructured topology, with Time-To-Live equal to 2. Last steps omitted for simplicity.

small TTL values, using a breadth-first approach; the TTL value is progressively increased until either a result is found, or an upper limit is attained. A more advanced solution that dynamically adapts the TTL according to the popularity of the searched content is presented in [162].

Random walks In contrast to basic flooding, *random walk* [195, 127, 199, 239] forwards the query to just one neighbor at time. The query randomly *walks* on the overlay until the target information is found, a pre-determined maximum number of hops has been reached, or the information has been found by some other walk. In the latter case, peers have to contact the originating node and check if the query still needs to be forwarded

or not [195]. While this approach significantly reduces the amount of traffic produced by search messages, the response time increases because the probability of obtaining results within an acceptable number of hops is reduced [129]. An improvement of this technique involves starting k multiple concurrent random walks [195] (*k random walkers*) in order to increase the probability of hitting a target. In [41, 42], random walk parameters are adapted according to the popularity of the searched content. Furthermore, random walks can be combined with shallow flooding (i.e. flooding with small TTL) to provide information about nearby nodes [16, 205, 128], hence both increasing the probability of success and reducing response time. Finally, [310] studies the convergence of random walking in different types of networks and with different random neighbor selection distributions. An example of random walk and k-random walk is depicted in Figure 2.14 a), respectively b): in each step, each *walker* is forwarded to a random neighbor.

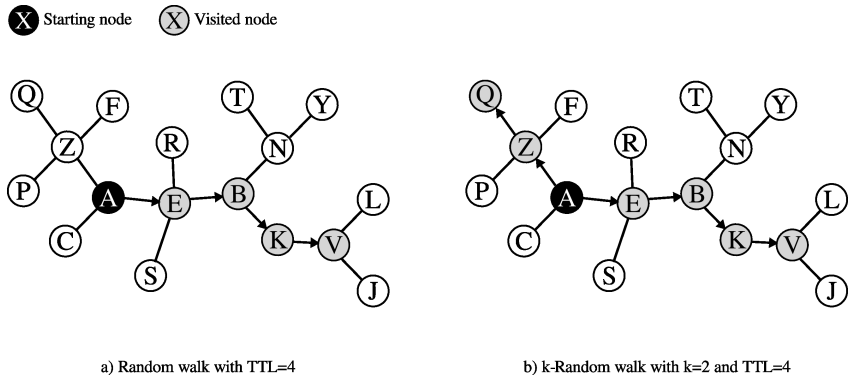


Figure 2.14: Example random walk (a) and k- random walk (b) in an unstructured topology, with Time-To-Live equal to 4 and k=2.

Teeming Flooding protocols visit a large number of nodes at each forwarding step because all neighbors of the current node are contacted. If the queried object is very popular across the network, forwarding the query to just a smaller number of neighbors still has a high probability of retrieving it. On this basis, probabilistic flooding protocols (or teeming) [167, 94] forward the query only to a random subset of all available neighbors on each node, namely they forward to each neighbor according to a fixed probability. A further improvement of this technique, called *teeming with decay* [184], involves reducing this probability as the number of hops increases. While teeming also results in an exponential growth of the traffic as the query travels deeper in the network, the growth is slower than in pure flooding.

Selective forwarding Teeming reduces the amount of traffic by limiting the number of visited neighbors, regardless of the fact that ignored peers might be able to fulfill the query. If some information about neighbor nodes is known *a priori*, the forwarding algorithm might be able to choose which peer is best suited to send the query to: this technique is known as *selective forwarding* or guided search. In [16, 290] queries are routed toward nodes with higher degrees. Similarly, in [219] nodes probe their neighbors before forwarding the query, in order to find the one with the shortest round-trip time. In [83], the authors

present a search mechanism that uses compound routing indices to select the best query routing path. Such indices define the goodness (i.e. probability of finding a matching document) of each outgoing path concerning a given topic, and are periodically updated by aggregating information about shared documents on nodes in each path. Following a similar principle, other solutions [153, 224] employ Bloom filters [56] to determine which forwarding path is more likely to lead to the queried object. Another approach [284] determines the desirability of a neighbor based on previous interaction: if forwarding to a given neighbor results in a success its goodness is increased, otherwise decreased. The appealingness of a neighbor can also be computed by combining several metrics, such as the communication cost, the node's degree, or the amount of shared information [313, 268, 298, 163].

Replication All aforementioned search techniques involve a trade-off between the probability of fulfilling a query and the resulting network traffic. In this context, replication plays an important role in improving the effectiveness and efficiency of search methods. Whereas in file sharing networks content might be naturally replicated by the interaction between peers (for example, a popular song downloaded and then shared by a large number of users), in other peer-to-peer systems an active replication mechanism might be needed. We note that replication can either involve a full replica of an object, or just a reference to the node storing that object. The simplest replication strategy is *one-hop replication*, where each node knows its neighbors' identities or shared resources, and can thus reply to queries on their behalf. Several replica allocation strategies have been thoroughly analyzed in [78]: uniform, proportional to the number of requests (which has been further analyzed in [278, 279]), and proportional to the square-root of the query rate. While the first two lead to comparable results, the latter yields optimal performance. An evaluation of distributed replication algorithms that converge to square-root allocation has been conducted in [78], and on different network topologies in [195]. In the latter, two easily implemented replication methods are also proposed: *owner replication*, where upon a successful search the object is replicated on the requesting node, and *path replication*, where the object is replicated on all nodes along the path between the providing and the requesting node. The benefits of replication in random-walk protocols are also illustrated in [242]. Other replication strategies are discussed in [280], where a replication strategy based on the popularity of the content and employing an optimal replica placement mechanism is proposed. It is important to note that replication can also be beneficial to structured peer-to-peer systems, as shown in [229, 233].

Topology optimization The topology of the network has great influence on the efficiency of a search protocol [70, 111]. Even without introducing structure into the network, it is possible to optimize the overlay in order to support efficient forwarding. In this context, several studies [181, 288, 139, 124] have proposed solutions for efficient multicasting, called *topology-aware* or *proximity-aware*, that adapt the overlay at runtime to match the underlying topology. These solutions minimize both retransmissions at the network level and delays. Other solutions employ super-peers [302] to reduce flooding traffic by creating a two-level overlay where a small number of high-capacity peers are elected to cache information and route queries for a large number of normal peers. A proximity-aware overlay based on super-peers has been presented in [161].

Other improvements Beside the aforesaid methods, there exist other techniques to improve the efficiency of search in unstructured networks. With *non-forwarding* search schemes [303, 188] each peer maintains a local cache with a number of addresses of other peers. To maintain this cache, each peer periodically selects an entry and sends to the corresponding node some of the addresses in the cache, similarly to the operation of a gossip protocol. When a search query is initiated on a peer, each node in its cache is iteratively probed for a result; when a peer is probed it returns a sample of its cache, providing new entries in the cache of the probing peer. Because search is managed locally by one peer, the generated traffic can be accurately limited and does not result in exponential growth as with flooding.

In [84], the authors propose to exploit the semantics associated with contents shared by each node to construct different overlay networks for each possible topic. Nodes may join several overlays, and queries can be efficiently resolved by forwarding them to the appropriate overlay. A similar approach based on the creation of semantic groups is discussed in [77]. In [73] a self-organized solution that extends the topology of the overlay according to the results of previous searches, hence promoting the emergence of strongly connected semantic communities, is presented. A number of outgoing links on each peer point toward nodes that are more likely to share common interests, thus providing paths to quickly resolve future queries. Other approaches [114, 291] cluster semantically similar information in order to increase the recall rate once a result for the query is found. As with other resource discovery improvement techniques, clustering can also be applied to structured overlays, as in [147, 113].

The bio-inspired solution proposed in [204] uses pheromone trails that are laid on the overlay and are linked to search topics. Queries are routed on the overlay following the path with the highest pheromone concentration; depending on the success of the query, the concentration on a path may be further reinforced, or not. Similar bio-inspired routing mechanisms are presented in [296, 108].

In the following we review some popular unstructured systems, and highlight their functional design.

2.4.3 Gnutella

The Gnutella protocol [3, 238], was quickly developed after the demise of the Napster [9], to create a replacement for the popular file-sharing network. Napster was the first successful deployment of a hybrid peer-to-peer file sharing system that relied on a centralized indexing server and decentralized content provisioning. Legal issues led to the shutdown of Napster servers in 2001 [133], rendering the network unoperable. In order to overcome the weakness of the latter, Gnutella proposed a fully decentralized search protocol based on flooding that removed the need for a centralized indexing service thus overcoming the risk of further shutdowns. Later versions of Gnutella implemented a superpeer infastructure, in order to reduce the overhead of flooding. In the following, we review both basic Gnutella protocol [4], as well as improved developments such as [5] and [69].

Message routing GNUTELLA employs just five types of messages: Ping, Pong, Push (to request the transmission of a file), Query (to search for a file), and QueryHit (to successfully respond to a Query). Pong and QueryHit messages are sent in reply to Ping, respectively Query messages. All messages exchanged by GNUTELLA peers are additionally labeled with a unique identifier and contain a TTL value to limit the number of times they can be forwarded. This identifer helps detecting and avoiding possible message retransmissions, as well as enabling routing of responses. Concerning routing, the protocol requires all response messages to be sent along the same path followed by the request; hence, peers maintain a routing table that stores the identifiers of the received packets, their source peer, and their destination peer if they have been forwarded. If a node receives an unexpected reply message (Pong, QueryHit, or Push) it will not further forward it.

Joining and leaving Peers can join the GNUTELLA network by connecting to a node already in the network. A cache server supports the bootstrap process by providing a list of peers. The node requests the connection and if accepted becomes part of the network. Each node periodically sends a Ping message to each neighbor: upon receiving a Ping, a node replies with a Pong message that contains its address to notify its presence, and then forwards the Ping on the network (up to a predefined distance). Pong replies are routed back along the same path that carried the corresponding Ping. With Pong messages nodes can thus discover new peers and create new connections.

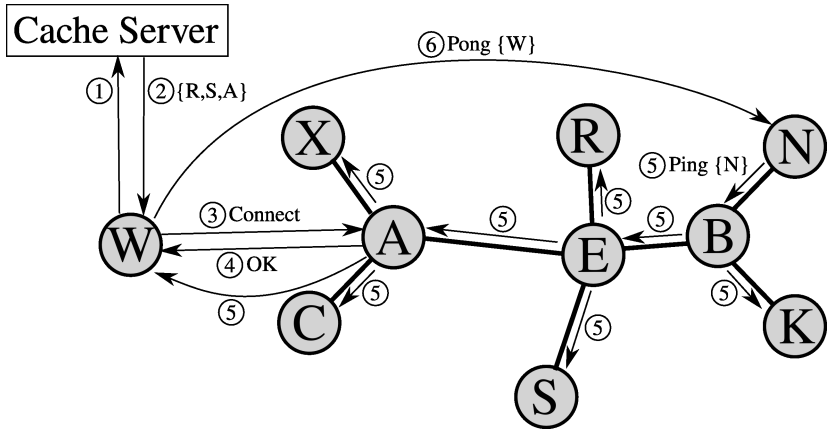


Figure 2.15: Joining and Leaving the Gnutella network.

An example of the process of joining the network is illustrated in Figure 2.15. Node W first requests a list of peers from the bootstrap cache server (steps 1 and 2); W then tries to connect to a random node from the list (3), A in our example. The connection is accepted (4), and W becomes part of the overlay. At some point, N pings its neighbors (5): each node receiving the Ping message replies with a Pong that is routed back to N, which finally discovers W (6).

Search GNUTELLA 0.4 uses a breadth-first flooding protocol to forward its requests on the overlay encapsulated in a Query message. Nodes that share a file matching the request can reply with a QueryHit message. To request a file transfer, the requesting node replies

to the QueryHit with a Push message. Actual transfer of the file is achieved using the HTTP protocol.

Since protocol version 0.6 [5], GNUTELLA employs a super-peer approach, by grouping peers into leafs and ultrapeers. Each leaf peer is connected to several ultrapeers, to which it sends its shared keywords, whereas ultrapeers are connected together. The superpeer design reduces traffic and the number of hops traveled by each query, improving response time. The GNUTELLA2 protocol [6] (a fork of protocol version 0.6 that employs random-walk instead of flooding) is also based on superpeers.

Further research The GNUTELLA protocol has been the subject of several studies aimed at understanding the dynamics of peer-to-peer interaction, as well as at providing adjustements to increase the effectiveness and efficiency. In depth analysis of GNUTELLA networks are the focus of [238, 15, 270]; moreover [305, 26, 212] give insights about the security of GNUTELLA: in particular, concerns such as Denial Of Service (DOS) attacks and malicious content spreading are analyzed, and solutions are discussed.

Improvements of GNUTELLA have been considered by GIA [69], which tackles the problem of search efficiency, and proposes to incorporate a number of techniques to ameliorate the scalability of the system. More specifically, GIA employs biased random walks that steer queries toward nodes with higher degree, one-hop replication, topology adaptation to ensure that only high-capacity nodes have high-degrees, and traffic control to adapt the network load to the capacity of each node. GIA developers prove that the proposed solution enhances the overall operation of the system by significantly reducing the traffic, while retaining the simplicity and flexibility of GNUTELLA.

2.4.4 FREENET

The FREENET project [75] aims at creating a distributed document storage on an unstructured overlay with small-world characteristics. The network operates on the principle of a *darknet* [40], and enables participants to store and retrieve document anonymously. FREENET exploits the fact that small-world networks are navigable [249] to ensure the convergence of the employed greedy routing protocol.

Message routing Nodes and shared objects are univocally identified using hash keys. Nodes' keys are randomly generated, whereas objects' keys are an hash of their contents. FREENET uses key-based routing to insert or retrieve content. With the help of a routing table maintained by each peer, object queries are routed toward the node with the closest matching identifier. The routing table is updated when receiving query replies. As the size of the table is limited, a Least-Recently-Used algorithm is employed to cleanup old entries; an enhanced entry replacement algorithm has been proposed in [308].

Joining and leaving Nodes join the overlay by contacting some nodes in the overlay; the latter add the newcomer's identifier in their routing tables. With time, the node will hopefully receive requests to publish files and reply to queries that closely match its identifier.

Search To search for a file, the requesting node computes the key of the file and sends it to itself. Each query has an associated expiration time (TTL): if the query is not answered within its TTL, it is considered as failed. Upon reception of a query, a peer checks in the local storage to determine if it owns the requested object. If the file is not found in the local storage, the query is forwarded to the node in the routing table associated to the key closest to the requested one. If a node that has already been contacted is reached, the request is returned to the previous node, which then tries to contact the peer associated with the next closest key in the routing table. When a peer sharing the requested file is found, a success message is sent back along the path traveled by the query until starting node. Each node traversed by the reply updates its routing table and stores a copy of the file.

To publish a new object, the request is routed similarly as if the object is being queried: each traversed node checks against collisions (existing objects with the same identifier). If no collisions are detected, the object is published on each node along the path.

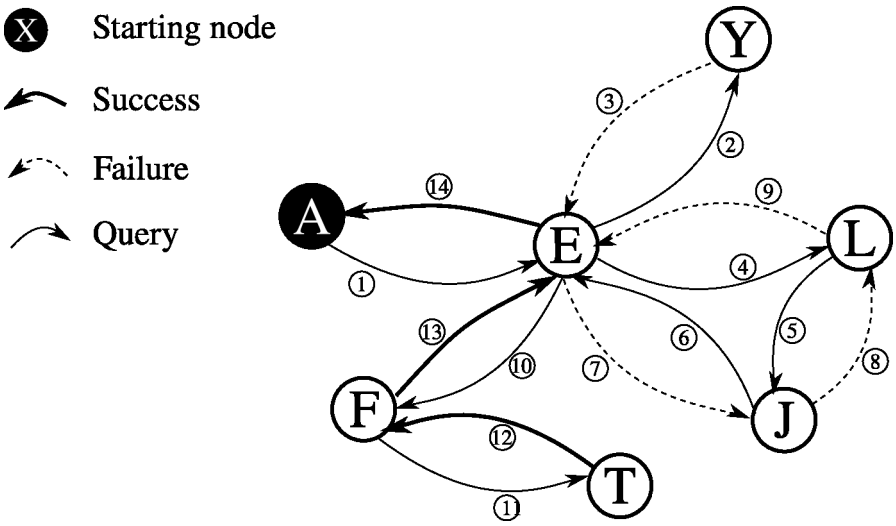


Figure 2.16: FREENET query routing.

Figure 2.16 depicts an example of messages exchanged during search in a FREENET overlay, numbers indicate steps in the process. We suppose that a user request for an object with key T. The request is handled over to node A, which forwards it to the closest known matching node, E. The query fails, and E forwards it to its closest matching node, Y, failing again to locate the object. The node then forwards if to the second-closest node, but this subsequently results in the query returning to E. The query is finally forwarded to F, and finally to T (where the requested object is found). The reply is forwarded back to A on the same path as the query.

Further research A change of the routing protocol to improve its efficiency has been proposed in [74]: the idea is to exploit information about response times, time to estabilish a connection, and success rate, to select the best path to forward a query to. While the latter indeed results in better routing performance [249], the proposal for such a new protocol was subsequently withdrawn in favor of a simpler approach in FREENET 0.7.

2.4.5 Kazaa/Fasttrack

Fasttrack [7] is a proprietary file-sharing protocol based on a super-peer architecture, and employed by the popular Kazaa client [8]. Due to the closed and encrypted nature of the protocol, precise information is relatively scarce; nonetheless, reverse engineering through traffic sniffing and analysis [182, 189] enabled a better understanding of the communication between ordinary peers and superpeers and the development of opensource clients [2]. In particular, our review is based on the information provided in [189].

Joining and leaving At startup, ordinary nodes probe the connection with several candidate superpeers and retain the best suited one. It is assumed that FastTrack takes locality and workload into account, by having ordinary peers preferably connected to closeby superpeers whose workload is low. The workload of a superpeer is related to the number of connections it maintains with ordinary peers; selection of the parent superpeer is based on this measure and provides a load balancing effect. After the initial connection, a peer receives list of additional superpeers that are cached for later use.

Fasttrack promotes ordinary peers to super-peers when higher performance and connectivity than globally defined thresholds are detected; superpeers maintain connections with each others and form the backbone of the system.

Search Superpeers index the content shared by normal peers: each object is identified by its hash. To search for a file, an ordinary peer sends the query to the superpeer to which it is connected. After the latter replies, the peer connects to other superpeers to gather additional results, and remains connected to the last contacted superpeer. On superpeers, queries are resolved by consulting the local index and by eventually contacting other superpeers: the traffic analysis in [189] reveals that superpeers do not exchange their indices.

2.4.6 Saxons

Saxons [262, 263] maintains an overlay with low latency, high bandwidth paths, as well as small distances. The overlay provides efficient multicast communication across the overlay that can be exploited to deliver higher level services. In addition to overlay neighbors, each node maintains a dynamically changing set of peers' addresses, a number of which is periodically sent to neighbors; the identifier of the node sending the information might be within the transmitted list, allowing the node to spread its identity across the overlay. Nodes periodically measure the latency and bandwidth of known nodes, and add them as active neighbors (possibly replacing existing ones) according to the desired structure quality and the maximum allowed node degree.

It is important to stress the fact that Saxons does not implement any specific querying mechanism, but leaves the choice to applications implemented on top of it. Nonetheless, the authors experimented with a Gnutella-like flooding protocol and obtained reduced latency and increased bandwidth compared to a random overlay built on the same underlying network.

Joining and leaving A node connects to the overlay by acquiring a random list of nodes from the local set of an existing (bootstrap) node, and subsequently trying to establish connections with peers in such list. The node then starts periodic exchanges of its local set with other peers, in order to gain knowledge about the network.

2.4.7 UMM

UMM [239, 240] (which stands for *Unstructured Multi-source Multicast*) uses a self-organized adaptation mechanism to optimize an unstructured overlay with the goal of improving bandwidth and reducing communication latency in multi-source multicast communication. UMM uses a two layer architecture separating the tasks of maintaining a base overlay and of disseminating the information in an efficient way. Connections in the base overlay are arranged both to reduce latency and to increase available bandwidth, similarly to SAXONS.

UMM constructs and maintains efficient multicast distribution paths by detecting and avoiding duplicate traffic. The system monitors incoming traffic and checks for duplicate messages; if duplication is detected, the source of the message is informed not to forward further messages through the same path (called tunnel). The connection between two nodes is not permanently removed, but temporarily filtered; to prevent partitioning of the overlay in the event of a crash, filters are reset when failures are detected.

Figure 2.17 illustrates the duplicate message detection: in the first step (a), A multicasts its message to its neighbor E, with a latency of $70ms$. The message is further transmitted from E to S and B with a latency of $40ms$, respectively $210ms$ (b). In the last step (c), S forwards the message to B; the latter will finally receive the message from both S and E: detecting the duplication, B will ask E to filter the tunnel (E to B), thus avoid using it for forwarding messages from A.

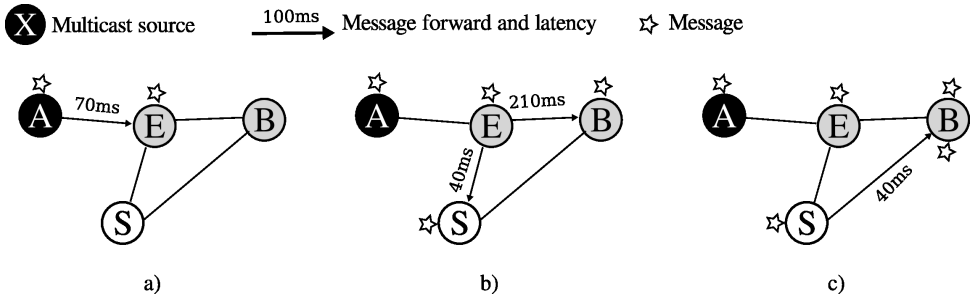


Figure 2.17: UMM duplicate message detection.

Joining and leaving When a node connects, it gathers the addresses of a number of peers by contacting a node in the overlay. These addresses define the initial neighbors for the base overlay. Information is further exchanged with other nodes by means of an epidemic protocol. Periodically, an optimization process measures the link quality for a random subset of known nodes, in order to determine tunnels optimized for latency and tunnels optimized for bandwidth.

2.4.8 PHENIX

PHENIX [10] constructs a low-diameter overlay with power-law degree distribution with the goal of offering faster query response time. Furthermore, the identity of high-degree nodes is hidden, to prevent malicious users from attacking them, and increase resiliency.

Joining and leaving A node n_i wanting to connect to the PHENIX overlay requests a list of addresses of peers in the overlay from a cache server. The list is divided into two subsets, G_{random} and $G_{friends}$. The node n_i then sends a ping message to all peers in $G_{friends}$: upon receiving the ping, peers reply with a pong message that contains the list of their own neighbors. The ping message is then forwarded one more step into the network: nodes receiving it, add n_i to a temporary list Γ. All neighbors of nodes in $G_{friends}$ are inserted into a $G_{candidates}$ list, which is sorted according to the frequency of appearance. The topmost nodes are thus the nodes that are most known in the network, and are used to create the $G_{preferred}$ list. The final neighbors of n_i are the union of G_{random} and $G_{preferred}$; subsequently, for each neighbor node in $G_{preferred}$, n_i tries to establish a connection: if accepted, the identifier of the node is moved to the $G_{highlypreferred}$ list, respectively n_i is added to $G_{backward}$ on the accepting node. A node may refuse a connection because the maximum number of neighbors has already been reached.

Thanks to the neighbor selection process, nodes that have a high-degree will preferably chosen as neighbors by incoming nodes. Random neighbors are nonetheless kept to improve resiliency.

Resilience to attacks PHENIX employs different mechanisms to protect the network from targeted attacks. On one side, the system attempts to hide the identity of high-degree nodes; on the other side, a node maintenance procedure recovers the network in the event of an attack. To conceal high-degree nodes, recurrent ping messages or malformed pings (for example, with TTL greater than 1) are silently dropped by the system. Furthermore, the $G_{backward}$ list is never disclosed in the list sent in response to a ping message, thus preventing nodes from gaining knowledge of the *popularity* of the node. The node maintenance procedure is used to probe for peers that may have left the system, and create new random or preferred connections.

2.4.9 NEWSCAST

NEWSCAST [159] employs a simple epidemic protocol that results in the emergence of a small-world network. The topology is determined by the list of addresses maintained by each peer, which is periodically exchanged with other peers. Beside a small diameter and a high clustering coefficient, NEWSCAST overlays exhibit resilient behavior in failure situations, even when a large portion of the peers simultaneously disconnect.

Cache merge Each node maintains a cache containing the identifiers and addresses of n other peers in the overlay. Each cache entry is associated with a timestamp that determines the age of the entry. Information contained in the cache is shared with other peers by means of an epidemic protocol [158]. Periodically, a node selects a random entry in

its cache, contacts the corresponding peer and initiates a cache merging operation. Cache merging consists in copying the contents of the two peers' cache and retaining at most the $n-1$ newest entries according to their timestamp. These entries constitute the new cache for both peers partecipating in the merge. To complete the merge, peers add an entry corresponding to each other in the cache with updated timestamps. The merge operation enables nodes to create new contacts, flushes old entries, while retaining a resilient and connected overlay.

A Cache

id	timestamp
S	45
P	53
Q	39
R	44

S Cache

id	timestamp
X	37
K	77
C	15
D	33

Figure 2.18: Before merge

Merged Cache

id	timestamp
K	77
P	53
S	45
R	44
Q	39
X	37
D	33
C	15

Figure 2.19: Merging

A Cache

id	timestamp
K	77
P	53
R	44
S	90

S Cache

id	timestamp
K	77
P	53
R	44
A	90

Figure 2.20: After

Figure 2.21: NEWSCAST cache merging operation.

Figures 2.18, 2.19, 2.20 illustrate the merging operation initiated by node A at time 90. Node S is selected as candidate for the merge, the contents of A and S cache are unified, and the most recent entries are retained. Node A is inserted in the resulting cache of S, whereas node S appears in A's cache.

Further research Epidemic membership management protocols have been also used in other systems, such as CYCLON [285] and T-MAN [157]. In particular, T-MAN replaces the random entry selection employed during cache merges with a deterministic choice based on a ranking function; accordingly, a control on resulting graph can be asserted to produce sorted or clustered topologies. Moreover, a semantic based overlay built on top of a CYCLON overlay is presented [287]. A general overview of gossip protocols for distributed systems can be found in [173].

2.4.10 Other approaches

As with structured overlays, beside the ones reviewed in the previous sections, a number of other unstructured designs exists. Among the interesting solutions, [311] presents a Gnutella-like system that employs topology adaptation to organize peers into semantic groups, whereas [252] proposes the construction of a Gnutella-like overlay that optimizes flooding by reducing the number of small cycles. In [65] the authors introduce a hybrid solution that employs unstructured search methods (flooding and random walks) on top

of a structured overlay (PASTRY): in contrast to a random overlay, a structured solution avoids redundancy and enables finer control of the nodes visited by a query.

2.5 Peer-to-Peer churn

Peer-to-peer networks are dynamic systems where peers continuously join and leave. Each peer remains connected to the system for some amount of time, defined as *session* [271]. Churn is determined by the dynamics of peer activity, namely the frequency of joins and leaves and the length of peer sessions. Understanding churn and its effect on the reliability of a network is essential for the deployment of robust and predictable large scale systems.

Research on churn in peer-to-peer networks has focused on analyzing this effect in real-world structured networks [29, 130, 248, 271], as well as in unstructured ones [304, 183, 38]. Several studies have highlighted the difficulty of precisely measuring the dynamics of a live system. Some of the achievements concern the development of statistical models to describe or predict the behavior of distributed systems under churn. In [248], the authors provide an analytical study of CHORD's performance and validate their result by means of simulations. Comparative analysis of different DHTs in [160, 236] highlight the cost of maintaining proper operation under churn and the negative effects of short session times. In particular, heavy churn results in either failed lookups requests (PASTRY) or increased latency (CHORD).

The *resilience* of a peer-to-peer system relies in its ability to cope with churn. In [136] three aspects of resilience to churn are identified: data replication, routing recovery, and static resilience. Routing recovery strategies can either be reactive or periodic [236]: whereas reactive recovery takes places only when a failure has been detected, periodic recovery involves a continuous exchange of information between nodes, regardless of the detected changes in the network. While consuming less bandwidth under low churn, reactive recovery becomes more expensive as the dynamics of the network increase, and can create positive feedback cycles if the network becomes congested. In particular, congestion may lead a node to think that a neighbor has failed, and the subsequent recovery can worsen the situation by increasing the traffic. To solve this problem, the use of a periodic recovery strategy combined with a more conservative reactive recovery has been suggested [236]. Static resilience refers to the ability of the network to avoid failure or partitioning even before recovery actions take place, for example using redundant links in an unstructured network.

It is often argued that unstructured solutions are more robust toward the effects of heavy churn; the results presented in [66] reveal that it is nonetheless possible to create resilient structured solutions. However, the proposed concepts further complicate the maintenance of structured solutions; moreover the search efficiency of the latter might be counterbalanced by real-world churn rates [250, 38] which induce high recovery costs.

2.6 Peer-to-peer for Grid Resource Discovery

Grids are distributed systems that support resource sharing and collaboration, and operate on well-defined infrastructures, which provide services for resource discovery, resource man-

agement, monitoring, and security [116]. Resource discovery is the process of determining which grid resource is the best candidate to complete a job [257]. The discovery operation has to complete in the shortest amount of time, with an efficient use of resources, and at minimum cost [257]. In this respect, resource discovery is typically achieved by means of centralized or hierarchical information systems, although proposals for fully decentralized approaches based on the peer-to-peer paradigm exist [152, 144, 282].

Grid *versus* Peer-to-Peer In order to understand how peer-to-peer technologies could harness the deployment of future grids, a brief review of the common traits and differences between the two concepts is required. The analysis conducted in [282] highlights several points of distinction, in terms of shared resources, target users, infrastructure, scale, security, and applications. Whereas grids are characterized by a moderate number of trusted entities, peer-to-peer communities consist of a multitude of untrusted systems that are less concerned with quality of service policies and reliable service provisioning [116]. These differences also reflect the interests put in the development and operation of such systems: grids are supported by large investments and common effort from the involved parties, meanwhile peer-to-peer systems are loosely coupled platforms with little incentives for cooperation.

Concerning resources and applications, peer-to-peer systems have mostly emerged as file-sharing platforms, whereas grids typically target large scientific computing tasks. In this regard, systems participating into a grid are more powerful, persistent, and better connected than those in a peer-to-peer network, and they are managed through stricter user and access policies which contribute to a more robust and reliable operation. Partly because of larger scales, loose dependency between resources, and non-criticality of the deployed applications, peer-to-peer systems have better fault-resilience than grids. This difference is aggravated by the fact that traditional grid systems rely on centralized or hierarchical management [178, 86], which determine weak points. Peer-to-peer systems also exhibit a higher degree of participation dynamism, with shorter session times and frequent connections and disconnections, while maintaining a relatively stable set of shared resources. In contrast, hosts connected to grids are relatively stable, but the availability of shared resources greatly varies over time [275].

Convergence of grid and P2P Convergence of grid and peer-to-peer has been deemed beneficial for both platforms [282]. As grid systems scale up and integrate a large number of commodity hardware, the boundaries that separate them from peer-to-peer networks disappear; according to this vision, future grids will see centralized management replaced by fully distributed solutions, seeking to increase reliability and to avoid bottlenecks. Meanwhile, such next-generation grids, composed of a large number of nodes, will need to relax partecipation requirements concerning trust and security, and assume the flexible and self-organized behaviors required to minimize management costs. The experience acquired with peer-to-peer systems is continuosly being transferred to grids. Accordingly, we have witnessed the arousal of peer-to-peer solutions for distributed resource discovery, scheduling, and storage. Nonetheless, implementing peer-to-peer information systems necessitates a choice between structured and unstructured overlays, which is tied to the goals and requirements of the intended deployment scenario. Whereas structured overlays enable very

efficient keyword search, unstructured ones allow for complex queries and typically require less effort to manage the overlay. In the following, we review the requirements set by our evaluation scenario of a grid, and highlight the directions and design choices that could benefit the project.

2.6.1 Peer-to-Peer Grid information systems

In this section we review some of the existing grid information systems based on peer-to-peer technologies. In-depth review and comparison of different models and solutions can be found in [231, 282, 201, 202].

Structured Systems As shown in the previous section, structured solutions enable efficient and deterministic information retrieval. Unfortunately, grid resource discovery cannot easily benefit from these systems, as queries cannot be mapped to simple hashed keys. More specifically, in contrast to file indexing (typical of peer-to-peer file sharing), querying grid resources depends on the use of complex queries composed of multiple attributes whose values are often numerical ranges rather than precise values. As discussed in the preceding sections, several DHTs support range request, and solutions targeting grids have been developed. As an example, in [141], the authors propose to use a P-Grid overlay to implement a decentralized information system; in a similar way, [60] implements a resource discovery service on a CAN overlay. To support multi-attribute requests (Section 2.3.11) the combination of different independent overlays, each indexing a different attribute [24, 218, 39], has been proposed. Moreover, some examples of DHT with support for multidimensional range queries within a single overlay exist [186, 258, 281, 273, 260], but come at the expense of additional complexity in managing the network.

The system presented in [275] implements a grid information system that employs different techniques to support multi-attribute queries. A Chord-like ring that uses consistent hashing provides support for range-queries; several rings are deployed to support multiple attributes. Furthermore, flooding is used to resolve arbitrary queries, and queries concerning dynamic resources. In contrast, XenoSearch [266] models resources in a multi-dimensional space that is distributed across the nodes. Using a Pastry overlay, queries are directed toward the nodes serving the required partition of the space. A similar solution proposed in [33] divides the attribute space among nodes using a tree structure.

Unstructured systems Unstructured systems do not limit the complexity of queries, as each request is resolved locally on each node. In this regard, unstructured P2P networks are better suited for integration with existing grid middlewares, because the latter may store information using an arbitrary format rather than a fixed schema [71]. In [200], the authors propose a super-peer system matching the physical organization of nodes. Each super-peer is responsible for indexing resources shared within its administrative domain, for communicating with super-peers of other domains, as well as for managing resource discovery requests from ordinary nodes. This research also highlights the need for strategies to improve the efficiency of resource discovery. In the same direction, research in [151] discusses the implementation of an unstructured peer-to-peer information system and analyzes different query forwarding strategies.

Common issues Both structured and unstructured systems have to deal with common issues, such as decreased performance compared to centralized indexes (longer response times), and security concerns [71]. Regarding the latter, several propositions have been made to address the problem, such as [92, 102]. These drawbacks are nonetheless balanced by the increased robustness and fault tolerance of the system.

2.7 Summary

This chapter presented an overview of the current state of the art in peer-to-peer systems. The fundamental principles of structured and unstructured solutions have been detailed, and noteworthy examples of both classes of peer-to-peer infrastructures have been discussed. Moreover, in relation to the considered evaluation scenario of a grid, important aspects and issues related to robustness under churn and application in grid environments have been analyzed.

Although structured systems exhibit deterministic search performance that enables efficient key based lookups, solutions that support multi-dimensional and multi-attribute queries are more complex and might still not be enough to support rich queries that are typical in some scenarios such as grids. Moreover, a complete analysis of the behavior and robustness of such systems under high churn is rather scarce. On the contrary, unstructured systems build on simpler designs and enable real full-text queries,

The knowledge acquired leads us to a better understanding of the benefits and limits of currently available solutions, hence providing a solid base for innovation.

Chapter 3

BlåtAnt Algorithm

Contents

BlåtAnt indicates a family of novel overlay management algorithms built around a common set of rules that focus on optimizing logical connections between nodes in order to reduce traffic generated by resource discovery queries broadcasted on the network. Furthermore, the BlåtAnt algorithms provide fault-tolerant and fault resilient behavior to prevent partitioning of the overlay and ensure reliable operation even in the case of unexpected failures.

3.1 Requirements and goals

An overlay management algorithm for the considered grid scenario must fulfill several requirements in order to provide a robust communication infrastructure that enables the development and deployment of high-level services, such as resource discovery and task scheduling. Accordingly, in the following we review the desired features, and highlight the corresponding research directions.

Fault tolerant and fault resilient operation The proposed solution should be able to cope with transient unexpected communication errors or node faults, without incurring a complete breakdown of the overlay but gracefully degrading its performance. Moreover, detected failures in the overlay connectivity must be recovered in order to ensure minimal negative impact on the operation of the system. In this regard, with fault tolerant operation we intend the ability of overcoming communication problems that lead to loss of the information being transferred between nodes; conversely, a resilient behavior is required to react to problems such as node crashes and ensure that the overlay remains connected.

Support for arbitrarily complex queries The overlay must be as generic as possible, in order to support different deployment scenarios other than the grid one chosen for evaluation. Hence, the overlay should not make any assumption neither on the type of resources shared by nodes, nor on the querying mechanism or the complexity of the queries.

Avoidance of weak spots In the review presented in the previous chapter, hubs were identified as weak point in power-law topologies. In this regard, the considered approach should avoid creating scale free topologies, and aim at almost uniform node degree distribution.

Support for efficient communication Search in a peer-to-peer overlay may incur large traffic overheads. For this reason, the topology should be optimized to avoid redundant connections and unnecessary links between nodes. Additionally, to avoid long response times, the maximum distance between any pair of nodes in the overlay must be small.

Self-organized and adaptive behavior In order to reduce management complexity, the overlay must be able to autonomously adapt to changes in the conditions of the network, such as the addition of new nodes and the removal of existing ones, in order to continuously

meet the aforementioned requirements. It is nonetheless desirable that stable network conditions, without any node joining or leaving the overlay, result in a stable overlay, hence a trade-off between adaptiveness and stability must be found.

Simple, fully distributed design The overlay management algorithm must not depend on centralized control. Nodes should cooperate in a fully distributed asynchronous way, without global information. Furthermore, the complexity of the algorithm must be low, to avoid unwanted processing overhead on the nodes.

As highlighted in the review of peer-to-peer systems in Chapter 2, structured solutions generally link the overlay structure to data, and typically do not allow for efficient resolution of arbitrarily complex queries. Accordingly, in our work and for our requirements an unstructured solution is preferable. Meanwhile, we strive to optimize the overlay so as to ensure efficient communication; under these premises, BlåtAnt goals are twofold: on one side, it aims at constructing and maintaining an overlay with bounded diameter, in order to limit the maximum delay to reach any node. On the other side, the algorithm also minimizes the number of redundant connections, by breaking up cycles that are shorter than a user-defined threshold. To meet the aforementioned requirements for fault tolerant, self-organized and adaptive operation, we propose to use bio-inspired techniques; in particular, some parts of the algorithms have taken inspiration from the behavior of ant colonies, a paradigm which has led to the successful deployment of solutions for other network related problems [62].

3.2 Basic algorithm

The optimization process implemented by BlåtAnt bounds the diameter and the girth of the overlay by creating and removing logical links between nodes. More specifically, we aim at transforming an existing overlay, represented by an undirected graph G, so that for $a, b \in \mathbb{N}^*$, the diameter d_G in the resulting graph satisfies $d_G \leq b$, and the girth g_G is $a \leq g_G$. The upper bound on the diameter ensures that nodes are reachable within a known number of hops in the overlay, thus the query forwarding can be limited without leaving a large part of the network unvisited. Conversely, the lower bound on the girth prevents small cycles, and reduces the probability that a query will be forwarded to the same node multiple times through different paths. The underlying process executed by the algorithm thus consists in rewiring the network by creating and removing logical connections between nodes. In order to ensure a stable and convergent behavior, the rewiring algorithm must terminate when a graph fulfilling the aforementioned conditions is obtained.

The optimization problem faced by BlåtAnt is similar to the degree-girth problem [106], which is concerned with finding topologies with the smallest possible number of vertices given degree and girth. This issue is related to the degree-diameter problem [207, 22], which aims at determining the largest graphs of given maximum degree and given diameter. Although our research focuses on resembling goals, a discussion of the mathematical implications of our approach in the field of graph theory and combinatorics is out of the the scope of this thesis.

3.2.1 Rewiring algorithm

The rewiring algorithm is composed of two steps: one for governing the creation of new links, and one for triggering the removal of existing links. In order to set an upper bound to the diameter of the resulting network, new links may be created. As we want to bound the diameter to $d_G \leq b$, we perform the:

Step 1	Connect two nodes x and y, when their distance is greater than b, i.e. $d_G(x, y) \geq b + 1$.

When connecting x, y a cycle of length $b+2$ is created in the graph, where the distance between all pair of nodes in the cycle is $\geq \frac{b}{2}$. Conversely, to enforce a lower bound on the girth, no cycle of length $< a$ must exist. This is accomplished by:

Step 2	Any cycle of length $< a$ is broken.

To ensure a stable and convergent behavior, the algorithm must nonetheless avoid destroying cycles it creates, hence the lower bound for the girth is $a = b + 2$ (i.e. the algorithm can create cycles that have a length greater or equal to the desired girth).

Algorithm	The rewiring algorithm is defined by repeating the aforementioned steps 1, 2, until the distance between any pair of nodes is $\leq b$ and no cycle has a length $< a$.

The following theorem relates the conditions on the diameter and on the girth:

Theorem 3.2.1. *Let G be an undirected graph where the rewiring algorithm has been applied until termination for a given a, b $\mathbb{N}^*$, $a = b + 2$; d_G and g_G be the diameter, respectively the girth of G. Then $\frac{b+2}{2} \leq d_G \leq b$, and $g_G \geq b + 2$.*

Proof. The lower bound for the girth, as resulting from the algorithm, is $a = b + 2 \geq g_G$, thus $\frac{g_G}{2} \geq \frac{b+2}{2}$. If the graph contains no cycles, then its girth is infinite; otherwise $d_G \geq \frac{g_G}{2}$: if the graph is a cycle, then the results follow; otherwise, the maximum distance between nodes in the smallest cycles (the size of which corresponds to the girth) determines the lower bound for the diameter. First consider the case of a graph with at least one cycle; in this case we have:

$$\frac{b+2}{2} \leq \frac{g_G}{2} \leq d_G \leq b \tag{3.1}$$

If the graph has no cycles, $g_G = \infty$, thus $a < g_G$ $\forall a$. Furthermore, its diameter is $d_G \leq b$, otherwise connections would have been created by the algorithm resulting in at least one cycle. □

To simplify the optimization rules that will be presented in the following section, we replace $\frac{b+2}{2} = D$, $D \in \mathbb{N}^*$, in equation 3.1 to obtain:

$$d_G \leq 2D - 2 \tag{3.2}$$

for the diameter, respectively for the girth:

$$2D \leq g_G \tag{3.3}$$

The value of D is considered as the optimization parameter of our algorithm.

3.3 Topology optimization rules

We now express the results of equation 3.1 as *Connection* and *Disconnection* rules. By applying these rules a finite number of times on a connected graph G, the resulting diameter d is is bounded according to $d_G \leq 2D - 2$.

Connection Rule Let n_i and n_j be two non-connected nodes in a connected graph G, and $d_G(n_i, n_j)$ the minimal routing distance from n_i to n_j in G. A new link between n_i and n_j is created if the following condition holds:

$$d'_G(n_i, n_j) \geq 2D - 1 \tag{3.4}$$

where $d'_G(x, y)$ is defined as $min(d_G(x, y), d_G(y, x))$. The logical connection is created by adding n_i to N_j, respectively n_j to N_i.

The Connection Rule bounds the maximum distance between each pair of nodes, hence the diameter of the network, to a value less than $2D - 1$. Conversely, the Disconnection Rule is applied in order to remove links that represent redundant paths in the graph thus breaking small cycles and bounding the girth to a value $g_G \geq 2D$.

Disconnection Rule Let n_i and n_j be two connected nodes in an overlay network G, $i \neq j$. Let $G' \leftarrow G \setminus \{n_i\}$ and N_i the set of neighbors of n_i. Node n_i is disconnected from $n_j \in N_i$ if:

$$\exists\, n_k \in N_i, k \neq j, |N_j| > |N_k| \;:\; d^*_{G'}(n_j, n_k) \leq 2D - 3 \tag{3.5}$$

where $d^*_G(x, y)$ is defined as $max(d_G(x, y), d_G(y, x))$. The disconnection consists of removing n_i from N_j, respectively n_j from N_i.

Safeness of the Disconnection Rule The presented rules ensure that the diameter as well as the girth in the resulting network are bounded according to the previously described limits. Nonetheless, the optimization process may converge only when global and precise information about the overlay is available. In a fully distributed implementation, where each node must rely on partial and potentially out-of-date information about the overlay, guaranteeing proper operation is more difficult. Whereas degraded information about path distances in the overlay just increases the average path length and results in a less optimized overlay, concurrent application of the *Disconnection Rule* may lead to a partitioning of the network and thus disruption of higher-level communication. In a situation where complete knowledge of the overlay is available each cycle can only be broken once, thus the overlay cannot be partitioned. In a fully distributed scenario local information on each node may be outdated, e.g. refer to cycles that have already been broken. Hence, to ensure that the algorithm works as expected a restriction on the *Disconnection Rule* is introduced: for each considered cycle, only the node with the greatest identifier (according to some ordering known to all nodes) is allowed to perform a disconnection. That node is referred as to the *master* of a given cycle. Letting only the master node perform disconnections prevents partitioning, but nonetheless requires the master to keep track of broken cycles, a solution which has the potential drawback of requiring a large amount of storage on each

master node, especially in very dynamic networks. The solution that is adopted in our work is to only allow the master of a cycle to remove links with its own neighbors, thus making it possible to verify whether the cycle has already been broken by relying only on already available local and up-to-date information.

3.4 BlåtAnt-R

The BlåtAnt-R algorithm is a fully distributed implementation of the topology optimization rules that employs bio-inspired swarm intelligence techniques to collect and spread information across peers. The algorithm presented in this thesis is derived from the one introduced in [51]; more specifically, the underlying logic of the connection and disconnection rules has been adapted to follow the mathematical construction presented at the beginning of this chapter. BlåtAnt-R is the second fully distributed implementation of the algorithm: the first distributed implementation of BlåtAnt [49, 50] didn't support fault tolerance, and was thus not suitable for deployment in a real network. In this respect, BlåtAnt-R represents the first fully fault tolerant version of the algorithm. In contrast to a centralized approach, the decentralized implementation has to balance between precise and up-to-date information and increased network traffic. Moreover, because of the fully-distributed design, actions executed by one node may invalidate the information collected by others.

3.4.1 Swarm intelligence

Swarm intelligence is a field of artificial intelligence that mimics the behavior of swarms of insects in order to solve computationally intensive optimization problems [48] or to implement collective intelligent behaviors [36]. Concerning optimization tasks, a number of different techniques have been proposed, with the two most known being Particle Swarm Optimization (PSO) [76] and Ant Colony Optimization (ACO) [98]. While PSO is more suited for solving numerical problems, ACO naturally targets graph and network related tasks. Accordingly, in the following we focus our attention on the latter, and briefly discuss how distributed systems can benefit from ant-inspired solutions.

Ant colony optimization The ant colony optimization [98] (also known as ACO) metaheuristic is an optimization technique that replicates the behavior of ants searching for food. An example of the foraging process is depicted in Figure 3.1. Each ant starts from its nest, and randomly wanders in the environment, until a food source is found. Subsequently, the insect returns to the nest, and lays a small amount of chemical pheromone to trace the path from the nest to the food. Other individuals in the colony will sense the chemical trail and choose to follow it to reach the food: on their way back to the nest, they will actively *reinforce* the trail by depositing more pheromone. In this view, pheromone trails represent a form of indirect communication between ants, called *stigmergy*, to signal where the food is located. Ants are not forced to follow an existing trail: when an ant wanders in the environment it can choose to either *exploit* an existing path, or randomly *explore* the environment. If a trail is not reinforced, it will disappear due to the evaporation of the chemical. By default, in absence of pheromone trails, ant exploration will take place.

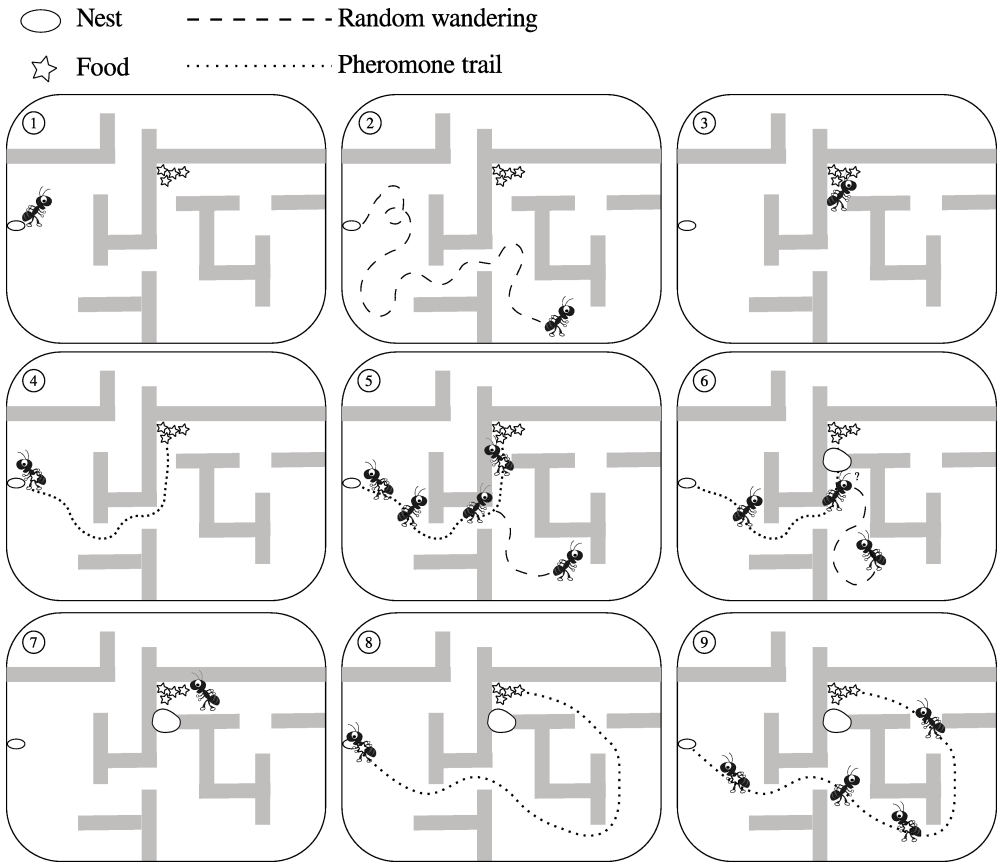


Figure 3.1: The ant foraging process

Shortest paths that lead to the food source naturally emerge from this process: because a short path takes less time to be traveled, the rate at which pheromone trails are reinforced is faster, thus the concentration levels remains higher than in other paths. Because the concentration of pheromone on the path also increases the attractiveness of it toward wandering ants, a positive feedback cycle will be created. Conversely, if a path becomes inaccessible, evaporation will render it less desirable. Accordingly, ACO exploits an emergent and adaptive behavior.

This process can be modeled by ant-like software agents and be used to find shortest paths in graphs. More specifically, in computer science, ACO has been used to solve NP-complete graph problems, such as the Traveling Salesman Problem [97]. Consequently, a number of other NP-complete problems have been solved using ACO by transforming them into an instance of TSP.

Application in Computer Networks Ant algorithms are of simple logic and inherently distributed, because neither central control, nor direct communication between agents are required. The foraging behavior of ants has been exploited for implementing adaptive routing algorithms, as shown in [170], or semantic resource discovery protocols [204]. Following the same principles, the clustering behavior of the *Messor Sancta* species of ants led to the development of fully distributed load balancing [210] or clustering [113] solutions.

In the same line of thought, we aim at exploiting some of the principles of ant colony optimization in order to simplify the implementation of a distributed version of our overlay optimization algorithm. In this respect, we deem that the fact that the ACO metaheuristic does not require direct interaction between agents can ease the development of fully distributed algorithms.

3.4.2 Distributed overlay optimization

The overlay optimization process modifies the logical connections between nodes depending on the obtained partial, transient information about the network. To collect such data, different types of mobile agents (referred to as ants) are employed. In the spirit of swarm intelligence algorithms, the outcome of the optimization process must not depend on individual agents but on the operation of a colony as a whole, that is the collaborative actions of multiple ants executing on the overlay. In the following we detail the distributed implementation of the optimization algorithm, which comprises the data structures maintained by each node, the semantic of each ant species, and the behavior executed by each node according to the perceived status of the network.

3.4.3 Local data structures

Each peer n_i maintains a set N_i of addresses of other peers representing its neighborhood. The maximum number of neighbors is m, although the algorithm itself can only create mo connections, $mo \leq m$, during normal operations: the remaining free connections are reserved for recovery procedures. To avoid the creation of large hubs, the size of the neighbor set is typically limited to small values < 10.

With the exception of the connection phase, a node n_i can only communicate with peers in its neighborhood N_i. Furthermore, it is possible to make a distinction between active and inactive neighbors. A neighbor of n_i is considered inactive until it has exchanged some information with n_i. We denote the fact that $n_j \in N_i$ is an active neighbor of n_i with $n_i \leftarrow n_j$; inversely, an inactive neighbor is denoted as $n_i \not\leftarrow n_j$. Because a node can only communicate with its neighbors, $n_i \leftarrow n_j$ implies $n_i \in N_j$.

Along with the neighbor set, each peer also keeps a fixed size cache table (α), containing information about other peers of the network. Each entry in the table has the form $\langle n_j, N_j, d_j, t_j, t_i \rangle$, where n_j is the identifier of the remote peer, N_j its neighbor set, d_j the estimated distance from n_j to n_i, t_j the time on n_j when that information was retrieved, and t_i the local time of the last entry update. The remote time t_j is used to determine if incoming information is older than the current one, whereas t_i is used to clean up old entries when the table fills up. The information found in the α table is highly volatile, and is continuously updated by ants traveling on the network. To support fault resilience, as long as $n_j \in N_i$ the entry corresponding to n_j in α_i cannot be removed: this ensures that the last known neighbors of n_j are always available and cannot be overwritten. An example of an α table is given in Figure 3.4.3.

Figure 3.2: Sample BLÅTANT-R α table

Identifier	Neighbors	Last Update	Timestamp	Distance
A	P, Q, K, L	87	101	4
U	E, F, B, W	89	85	5

3.4.4 Pheromone trails

As previously discussed, communication between real ants occurs using a stigmergic (i.e. indirect) mechanism which involves leaving chemical *pheromone* trails in the environment. These chemical traces can be sensed by other individuals in the colony and their concentration indicates the desirability of a given path. With time, unless new chemical is left by an insect, the concentration of the trail completely evaporates. Evaporation has the added benefit of seamlessly suppressing errors and overcoming bad system decisions. In our system, we emulate this phenomenon, and in that respect pheromone concentrations are represented as numerical values $\tau \in [0, 1]$ stored on each node and associated with paths to neighbors in the overlay. Ants executing on a node can both read the actual concentration of a trail, and *reinforce* it by increasing its value up to a maximum of 1.

Each node periodically simulates *evaporation* by lowering the value of a trail τ according to an update function $\tau \leftarrow \tau * \psi$, and $\psi < 1$. If the concentration on a trail falls below a threshold ε, the trail is removed. We distinguish between incoming β trails, and outgoing γ trails. When an ant travels from node n_i to a neighbor n_j, the corresponding trail $\gamma_i[n_j]$ on n_i is reinforced. Conversely, when the ant arrives on n_j, pheromone trail $\beta_j[n_i]$ is reinforced.

3.4.5 Ant species

In the considered framework, ant species describe information containers that can be exchanged between nodes and that trigger particular response behaviors, such as creating or removing overlay links. Ants can nonetheless be viewed as *living* entities that carry information, move across the overlay, and perform specific tasks proper to their species. More specifically, BLÅTANT-R defines six different species of ant agents:

- *Discovery Ants* are used to collect and spread information about the status of the network (nodes and links). Ants wander across the network and store data about each visited node n_k represented as a triple $\langle n_k, timestamp_{n_k}, N_k \rangle$ containing the node's identifier, the remote timestamp at n_k when the information was collected, and its actual neighbors N_k. This triple is appended to a bounded-size vector V of maximal length l_v. Visited nodes also receive the vector currently carried by the ant, and use this information to update the local view of the network (stored in the α table). Depending on the position of each entry in the vector, a node can thus infer an estimation of the distance of the node in the overlay. An entry in the vector corresponding to a node n_j contains the following information:
  - n_j: identifier of the visited node;
  - N_j: set of neighbors of n_j;

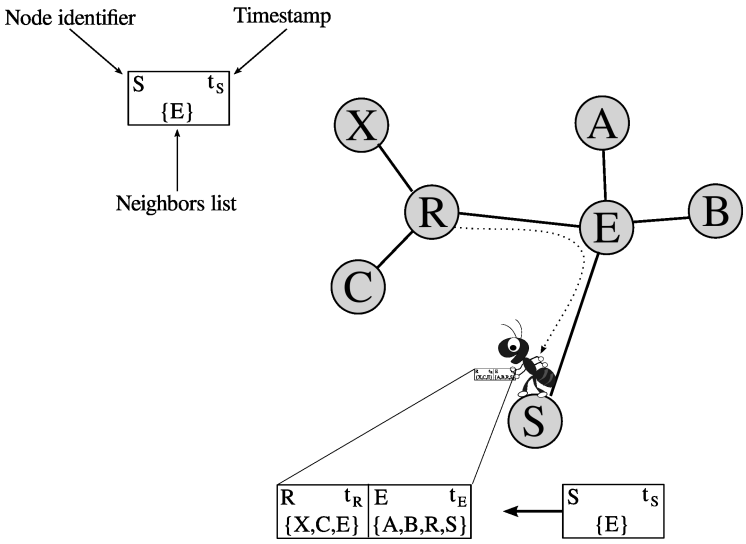


Figure 3.3: BLÅTANT-R Discovery Ant wandering

– $timestamp_j$: timestamp on n_j when the information was collected.

Discovery Ants wander on the overlay following existing links between nodes. At each step, an ant may choose to either proceed on a random path (*exploration*) to one of nodes in the local neighborhood set with probability κ, or select a path depending on its actual pheromone concentration (*exploitation*) with probability $1 - \kappa$. More specifically, paths with lower γ pheromone concentration are preferably chosen, ensuring a fair coverage of the network. Visited nodes in V are avoided.

Discovery Ants are responsible for continuously monitoring the state of the network, and have a limited lifespan π (maximum number of wandering steps). As *Discovery Ants* may get lost due to node crashes, at regular intervals ι a new individual is generated on every node with probability μ, ensuring the survival of the population. As the optimization task depends on the information gathered by *Discovery Ants*, running the algorithm with an empty population will prevent any improvement of the topology.

An example of the behavior of a *Discovery Ant* is shown in Figure 3.3. The ant is created and executes initially on node R, then moves to E and finally to S. Accordingly, the information passed to each of the nodes is as follows:

– R receives ();
– E receives $(\langle R, timestamp_R, \{X, C, E\}\rangle)$;
– S receives $(\langle R, timestamp_R, \{X, C, E\}\rangle, \langle E, timestamp_E, \{A, B, R, S\}\rangle)$.

The ant finally collects information on node S and continues its wandering.

– *Construction-Link Ants* are sent by nodes wanting to join the network, but also during recovery procedures. A node can either accept the connection, or forward it to

one of its neighbors (randomly chosen amongst the ones with the smallest degree). Forwarding is required if the node has already reached the maximum number of allowed neighbors. To avoid long connection delays, each ant can only travel a maximum number of steps $clant_{ttl}$: when the limit is reached the connection procedure must be completed by the first visited node with a free slot. When node n_i accepts a *Construction-Link Ant* sent by n_j, it adds n_j to N_i and then sends the ant back to n_j, where n_i is added to N_j.

- *Optimization-Link Ants* are instantiated by peers in order to optimize the diameter of the network. When a node n_i wants to create a connection with n_j it sends an ant to it. At n_j the ant checks the estimated distance to n_i (in the α_j table). If the estimated distance is $> 2D - 1$, or no information is found in α_j, the connection procedure can continue. In this case, n_i is added to N_j, and the ant migrates back to n_i, where n_j is finally added to N_i.

- *Unlink Ants* remove the links as result of the application of the disconnection rule or when nodes leave the overlay. When a node n_i wants to disconnect $n_j \in N_i$, it first removes n_j from N_i, and then sends an *Unlink Ant* to n_j in order to remove n_i from N_j.

- *Update Neighbors Ants* notify a node when one of its neighbors has changed its neighbors set. This ensures that each node is able to recover from abrupt disconnection of a neighbor by connecting with its last known neighbors. *Update Neighbors Ants* carry the list of the neighbors N_i from the source node n_i, and update the entry corresponding to n_i in the α_j table of each target neighbor $n_j \in N_i$.

- *Ping Ants* are used to keep connections between nodes alive by reinforcing pheromone trails on visited nodes. Abrupt node disconnection can be detected by monitoring the concentration of β trails: when values approach a lower threshold a recovery procedure is started. If application traffic is low, the trail between two nodes may not be frequently reinforced, and thus completely evaporate even though the corresponding nodes are still connected to the overlay. To prevent this from happening, *Ping Ants* are periodically deployed as soon as trail concentration falls below a certain threshold.

3.4.6 Fault resilience

Disconnecting from the overlay can occur either properly or improperly. Proper disconnections require the leaving node to inform all of its neighbors and initiate a recovery procedure to ensure connectivity is preserved. This procedure involves sending out *Construction-Link Ants* to all neighbors and connecting them using a ring topology (Figure 3.4). Improper or abrupt disconnections occur when a node stops communicating with its neighbors, either because of a crash or because of network issues. In this situation, each neighbor starts the recovery procedure on its own as soon as the failure is detected (by means of monitoring the concentration of β pheromone).

Proper disconnection: Leaving procedure When a peer wants to quit the network, it must ensure that all of its neighbors remain connected. When node n_i leaves the network, it first sends an *Unlink Ant* to all of its neighbors. Next, it sends a *Construction-Link Ant* to all its neighbors in order to create a ring connecting all of them. Figures 3.4a) and 3.4b) depict an example topology before, respectively after the departure of node n_i.

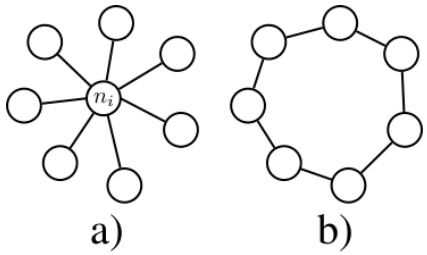


Figure 3.4: Leaving procedure

Improper disconnection (crash): Recovery procedure The recovery procedure is used to prevent network partitioning in the event of a node crash. When a node n_j detects the departure of one of its neighbors n_i by sensing the complete evaporation of its β pheromone trail, it may start a recovery procedure. The exact behavior of the node depends on whether $n_j \not\leftarrow n_i$ or $n_j \leftarrow n_i$.

- if $n_j \not\leftarrow n_i$ no information was ever received from this connection. This situation can either happen when a node leaves just after being connected, or when a connection procedure is interrupted. In such cases, n_i is just removed from N_j.
- if $n_j \leftarrow n_i$ some data was already successfully exchanged through this connection. It is thus necessary to ensure that connectivity of the network is preserved by executing the recovery procedure. This procedure involves removing n_i from N_j and subsequently send *Construction-Link Ants* to all last known neighbors of n_i in order to construct a ring topology as in Figure 3.4. In contrast to a proper disconnection, the recovery procedure is started by all neighbors of n_i, as soon as the failure has been detected: although this can increase network overhead (proportionally to the size of the neighborhood set), each neighbor must initiate the recovery process because it cannot assume that other did or would do it.

3.4.7 Optimization rules evaluation

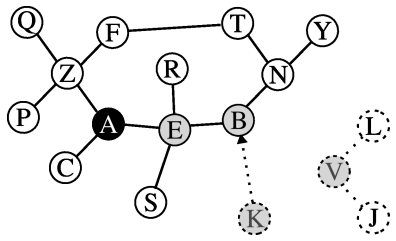
During its lifetime, each node n_i receives information from *Discovery Ants*, and correspondingly updates its local α_i table. Each triple in the ant vector V updates the corresponding entry in the α_i table; if no such entry exists, a new one is created. When the table reaches its maximum capacity, as well as after a certain amount of time, the least recently updated entries are replaced. To solve conflicts when receiving concurrent information about the same node, the remote timestamp in the table and in V are used. At regular intervals ω the contents of the table and the neighbor set are used to construct a partial graph of the network and evaluate the *Disconnection Rule* and the *Connection Rule*. All disconnected

components that cannot be reached from n_i, as well as components connected by means of non-bidirectional paths are removed from the graph.

Evaluating disconnections For disconnections, node n_i computes the shortest paths not traversing n_i between each pair of neighbors $n_j, n_k \in N_i$. The shortest path is then selected, and if its length is less than $2D-2$, n_i initiates a disconnection (by means of an *Unlink Ant*) from either n_j or n_k: in particular, the neighbor with the highest degree is disconnected, in order to promote a more balanced link distribution.

Evaluating connections To evaluate new connections, node n_i determines the distance to all nodes $n_z \notin N_i$, and initiates a connection procedure with the farthest node (by sending an *Optimization-Link Ant*) if its distance is $\geq 2D-1$. Nodes that are being connected are marked, so that subsequent rule evaluation will ignore them. Furthermore, all computed distances are used to update the corresponding distance field in the α table.

Example Figure 3.5 illustrates an example of the rules evaluation procedure: the contents of the α table for node A, the neighbor set N_A, as well as the corresponding partial graph are shown. Node K is removed from the partial graph because there is no edge from B to K; conversely, nodes J, L, V are removed because they belong to a disconnected component. To evaluate the disconnection rule the distance between C, E, Z along paths that do not traverse A is computed. Accordingly, a path of length 5 hops connects nodes E and Z, and depending on the value of D either one of the nodes along this path could be disconnected. Conversely, for evaluating the connection rule, the distance to nodes that are not within the neighbor set is computed. The obtained value is used to update the estimated distance field in the α table (for the entry corresponding to the considered node), and eventually triggers a connection procedure.



$N_A = \{C, E, Z\}$

α_A (timestamps omitted)

identifier	neighbors	distance
Z	A,F,P,Q	1
E	A,R,S,B	1
B	E,N	2
F	Z,T	2
T	F,N	3
K	B	3
N	B,T,Y	3
V	L,J	3

Figure 3.5: BLÅTANT-R Rules Evaluation

3.5 BLÅTANT-S

BLÅTANT-S is the third implementation of the algorithm that focuses on simplicity by reducing both the computational complexity and the amount of information exchanged

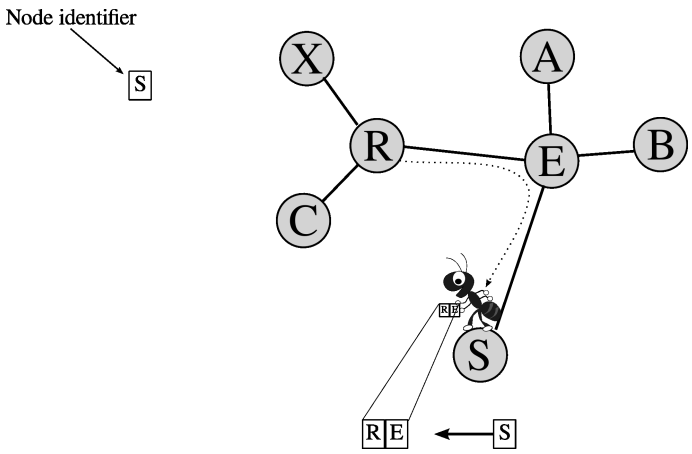


Figure 3.6: BLÅTANT-S Discovery-S Ant wandering

between nodes. It is noteworthy to say that the solution proposed by this version is not meant to replace the $-R$ one, but it is to be considered as an alternative implementation based on the same optimization rules. In this section the differences between BLÅTANT-S and BLÅTANT-R will be highlighted.

3.5.1 Ant species

The only difference of the $-S$ version compared to the $-R$ one is in the amount of information collected and carried by *Discovery Ants*. In particular, the latter carry a bounded-length vector V that contains only the identifier of the nodes visited by the ant. In contrast to the $-R$ version, no other information about the neighbors is collected, thus the traffic generated by *Discovery Ants* is considerably lower. In the following, *Discovery Ants* used by the -S version of the algorithm will be referred to as *Discovery-S Ants*. The example depicted in Figure 3.6 shows an ant traveling from R to S, passing through E. The information passed to each of the nodes is as follows:

- R receives ();
- E receives ($\langle R \rangle$);
- S receives ($\langle R \rangle, \langle E \rangle$).

The ant then adds the identifier of node S at the end of its vector, and continues its wandering around the network.

3.5.2 Optimization rules evaluation

The evaluation process is performed each time a *Discovery Ant-S* visits a node and uploads its information vector. Distance estimations are computed on the base of the relative distance between identifiers in the vector, rather than through the construction of a graph. This estimation is used both for evaluating the optimization rules, as well as to update the corresponding field in the α table. The complexity of evaluating the optimization rules thus

reduces from $O(n^2)$ in BlåtAnt-R to $O(n)$ in BlåtAnt-S. On the downside, distance estimations are less precise and more errors are inevitable.

Evaluating disconnections Nodes first identify all pairs of neighbor's identifiers within the information vector V, and then compute the absolute value of the difference between the positions of the identifiers of each pair in the vector. If the identifier of the node appears between the pair of neighbors, their distance is set to ∞. The disconnection rule is then applied on the pair of neighbors with the smallest in-between distance in the vector.

Evaluating connections A node n_i evaluates the distance to any other node $n_k \notin N_i$ by calculating the absolute value of the difference between the position of the target's node identifier and either the closest neighbor $n_j \in N_i$ identifier, or the tail of the vector, whichever value is smaller. This evaluation is stored in the α table. Periodic evaluation of the connection rule occurs at intervals ω, and might subsequently trigger the creation of a new link. To prevent blatant errors, the list of neighbors of neighbors stored in the table is used to adjust distances if necessary: more specifically, if a node appears to be connected to a neighbor, its distance is automatically set equal to 2.

Example Figure 3.7 illustrates an example of the rules evaluation procedure triggered by the reception of an information vector on node F. We suppose that the neighbors of node F are A and M, and the vector carried by the incoming *Discovery Ant-S* contains $\{U, M, O, S, E, W, A\}$, where A is the last node visited by the ant prior to F. To evaluate disconnections, the distance between neighbors A and M in the vector is considered: in the example, their distance is 5. Conversely, for connections, the distances between nodes W, E, S, O and either A, M or the end of the vector are computed. Accordingly, the distances are of 2 and 3 hops, for nodes W, O, U, respectively E, S.

$N_F = \{A, M\}$

Discovery Ant Vector

U	M	O	S	E	W	A

Figure 3.7: BlåtAnt-S Rules Evaluation on node F

3.6 Evaluation

By means of extensive experimentation of BlåtAnt-R and BlåtAnt-S, we aim at evaluating their behavior along different axes. More specifically, both the properties of the resulting networks as well as the robustness of the algorithm need to be assessed. Accordingly, in this section we present the considered measurements and the corresponding test scenarios, a summary of which is provided in Table 3.1.

Important measurements for evaluating the optimization process are the diameter, the average path length, the number of (directed) edges in the overlay, the degree distribution, and the number of cycles as well as their length. While the diameter and average path length assess the capacity of the algorithm to bound the maximum distance between any pair of peers, values concerning cycles quantify the girth of the graph and the amount

of redundant paths on the overlay. In addition, we deem the degree distribution and the number of edges useful for determining the presence of hubs, respectively the complexity of the resulting network.

BlåtAnt-R and BlåtAnt-S are compared to highlight their benefits and drawbacks; in particular, we aim at understanding if the increased complexity of the -R version provides advantages over the simpler distance estimation logic implemented by the -S version.

Scenario	Focus of the evaluation
A	Convergence in a stable overlay
B	Adaptiveness
C	Scalability
D	Overlay fault resilience
E	Communication fault tolerance
F	Sensitivity
G	Comparison with Newscast and Gnutella

Table 3.1: Summary of overlay evaluation scenarios

3.6.1 Simulation setup

All evaluation scenarios are executed on a custom discrete-time simulator with a resolution of 50 ms that enables accurate measurements of the aforementioned variables. Communication delays between peers are determined by an underlying topology of 3037 nodes and 4788 links created with i-net 3.0 [295]. The average path length is of 3 hops, and the average latency on each link is 78 ms. The topology is depicted in Figure 3.8.

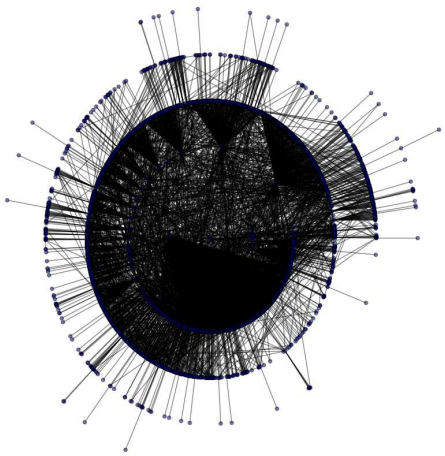


Figure 3.8: Underlying topology

Unless otherwise specified, in all scenarios an overlay of 1281 nodes is constructed; 10 nodes out of 1281 constitute the *well-known* connection points of the overlay, which are linked together with random connections. At the beginning of the simulation, the remaining 1271 nodes initiate a connection procedure by sending a *Construction-Link Ant* to one of the *well-known* peers (chosen uniformly at random). Unless otherwise specified, in dynamic

scenarios the number of nodes varies during execution, as nodes joining and leaving the overlay are simulated in the interval between 6 hours and 9 hours into simulations, with a rate of one node added and one removed every 4 seconds. In this regard, two disconnection strategies have been considered, proper and improper, with the probability of an improper disconnection being 50%. The reference parameter values used during all simulations, unless otherwise stated, are detailed in Table 3.2. According to the rewiring algorithm performed by BlåtAnt, the choice of the optimization parameter $D = 5$ aims at obtaining a diameter $\leq 2D - 2 = 8$ and a girth $\geq 2D = 10$.

A sensitivity analysis of some of these values is detailed in the following. To obtain statistically representative data, 5 simulation runs for each scenario are performed. Each run simulates an execution of 12 hours, which includes the time required to initially setup the overlay. A baseline for comparison for all considered values is represented in a reference scenario, that is used in our sensitivity analysis in scenarios of set **F**.

	Value	Description
D	5	Optimization parameter
$\lceil \lvert\alpha\rvert \rceil$	28	Maximum number of entries in the α table of each node
$\lceil \alpha_{age} \rceil$	300	Maximum age for valid entries in the α table (in seconds); entries older than this value are removed from the table
m	8	Maximum node degree (number of neighbors)
mo	6	Maximum number of allowed connections created by *Optimization-Link Ants*
ι	100	*Discovery Ants* respawn interval (in seconds)
π	25	Maximum number of hops that *Discovery Ants* can travel in the network
μ	5%	*Discovery Ants* respawn probability
l_v	15	*Discovery Ants* information vector maximum length
ε	0.02	Minimum pheromone concentration (for both β and γ)
ψ_γ	$0.02^{\frac{1}{300}}$	γ pheromone decay (applied every $100ms$, corresponds to a complete evaporation in 30 seconds)
ψ_β	$0.02^{\frac{1}{600}}$	β pheromone decay (applied every $100ms$, corresponds to a complete evaporation in 60 seconds)
κ	50%	*Discovery Ants* exploration probability
ω	1	Rules evaluation period (in seconds)
$clant_{ttl}$	10	Maximum number of hops for *Construction-Link Ants*

Table 3.2: Summary of overlay evaluation parameters

To provide a simple baseline for comparison with different overlay management algorithms, simulations in scenario **G** experiment with topologies constructed using the Newscast [158] epidemic algorithm and a Gnutella algorithm. Concerning Newscast experiments, each node maintains a cache table of 20 entries, which is merged with other peers every 10 minutes on average, whereas in Gnutella runs each node maintains at most 10 neighbors (an exception is made for *well-known nodes*), and ping messages are

forwarded every 30 seconds at a distance of 7 hops in the overlay to at most 4 neighbors at each step. Dynamic network characteristics are simulated as in BLÅTANT dynamic scenarios, with nodes joining and leaving the overlay in the interval between 6 hours and 9 hours in to the simulations, with a rate of one node joining and one leaving every 4 seconds. The size of the cache in NEWSCAST is derived from the experiments detailed in [286], and the merge frequency has been chosen in order to obtain a sufficiently stable overlay without compromising its fault resiliency.

3.6.2 Traffic estimation

Besides the qualities of the resulting network, another important measure is the traffic generated by the algorithm. By determining the amount of ant agents employed by the algorithm, the overall consumed bandwidth can be determined. Concretely, the communication cost for transferring each species of ant between two nodes in the overlay has been estimated as follows:

- *Discovery Ant*: 388 bits *plus* 144 bits/visited node in BLÅTANT-S; 388 bytes *plus* 176 bits/visited node *plus* 144 bytes per each neighbor of a visited node in BLÅTANT-R;
- *Construction-link Ant*: 532 bits;
- *Optimization-link Ant*: 532 bits;
- *Unlink Ant*: 532 bits;
- *Update Neighbors Ant*: 532 bits *plus* 144 bits/neighbor;
- *Ping Ant*: 532 bits.

These estimations are based on the actual information carried by each ant, and include both the size of an IPv6 header (320 bit), a UDP header (64 bit), as well as a 4 bits packet type identifier and 144 bits source identifier (128 bits IPv6 identifier *plus* 16 bits port number). For visited nodes, the assumed 144 and 176 bits comprise 128 bits for the IPv6 address, 16 bits for the port number, and 32 bits (BLÅTANT-R ONLY) for the remote timestamp. It is noteworthy to say that the provided results refer only to the case where no application traffic is produced, thus the number of *Ping Ants* might be higher than in real situations and thus represents an upper bound rather than a typical value. Low-level network pings account for 224 bits.

In NEWSCAST simulations, the size of each exchanged entry in cache tables is estimated at 176 bits, comprising of 128 bits for the address, 16 bits for the port number and 32 bits for the timestamp. The base cost of a merge operation is assumed to be 532 bits per transmission (each merge involves 2 transmissions) *plus* the cost of transmitting each entry.

In GNUTELLA simulations the cost of a ping message is 184 bits (the size of the message header according to protocol version 0.4 [3]), whereas a pong message weights 296 bits (of which 184 bits concern the protocol header). For simplicity, we assume here that messages are transmitted using UDP instead of TCP, thus an overhead of 384 bits per packet is considered. Accordingly, for connections, the overall amount of the data exchanged is assumed to be 280 bits, and two packets are used.

3.6.3 Scenario details

In the following we present the details of each evaluation scenario, and highlight the parameter values that have been considered for both a performance and a sensitivity analysis. The corresponding results are presented in Section 3.7.

A - Convergence of the optimization process To evaluate the fully distributed optimization process implemented by BlåtAnt-R and BlåtAnt-S, a set of experiments with reliable communication was considered. Once constructed, the 1281 nodes composing the network are not modified, allowing the execution of the algorithm in a static situation. According to the value of optimization parameter $D = 5$, successful optimization of the overlay should lead to a diameter ≤ 8, and a girth ≥ 10. These experiments also provide a useful insight on the the disadvantages incurred running the optimization process locally on each node without global and reliable information, and highlight the differences between the two versions of the algorithm.

B - Adaptiveness In the previous sections, the ability of the algorithm to adapt to different network situations has emerged as an important feature. Accordingly, we examined the behavior of both versions of the algorithm in different dynamic scenarios, with the goal of determining the reactivity of the system to changes in the overlay such as the addition or removal of nodes. Experiments in set **B** simulate a dynamic network where new nodes connect to the overlay and subsequently disconnect from it. The disconnections happen cleanly, with nodes quitting the network ensuring proper connectivity by executing the leaving procedure. In contrast to other scenarios, during simulations, nodes are added and removed in the period between 30 minutes and 6 hours into simulation, according to a Poisson process with an average rate of one connection and one disconnection every 4 seconds.

C - Scalability In the same spirit as in the previous scenario, scalability experiments in scenario **C** aim at assessing the response of the algorithm in a growing network, where new nodes periodically join-in at a rate of a node added every 2 seconds, from 30 minutes up until 6 hours into simulation. Nodes send their connection request (using a *Construction-Link Ant*) to one of the *well-known* nodes. The final size of the network, after the expansion phase, is of 10620 nodes.

D - Overlay fault resilience The fault resilience of BlåtAnt overlays is evaluated in experiments of scenario **D**, with nodes joining the network and nodes leaving it improperly without informing neighbors, hence simulating a crash or an unexpected failure. In this case, we expect a recovery procedure to be initiated by nearby nodes. As with scenarios in **B**, during the dynamic part of the simulation nodes are also added to the system. More specifically, additions and removals are performed between 30 minutes and 6 hours into simulation, according to a Poisson process with an average rate of one connection and one improper disconnection every 4 seconds.

Furthermore, high-churn was simulated to determine the robustness of the overlay in the event of sudden disconnection of a large portion of the nodes. To evaluate such high-

churn situations, random sets of nodes are removed from the overlay at 60 minutes into simulation, and the size of the largest connected component in the overlay is measured; more precisely, we consider the concurrent removal of $50, 100, 250, 500, 750$ and 1000 nodes selected uniformly at random out of the initial 1281. All nodes are disconnected without performing a proper leaving procedure, thus connectivity has to be ensured by recovery procedures started by surrounding nodes.

E - Communication fault tolerance An important aspect of a fully distributed algorithm, is its ability to avoid disastrous consequences in the presence of minor communication errors. In this regard, in scenario **E** the fault tolerance characteristics of *BlåtAnt* are determined by simulating packet delay and packet loss. Specifically, each ant has a 2% chance of getting lost migration, and 20% chance of being delayed by $2500ms$ (thus preventing FIFO communication between nodes). Concerning the dynamics of the network, this set of experiments assess the performance of *BlåtAnt* in comparison to scenarios **A**, **B**, and **D** (namely stable network conditions, dynamic network conditions with proper disconnections, and dynamic network conditions with inproper disconnections).

F - Sensitivity A number of parameters influence the behavior of the algorithm and its performance. Hence, it is important to understand how each value modifies the outcome of the optimization process, the robustness of the overlay against failures, and the consumed bandwidth. The baseline for this comparison is a dynamic network scenario as described in section 3.6.1. From this perspective, scenario **F** experiments with different values for each important parameter, more specifically:

- **F0 - Baseline scenario**: the baseline for comparison is determined by a dynamic scenario where the default parameter values, as described in the previous sections, are used. In particular, the optimization parameters D is set to 5, hence the expected upper bound for the diameter is $2D - 2 = 8$, whereas the lower bound for the girth is $2D = 10$. The birth probability for *Discover Ants* is 5%, and each ant carries a vector of at most 15 entries, for at most 25 hops in the overlay. At each wandering step, *Discover Ants* have a 50% of probability of following a random path instead of the one associated with the lowest γ pheromone concentration. Finally, each node is allowed to create at most 6 connections with other nodes by means *Optimization-Link Ants*, out of a total of 8 connections, and the α table on each node is allowed to contain a maximum of 28 entries.
- **F1 - Optimization parameter** D: in this scenario we focus on how the optimization parameter D affects the characteristics of the resulting overlay, by experimenting with different values: $3, 4, 6$ and 7 (compared to $D = 5$ being the default value used in the reference scenario and throughout the rest of the simulation scenarios). The expected upper bounds for the diameter, $2D - 2$, are thus $4, 6, 8$ and 12; conversely, the lower bounds for the girth, $2D$, are $6, 8, 12$ and 14.
- **F2 - *Discovery Ant* birth probability** μ: the goal is to understand how the number of *Discover Ants*, thus the amount of information exchanged by nodes, affects the convergence rate and the quality of the overlay. In this regard, the probability

μ of each node generating a *Discovery Ant* at intervals of $\iota = 300$ seconds is varied from 5% (the default value used in all other scenarios), to 7.5%, 1%, and 0%. In the latter case, the optimization process is actually disabled, as it depends on the information provided by *Discovery Ants*.

- **F3 - Maximum length l_v of the information vector**: these experiments aim at understanding the influence of the amount of information carried by *Discovery Ants* on the optimization process. Herein, the upper bound for the length of the information vector l_v is chosen to be either 12, or 17 (15 being the default value). Eventual benefits of additional information are to be assessed in the light of the increased generated traffic.

- **F4 - Maximum number of allowed *Discovery Ant* hops** π: conversely to previous experiments, we assess here the influence of the maximum number of hops π that *Discovery Ants* are allowed to travel in the overlay before being discarded. The considered values are 15, 25 (the default), 50, and 100.

- **F5 - Maximum per-node degree** *mo*: to determine how much the optimization process depends on the constraint on the maximum per-node degree, in these experiments the value of *mo* is chosen as either 4 or 8, compared to 6 in the reference experiments. The general maximum node degree m is left equal to 8.

- **F6 - Exploration *versus* Exploitation**: the tradeoff between exploration and exploitation is assessed by varying the probability κ of a *Discovery Ant* following a random path, which is changed from the default value of 50% to 0%, 25%, 75%, and 100%.

- **F7 - Size of the α table**: we assess the influence of the amount of information stored by each node in its local α table by varying its size from the default 28 entries, down to 20 and up to 36.

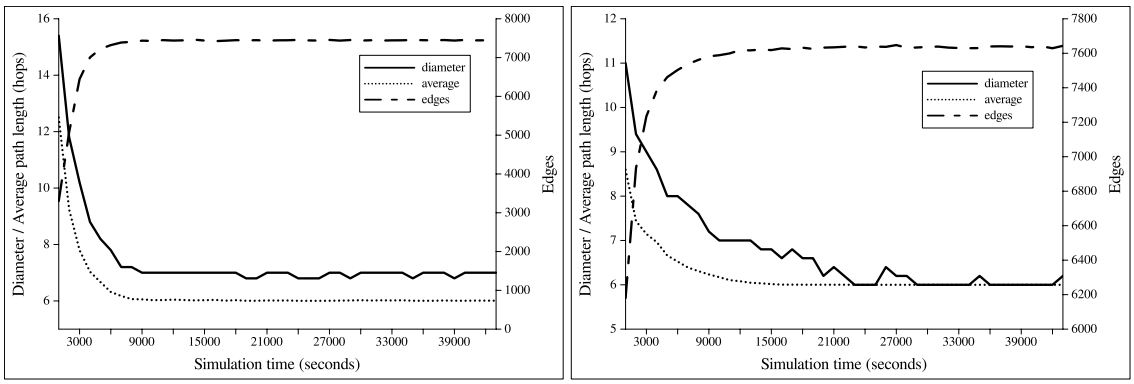
G - Comparison with Newscast and Gnutella To understand how BlåtAnt compares with existing unstructured overlay management algorithms, scenario **G** details the behavior of Newscast and Gnutella in the same network conditions as the reference evaluation in scenario **F0**. More specifically, different measurements will be compared, as for example the average path length, the amount and type of cycles in the graph, and the consumed bandwidth.

3.7 Results

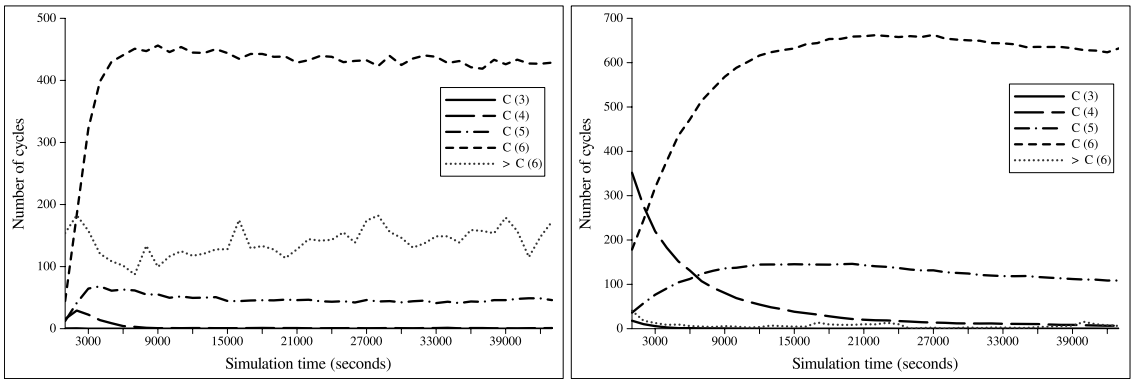
In this section the results obtained by both versions of the algorithm in the aforementioned scenarios are presented and discussed. The qualities of the algorithms are analyzed with respect to the requirements and goals defined at the beginning of this chapter. Concerning the edge count, results refer to the number of out-links, determined by the size of the neighborhood set of each node.

3.7.1 A - Convergence of the optimization process

BlåtAnt-R BlåtAnt-S



(a) Diameter, average path length, edge count



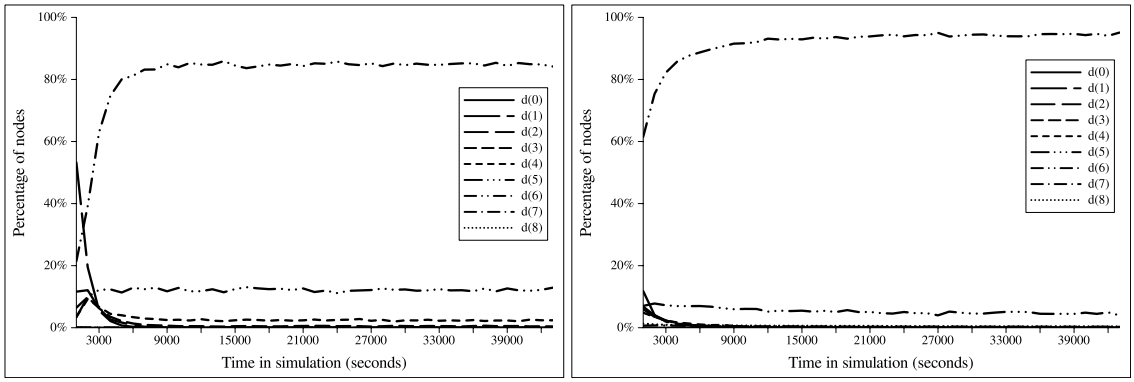
(b) Graph cycles

Figure 3.9: A - Convergence of the optimization process (diameter and cycles)

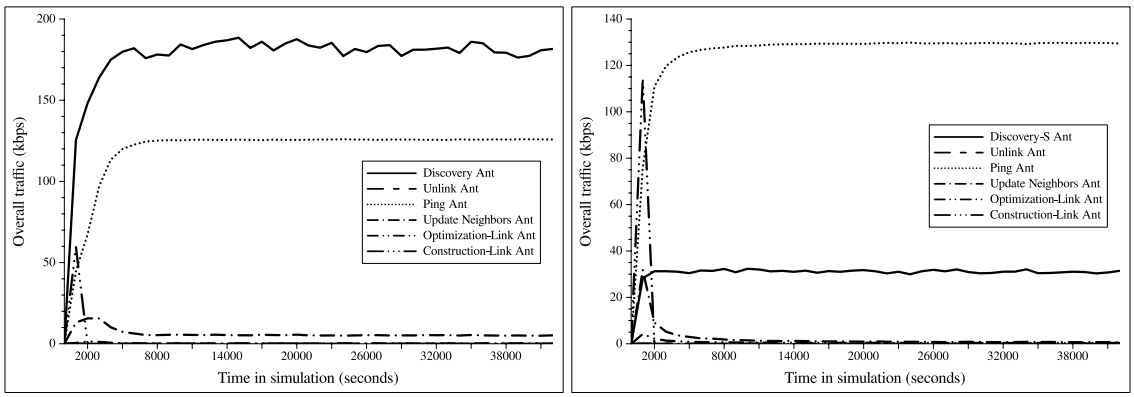
The results shown in Figure 3.9 demonstrate the convergence of the diameter (a) and the number and length of cycles (b) in both the BlåtAnt-R and the BlåtAnt-S simulations. With both algorithms, the average path length converges toward 6, well under the bound of $2D - 2 = 8$; with the -S version this convergence is slower, and the resulting graph exhibits a slightly higher number of edges, namely an average of 7643 links at the end of the simulation for BlåtAnt-S versus 7438 for BlåtAnt-R. This result can be attributed to the lower accuracy of algorithm -S, which has more difficulty finding correct distance estimations. The inaccuracies of the -S version are even more evident when the lengths of cycles are compared: with the -R version, the number of large cycles (of length larger than 6) is significant, whereas with the -S version such cycles are almost non-existent. From this point of view, neither algorithm is able to fulfill the optimization goal of a girth ≥ 10; it is nonetheless noteworthy to mention that both versions are able to limit the number of small cycles (of length up to 4), and maintain an average clustering coefficient close or equal to 0 in all simulation runs. A comparison between the rate of the emergence of large cycles and the rate of convergence of the average path length also hints at a slight relation between

the two: as small cycles are broken and only the larger ones are left, *Discovery Ants* are less likely to wander on redundant paths. A positive feedback cycle is thus created, and benefits more precise distance estimations which consequently lead to better decisions in the optimization process.

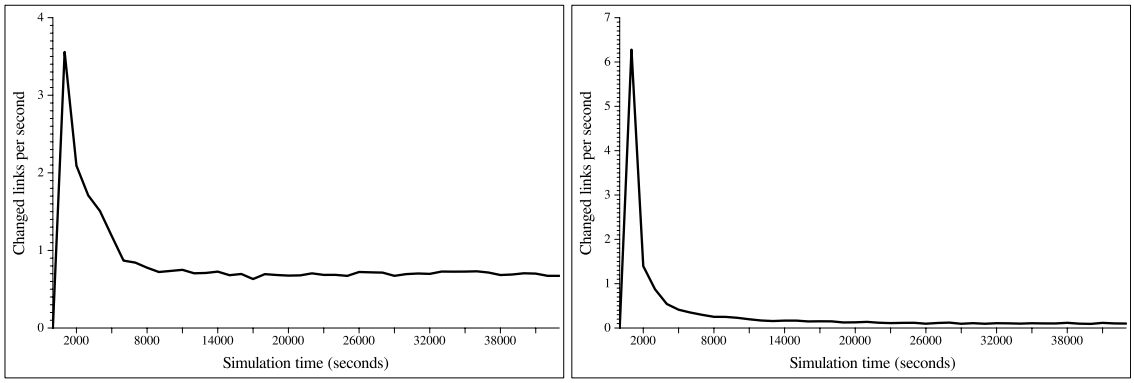
BLÅTANT-R BLÅTANT-S



(a) Degree distribution



(b) Network traffic



(c) Network stability

Figure 3.10: A - Convergence of the optimization process (degree, traffic, and stability)

Figure 3.10 (a) (b) depicts the evolution of the degree distribution and the bandwidth consumed by both algorithms. Concerning the degree, the limit of 6 neighbors per node is reached by a very large fraction of the nodes ($> 80\%$) in both algorithms, indicating that the upper bound *mo* is an important parameter for limiting the number of edges in the resulting overlay. BLÅTANT-S nonetheless shows a slightly worse performance, with an average of 95% of the nodes having a degree of 6 or higher, compared to 85% of the nodes in BLÅTANT-R. The small number of nodes with a degree higher than 6 can be attributed to well-known nodes that are subject to a higher number of incoming connection requests. Concerning the network cost, as expected algorithm -S generates less traffic than -R because of the small amount of information carried by *Discovery Ants*. More precisely, the former algorithm consumes 160 kbps on average, whereas the latter consumes 310 kbps.

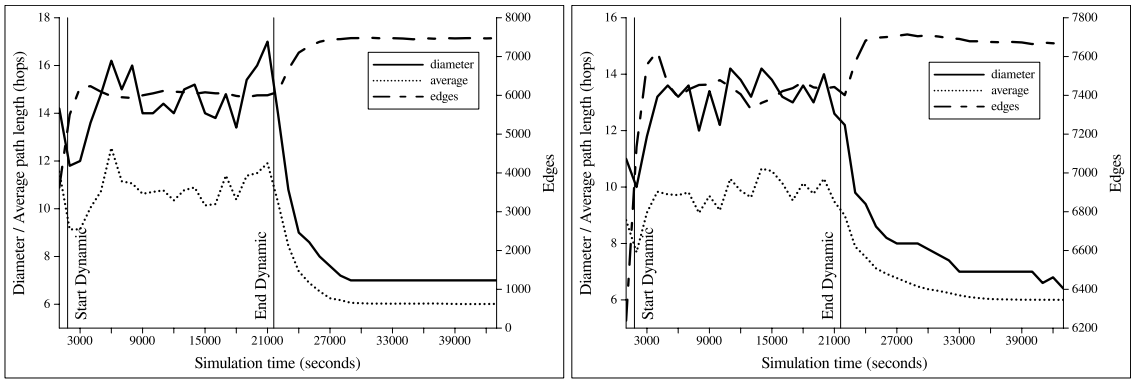
The final measure we take into account concerns the stability of the algorithm in terms of the average number of links that are changed every second in the overlay. Surprisingly, as shown in Figure 3.10 (c), the increased accuracy in the distance evaluation process of BLÅTANT-R does not seem to lead to an increase in stability: whereas the -S version modifies an average of 0.20 links per second (after the initialization phase), the -R version modifies 0.79 links/second. The reason for the unexpected more stable behavior of the -S version is due to the larger number of edges, which limits the possibility of creating a large number of new links when the network becomes saturated.

3.7.2 B - Adaptiveness

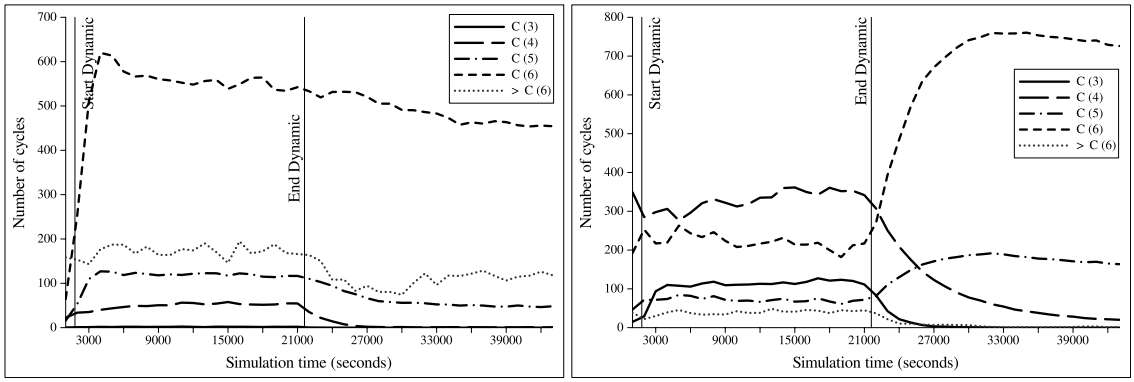
Both algorithms are able to control dynamic situations, with nodes connecting and disconnecting at 4 seconds intervals, by maintaining a diameter value slightly higher than the upper bound ($2D - 2 = 8$), as shown in Figure 3.11 (a) (vertical lines are used to mark the start and end of the dynamic part of the simulation). However, the behavior related to the lower bound of the girth in the dynamic phase of the simulation is noticeably different: with the -S version the number of large cycles is significantly reduced, while with the -R many cycles of length greater or equal to 6 are present. These results further highlight the benefits of a more accurate distance evaluation in detecting and removing small cycles, and therefore redundant paths in the overlay.

With regards to the data exchanged by nodes during the simulations, detailed in Figure 3.11 (c), it is possible to note the contrasting behavior of the -R and -S versions: while in the former the consumed bandwidth decreases during the dynamic phase (from an average of 310 kpbs, as observed in scenario **A**, to 285 kbps), in the latter it increases substantially (from 160 kbps to 220 kbps). With the -R version, the traffic reduction can be attributed to the diminished number of *Discovery Ants*, as well as *Ping Ants*, that are lost on nodes disconnecting from the network. This reduction is significant, and compensates for the increase of the *Optimization-Link Ant*, *Unlink Ant*, and *Update Neighbors Ant* populations. On the contrary, with the -S version the benefits of a reduced population of *Discovery Ants* are less evident, thus the traffic is heavily influenced by the additional required species that are instanced to connect new nodes and ensure connectivity across the overlay.

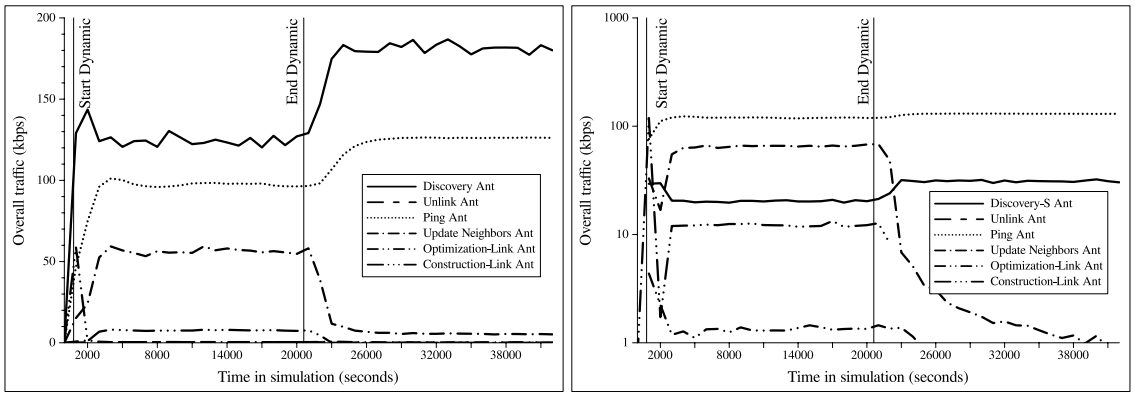
BlåtAnt-R BlåtAnt-S



(a) Diameter, average path length, edge count



(b) Graph cycles



(c) Traffic

Figure 3.11: B - Adaptiveness (diameter, cycles and traffic)

3.7.3 C - Scalability

As shown in Figure 3.12 (a), both the -R and the -S version of the algorithm are able to accommodate the additional nodes that connect to the overlay during the expansion period

(from 30 minutes until 6 hours into simulation) that made up the final overlay composed of 10620 nodes as well as 64555 and 65673 edges in the -R and -S version respectively. The diameter and average path length in the -S algorithm are more unstable than in the -R during the expanding phase of the simulation; nonetheless, both algorithms maintain the average path length closer to the user defined target $2D - 2 = 8$. Similarly, the number of cycles (Figure 3.12 (b)) of length 4 and 5 is considerably higher with the $-S$ version, which confirms the evidence found in the previous scenarios that the latter executes a less accurate optimization process.

An analysis of the bandwidth consumed by both algorithms, illustrated in Figure 3.12 (c), shows an expected significant growth in the traffic generated by *Discovery* and *Discovery-S* ants. More specifically, the overall traffic required to maintain the final overlay of 10620 nodes is 2670 kbps with BlåtAnt-R, and 1390 kbps with BlåtAnt-S, which scales proportionally to the size of the network (as each node has a 5% probability of generating an ant every 100 seconds).

3.7.4 D - Overlay fault resilience

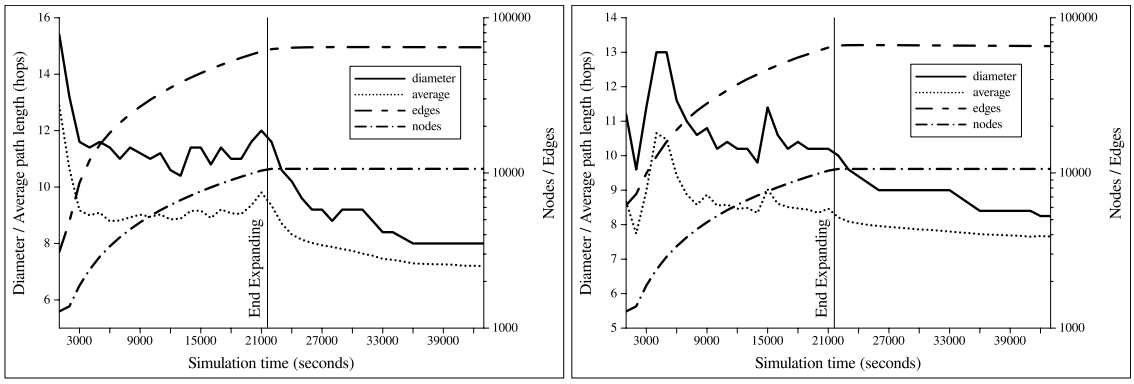
Overlay fault resilience describes the ability of the algorithm to respond to node failures that result in abrupt disconnections without compromising the connectivity of the overlay. Figure 3.13 details the results pertaining to the diameter, edge count, and average path length obtained in our simulation where new nodes connect to the overlay, and existing nodes unexpectedly stop interacting with other nodes and disconnect from the network at intervals of 4 seconds from 30 minutes to 6 hours into the simulation. In all experiments, the overlay remains fully connected, thanks to the emergency recovery procedures that are started by the neighbors of leaving nodes. The recovery activity is highlighted by the increased traffic generated by *Construction-Link Ants*. Results concerning the cycles in the graph, as well as the generated traffic, reflect those obtained in scenario **B**, with only a slight increase in the number of *Construction-Link Ants*.

The behavior of the algorithm in the event of a failure of a large portion of the nodes is shown in Figure 3.14 (a)(b)(c). The results prove the ability of both BlåtAnt-R and BlåtAnt-S to cope with such extreme situations without catastrophic consequences, such as a partitioning of the overlay, even when 750 out of 1281 nodes are simultaneously disconnected (hence results for concurrent disconnection of 25, 50, 100, 250, and 500 nodes are omitted). When 1000 nodes are disconnected the network becomes slightly partitioned, with the size of the largest partition being about 280 nodes (out of 281) on average with BlåtAnt-S and 276 with BlåtAnt-R.

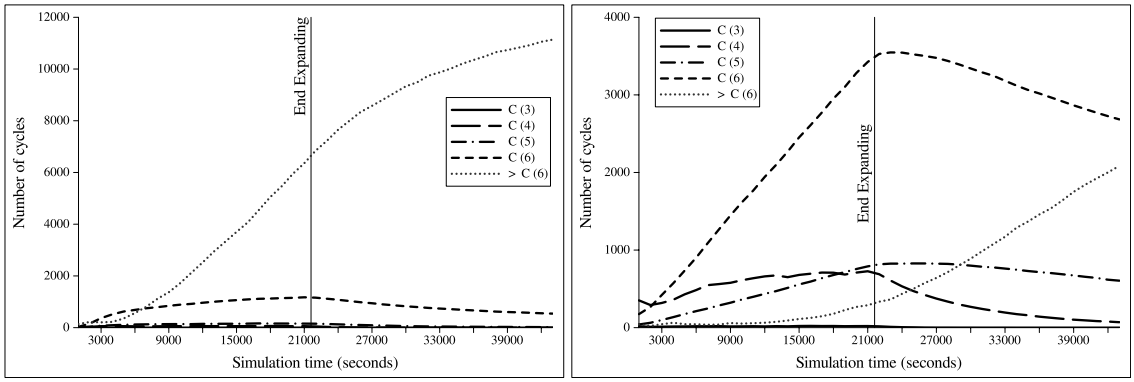
3.7.5 E - Communication fault tolerance

Surviving the loss of information during communication is another important aspect of the robustness of a distributed system. The results of the experiments replicating the conditions defined in scenarios **A**,**B**, and **D** are depicted in Figure 3.15, 3.16, and 3.17 respectively. The obtained results show that both the -R and the -S versions of the algorithm are able to manage loss of information transmitted over the network (namely, ant agents) and communication delays, and maintain the diameter bounded; however, it should be noted that small average path lengths and diameter are mostly due to the

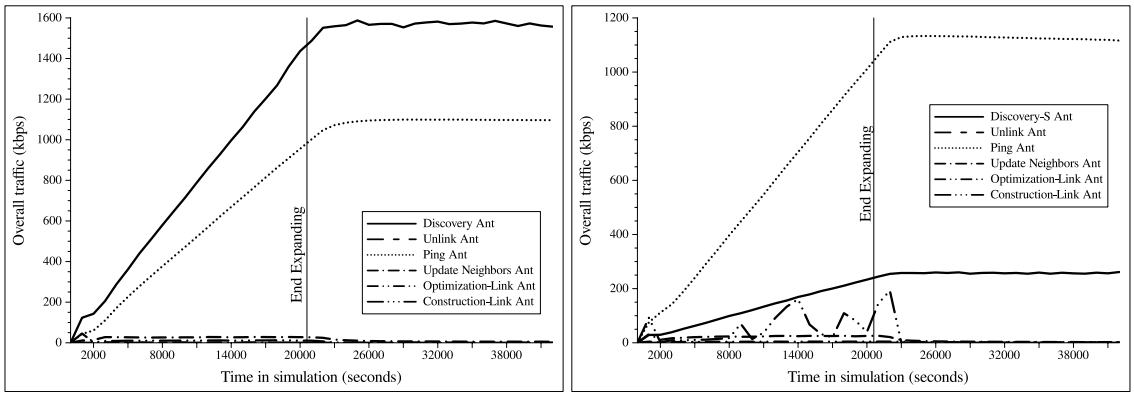
BLÅTANT-R BLÅTANT-S



(a) Diameter, average path length, edge count



(b) Graph cycles



(c) Traffic

Figure 3.12: C - Scalability (diameter, cycles, and traffic)

excessive number of links that are erroneously created by the algorithm, rather than being the result of a controlled behavior. During the dynamic phase, with proper and improper disconnections, the number of edges is heavily influenced by disconnecting nodes, as we note a sharp increase when these node dynamics end.

BlåtAnt-R BlåtAnt-S

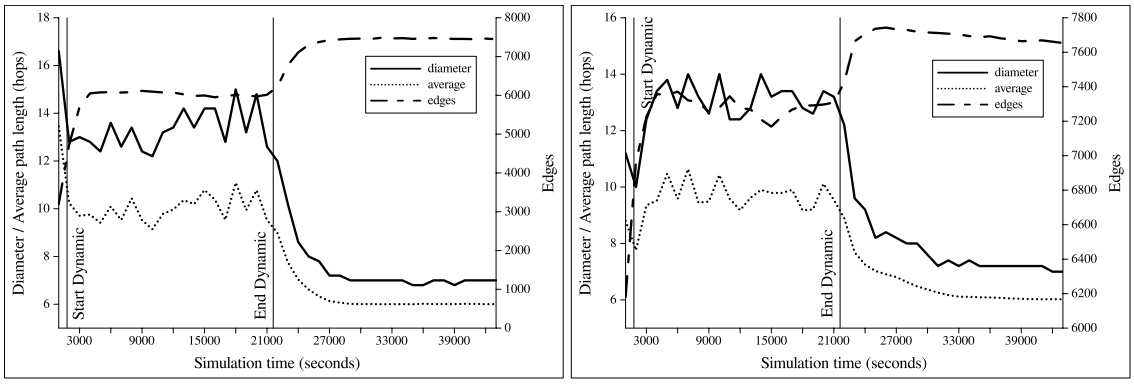
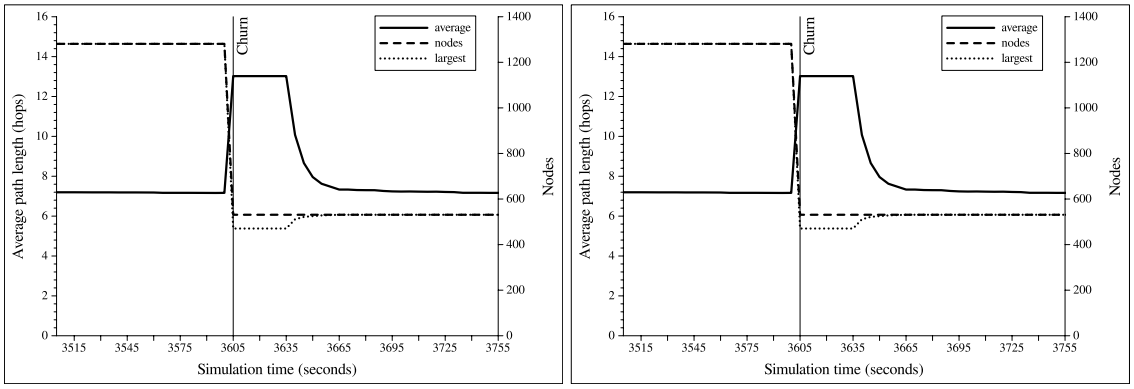
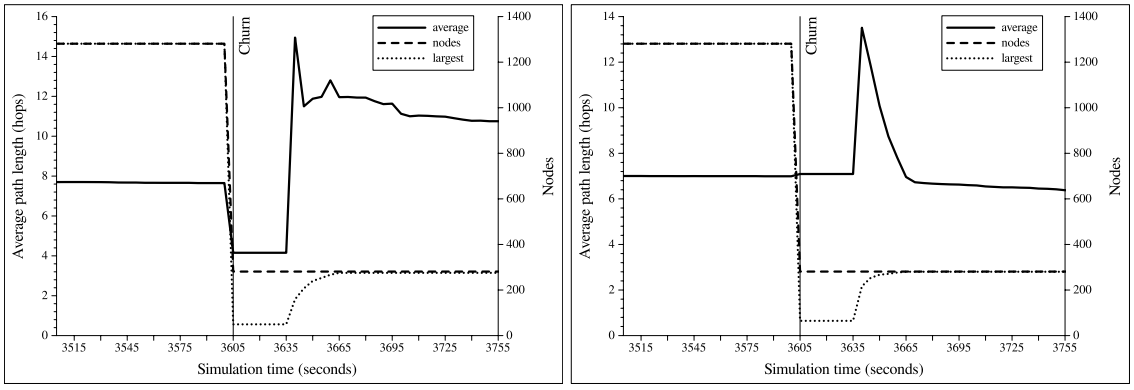


Figure 3.13: D - Overlay fault resilience (diameter, average path length, edge count)



(a) Simultaneous disconnection of 750 out of 1281 nodes

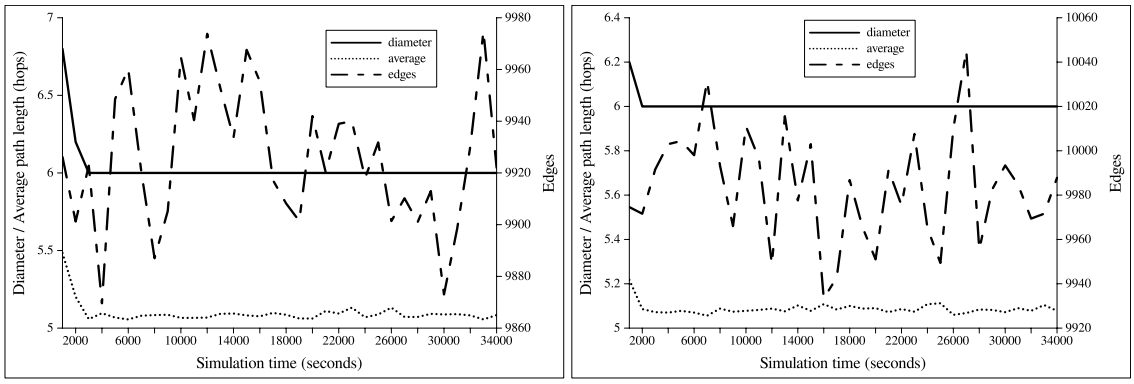


(b) Simultaneous disconnection of 1000 out of 1281 nodes

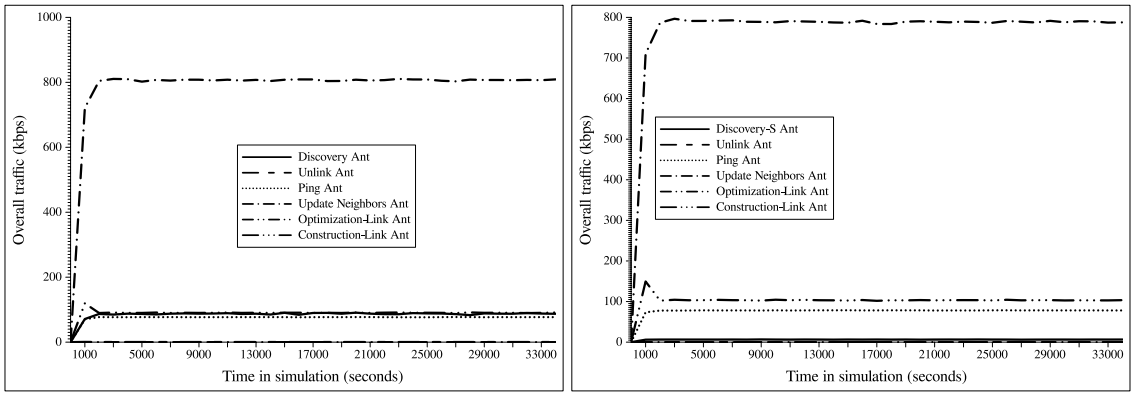
Figure 3.14: D - Overlay fault resilience (average path length, network size, and largest connected component)

Although not shown in the figures, the behavior concerning graph cycles is highly variable, with the presence of a high number of small cycles of length up to 4. During all simulations, despite the harsh conditions of the network, the overlay remains fully con-

BlåtAnt-R BlåtAnt-S



(a) Diameter, average path length, edge count



(b) Traffic

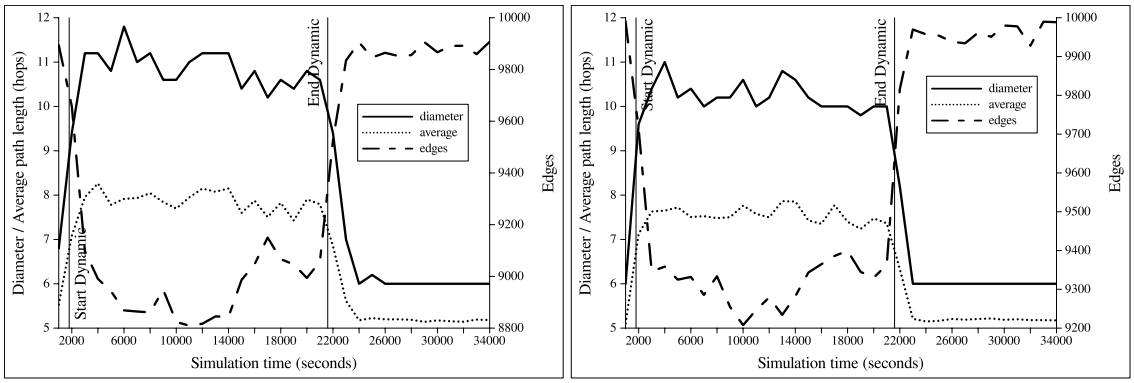
Figure 3.15: E - Communication fault tolerance, stable scenario (diameter, and traffic)

nected; however both the number of links and the traffic significantly increase: concerning the former, we observe an average of 8900 links (-R version) and 9200 links (-S version) during the dynamic phases, and 9900 links (for both algorithms) afterwards. As depicted in the graphs, the traffic increase is due to the *Construction-Link Ants*, which are the result of a high number of recovery procedures that are started when a node stops receiving ants from a neighbor because of lost network packets. Frequent changes in the neighborhood sets can also be observed, as signaled by the number of *Update Neighbors Ants*, which account for the largest portion of the overall traffic.

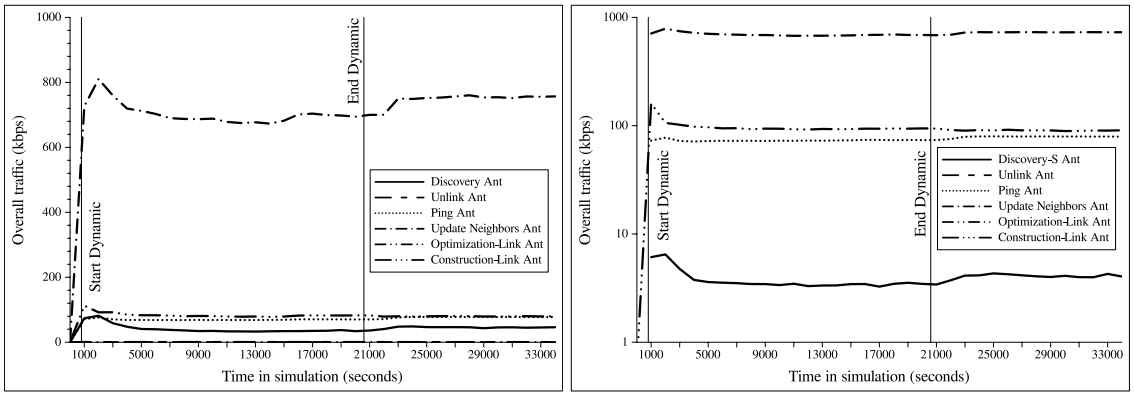
3.7.6 F - Sensitivity

The results obtained through our sensitivity analysis enable a deeper understanding of the influence of each parameter on the outcome of the optimization process, namely the complexity of the resulting topology, as well as its network cost. Each experiment targets one single parameter and results are compared with the reference scenario that is detailed below. Except for the reference scenario, detailed graphs for this section are available in Appendix A.

BlåtAnt-R BlåtAnt-S



(a) Diameter, average path length, edge count



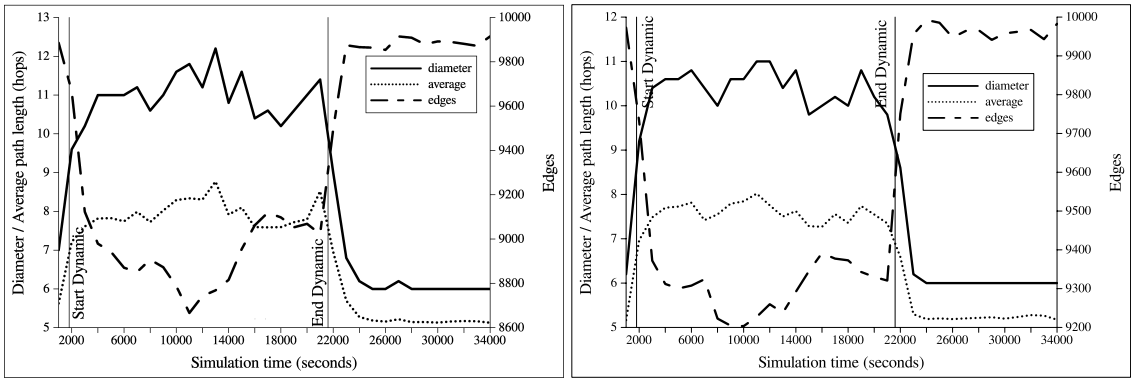
(b) Traffic

Figure 3.16: E - Communication fault tolerance, proper disconnection (diameter, and traffic)

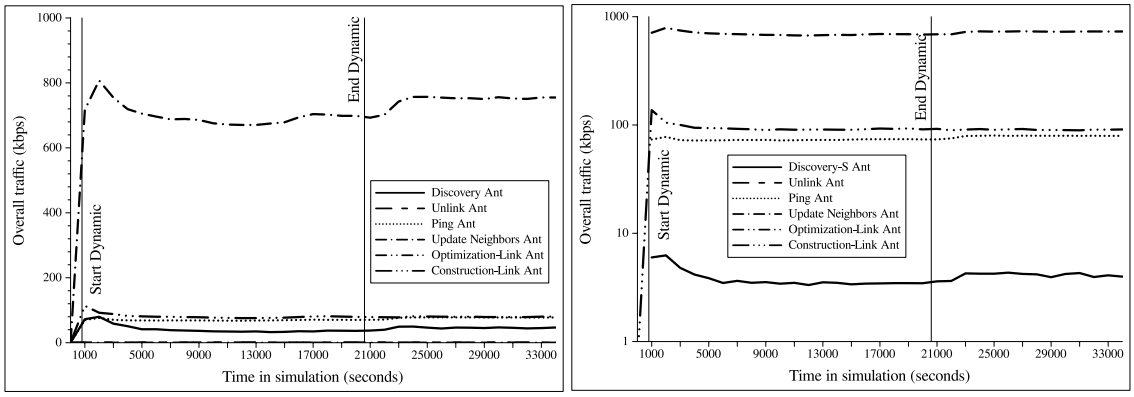
F0 - Reference scenario In the reference scenario the network is stable until 6 hours into simulation; subsequently, one node is added and one is removed every 4 seconds. As shown in figure 3.18 (a), the -R and the -S versions exhibit a similar behavior by reducing the average path length to 6 during the initial phases of the simulation; however, when dynamicity is introduced in the overlay, both algorithms manage to maintain an average path length between 8 and 10. Because a large number of existing nodes leave the overlay, the number of edges reduces from around 7500 (-R) and 7600 (-S), to about 6000 and 7300, respectively.

The information about graph cycles depicted in Figure 3.18 (b) supports the evidence found in early results of scenario **A**: the more accurate distance estimation mechanism employed by BlåtAnt-R compared to the -S version improves the optimization process and prevents cycles of length 3 and 4 even in the dynamic part of the simulation. On the contrary, results in Figure 3.19 (a) show very similar behavior of both algorithms concerning the degree distribution, with the -R version obtaining a slightly smaller number of nodes with maximum optimization degree ($d(6)$); the drop during the dynamic phase is due to the

BLÅTANT-R BLÅTANT-S



(a) Diameter, average path length, edge count



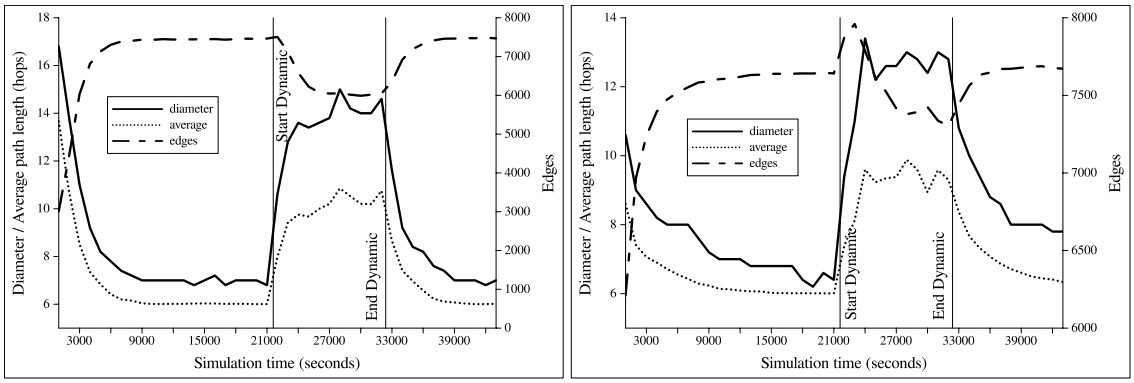
(b) Traffic

Figure 3.17: E - Communication fault tolerance, improper disconnection (diameter, and traffic)

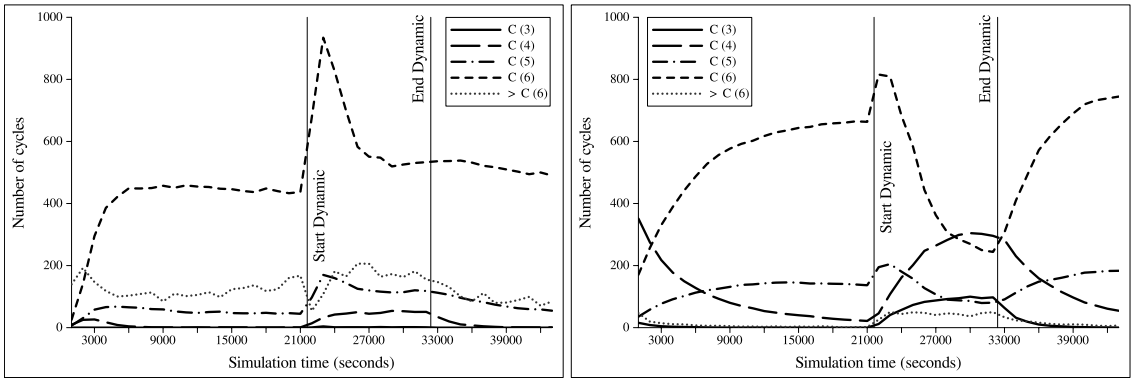
removal of a large number of nodes, and the addition of new nodes that join the network with only one neighbor. Finally, in relation to the generated network traffic, it is possible to note the influence of *Construction-Link Ants* in the dynamic interval, with an average of 50 kbps generated with both versions of the algorithm.

A detailed analysis of the connection and disconnection operations started by both algorithms reveals that the -R version initiates significantly less connections procedures than the -S version, with an average of 15560 and 65292 respectively. These results are reflected in the average bandwidth consumed by *Optimization-Link Ants*, which amounts for 0.37 kbps for the -R version and 0.87 kbps for the -S one. The improved distance evaluation implemented in the -R version also results in a higher percentage of initiated connections that are successfully and correctly completed (i.e. connections that correspond to a correct distance estimation and are accepted by both nodes). More specifically, the -R version attains an average of 46.5%, whereas the -S one only achieves an average of 6.65%. Nonetheless, both algorithms exhibit a similar error rate concerning completed connections, with an average of 47.26% (-R) and 48.98% (-S) of the completed connections resulting

BlåtAnt-R BlåtAnt-S



(a) Diameter, average path length, edge count



(b) Graph cycles

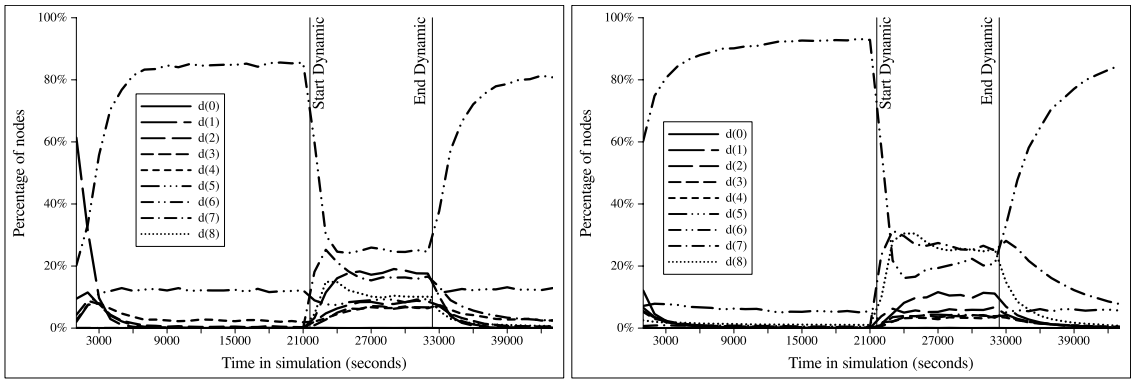
Figure 3.18: F0 - Sensitivity reference scenario (diameter and graph cycles)

from wrong distance estimations.

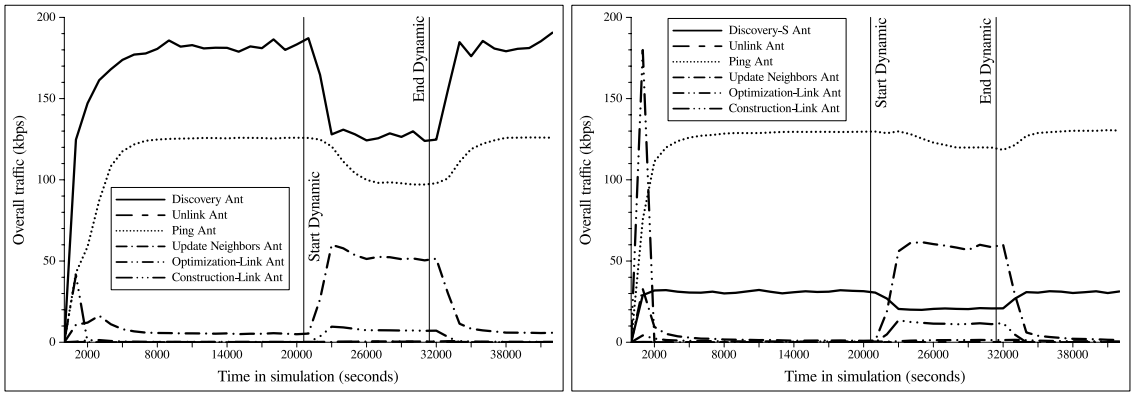
Additional experiments showed that disconnection procedures help improving the convergence of the diameter and average path length. In particular, the achieved average path length at 5 hours 50 minutes into simulation was measured to be 6 with both algorithms when disconnection procedures were enabled; conversely, without disconnections, the average result was 6.37 and 7.5 for BlåtAnt-R and BlåtAnt-S respectively. These results highlight the need for both optimization rules (connection and disconnection) for the proper operation of the algorithm.

F1 - Optimization parameter D Experiments with different values for the parameter D aim at assessing the achievable control over the optimization process. As shown in Figure A.1 (a), differences are noticeable only with BlåtAnt-R, mostly during the initial phase of the simulation, where the average path length converges toward a common minimum of 6. In this regard, only the experiment with $D = 7$ shows a significantly different behavior throughout the whole simulation. Concerning the graph cycles, shown in Figure A.1 (b),

BLÅTANT-R BLÅTANT-S



(a) Degree distribution



(b) Network traffic

Figure 3.19: F0 - Sensitivity reference scenario (degree and traffic)

it is possible to note how smaller values for D result, as expected, in a larger number of cycles of length less than 6. Conversely, the number of edges (Figure A.1 (d)) decreases with larger D values, as less links are required to bound the diameter and to maintain the lower bound on the girth. Ultimately, in respect to the network overhead, we note that the generated traffic is inversely proportional to the value of D, a fact that can be attributed to the reduced number of links which leads to a diminished number of *Ping Ants* that are sent by nodes to their neighbors.

F2 - *Discovery Ant* birth probability μ The optimization process depends on the information collected and spread by *Discovery Ants*. Intuitively, a larger population of such ants would provide more information to each node, possibly improving the outcome of the optimization. As shown in Figure A.2 (a), when ants are not deployed in the overlay (i.e. birth probability equal to 0%), the average path length is about 15 hops with both versions of the algorithm. However, faster convergence toward 6 hops is achieved as the population is increased, and better control during the dynamic phase of the simulation is

obtained. A side-effect of the dynamics of the network also affects the simulation with no *Discovery Ants*, which sees its average path length decrease to a value of 13. With regard to the graph cycles (A.2 (b)) the simulation with no *Discovery Ants* seems to perform better than the others, by exhibiting a smaller number of cycles of length less than 6. This result is easily explained by the fact that typically no cycles are created during the construction phase, as each new node connects as a *leaf* of one of the existing nodes, unless multiple connection requests are started. This fact is also reflected in the smaller number of edges present in the overlay, as shown in Figure A.2 (c). During the dynamic part of the simulation, we note that larger populations help achieve a more stable behavior. Clearly, more ants not only provide better results, but also generate more traffic on the network (Figure A.2 (d)), hence a tradeoff is required: consequently, a birth probability $\mu = 5\%$, chosen as default value in our other experiments, seems to provide satisfactory results.

F3 - Maximum length l_v of the information vector The maximum length of the information vector carried by *Discovery Ants* determines *how far* each node can see in the overlay. Apart from a slight difference in the consumed bandwidth, none of the sensitivity analysis experiments concerning the maximum length l_v of the information vector carried by *Discovery Ants* show significant variations. The observed convergence of average path length (Figure A.3 (a)), as well as the type and length of the detected graph cycles (Figure A.3 (b)) are relatively similar, meaning that small variations in the amount of information available to each node neither benefits nor worsens the optimization process.

F4 - Maximum number of allowed *Discovery Ant* hops π By letting *Discovery Ant* travel for more hops in the overlay increases the number of visited nodes and the amount of information available to each node. As shown in Figure A.4 (a), faster convergence of the average path length can be achieved with more than 15 hops. Concerning graph cycles (Figure A.4 (b)), significant differences are visible only in BlåtAnt-S, with the detection and removal of small cycles improving as ants are given more hops to travel in the overlay. This finding can be explained by the fact that distance evaluations in the -S version are extracted directly from the vector carried by *Discovery Ants*, hence better information quickly leads to an improved optimization process. Unsurprisingly, the amount of traffic generated by the algorithm is proportional to the number of hops each ant travels in the overlay, thus a trade-off is required (Figure A.4 (d)).

F5 - Maximum per-node degree mo The maximum number of neighbors for each node represents the second most important optimization constraint after the parameter D. On one hand, a value that is too small could hinder the optimization process and prevent successful convergence of the diameter and average path length. On the other hand, a value that is too large would permit the creation of large hubs in the overlay, and increase the overall traffic generated by *Ping Ants*. The results illustrated in Figure A.5 (a) show that neither version of the algorithm can ensure the upper bound on the diameter when only 4 neighbors are allowed. Meanwhile, in Figure A.5 (b) we note that a value of $mo = 8$ leaves a larger number of cycles of length less than 6, indicating a less optimized overlay. Therefore, the default value of 6 represents the best trade-off.

F6 - Exploration *versus* Exploitation *Discovery Ant* wandering on the overlay can either choose to follow a random path (*exploration*) or the one with the lowest γ pheromone concentration (*exploitation*). As shown in Figure A.6, no noticeable variation can be detected when the overlay is stable. However, some slight differences can be observed in the dynamic phase of the simulation. In particular, when full exploration (100%) is employed the overall traffic is higher; conversely, with full exploitation (0%), a less optimized overlay with a higher number of cycles smaller than 6 is obtained. These results motivate our default choice of $\kappa = 50\%$ as the more appropriate one.

F7 - Size of the α table The α table maintained by each node contains partial knowledge about the overlay which is used to determine if new connections are to be created and if cycles that are to be broken exist. It is interesting to determine how this information influences the optimization process achieved by BlåtAnt. As expected, the results shown in Figure A.7 show no significant difference between simulations with different table sizes in BlåtAnt-S, as only the connection process depends on the information in the table. On the contrary, with BlåtAnt-R the results concerning graph cycles (Figure A.7 (b)) show that a larger table enables better detection and removal of small cycles. Although a larger table does not increase the traffic generated by the algorithm, it augments the time required to evaluate the information and extract distance estimations, in particular with BlåtAnt-R. Moreover, a larger table is more prone to containing outdated information which could lead to wrong optimization decisions.

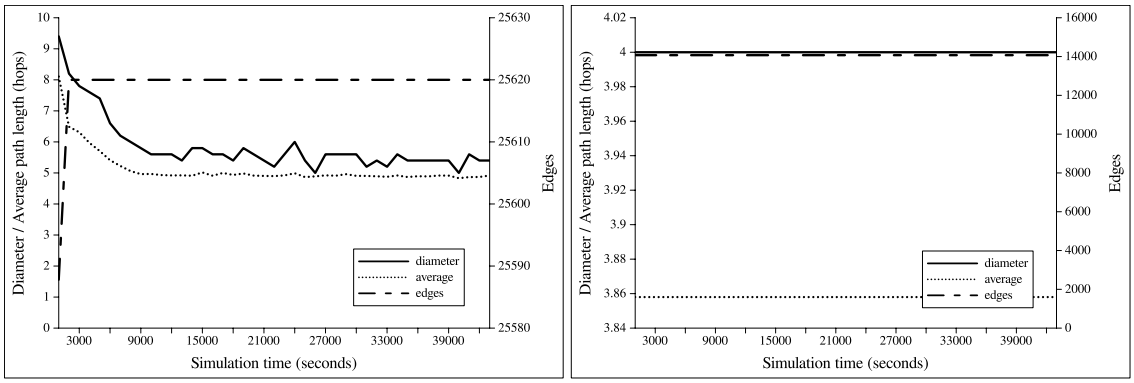
3.7.7 G - Comparison with Newscast and Gnutella

To better understand the benefits of BlåtAnt, we compare here the results obtained in the stable scenario **A**, as well as in the dynamic baseline scenario **F0**, with two other overlay management algorithms, namely Newscast and Gnutella. Comparisons in stable conditions provide useful information about the static characteristics of the resulting overlays, whereas dynamic simulations provide an insight into their behavior in realistic conditions.

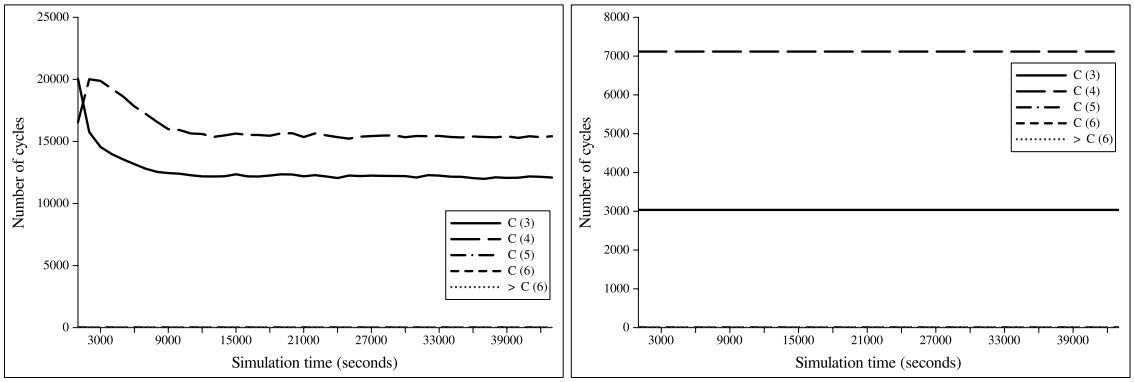
Stable overlay The diameter and average path length are reduced by both Newscast and Gnutella to a value between 5 and 6, and 4 respectively (Figure 3.20 (a)). In contrast to BlåtAnt, it is not possible to constrain these values as they are emerging characteristics of the management algorithm rather than the result of a willful optimization process. The same observations can be made concerning the graph cycles (Figure 3.20 (b)): with both Newscast and Gnutella a large number of small cycles of length 3 and 4 exist in the graph, and the average clustering coefficient is 0.72 for the former (which is compatible with the observations made in [158, 159]), and 0.49 for the latter. Accordingly, in contrast to BlåtAnt (which exhibits an average clustering coefficient of 0), many redundant paths exist in these overlays.

Figure 3.21 (a) shows the stability of the overlay in terms of changed links per second. Because Newscast nodes periodically merge their caches, an average of 35 links change every second even though no nodes are added or removed. On the contrary, with Gnutella as soon as all nodes have filled their available slots after the initialization phase,

NEWSCAST GNUTELLA



(a) Diameter, average path length, edge count



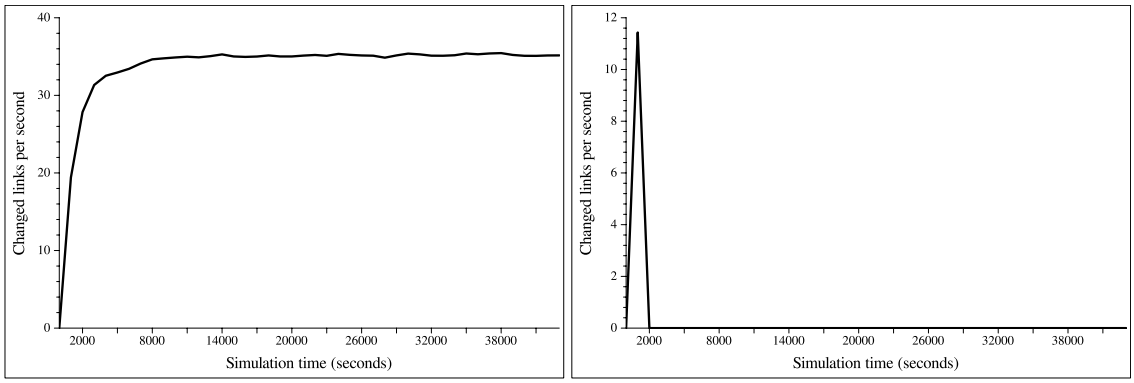
(b) Graph cycles

Figure 3.20: G - Comparison with Newscast and Gnutella (stable overlay - diameter and cycles)

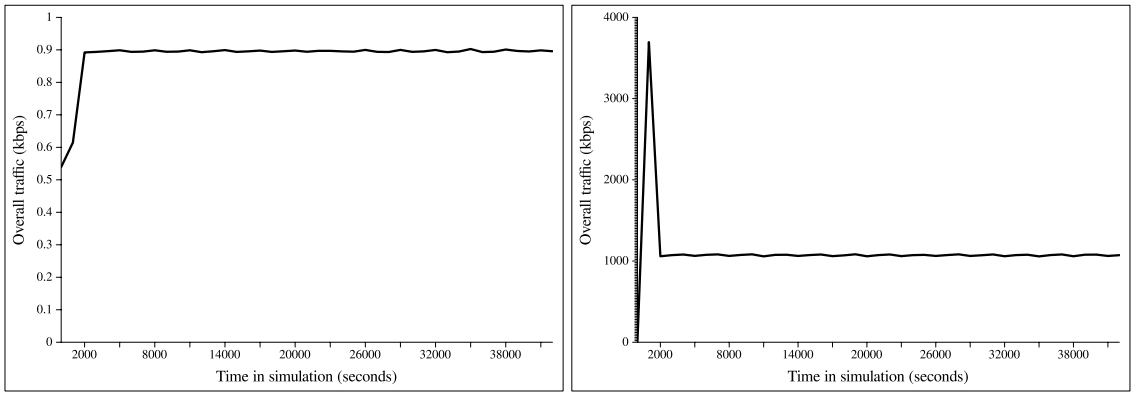
no subsequent changes are made to neighbors, hence the overlay is perfectly stable. In this regard, the results achieved with BLÅTANT in scenario **A** (where less than 1 link is changed every second) are satisfactory. Concerning traffic (Figure 3.21 (b)), it is possible to note that the constant merges in NEWSCAST account for a negligible bandwidth consumption of 0.9 kbps. On the contrary, GNUTELLA requires constant probing of each node's neighbors by means of ping messages, thus its overall bandwidth consumption is considerably higher than both NEWSCAST and BLÅTANT.

Dynamic overlay Simulations in the dynamic overlay (Figure 3.22) show that NEWSCAST exhibits a larger increase of the average path length than GNUTELLA during the dynamic phase of the simulation. Moreover, the diameter in the former is highly variable, reaching a maximum of 18. Network dynamics significantly reduce the number of small cycles in GNUTELLA, a phenomenon that is prevented in NEWSCAST as a result of the constant cache merging process. In both cases, the number of edges at the end of the simulation is reduced because the size of the network shrinks from 1281 to 1280. We note

NEWSCAST GNUTELLA



(a) Stability



(b) Network Traffic

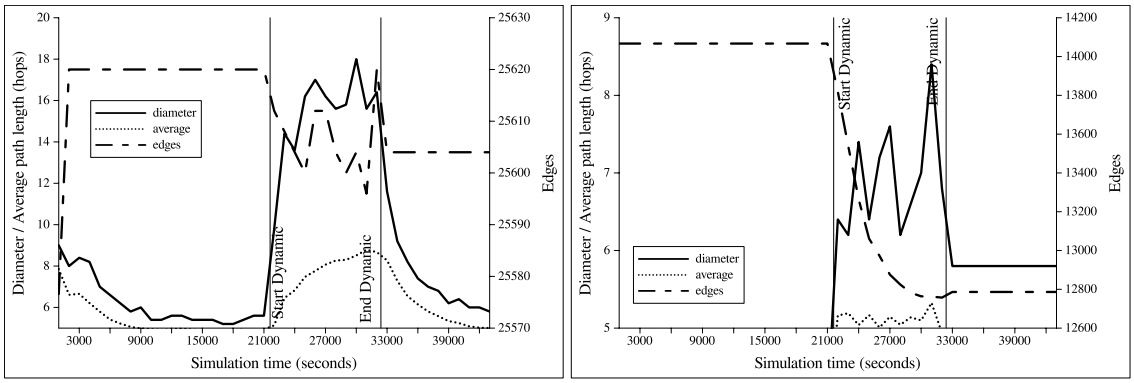
Figure 3.21: G - Comparison with Newscast and Gnutella (stable overlay - stability and traffic)

a sharp increase in traffic in both algorithms (Figure 3.22 (c)), with an increase of over 6 times in NEWSCAST (up to 6 kbps) and more than 15 times in GNUTELLA. Concerning the former, the additional network overhead is caused by new nodes sending pings as they join the overlay, while in the latter the increased traffic is the result of merges performed by new nodes. Compared to BLÅTANT, the bandwidth consumption of NEWSCAST is still lower, while that of GNUTELLA is significantly higher. To this extend, NEWSCAST seems a better competitor for our approach.

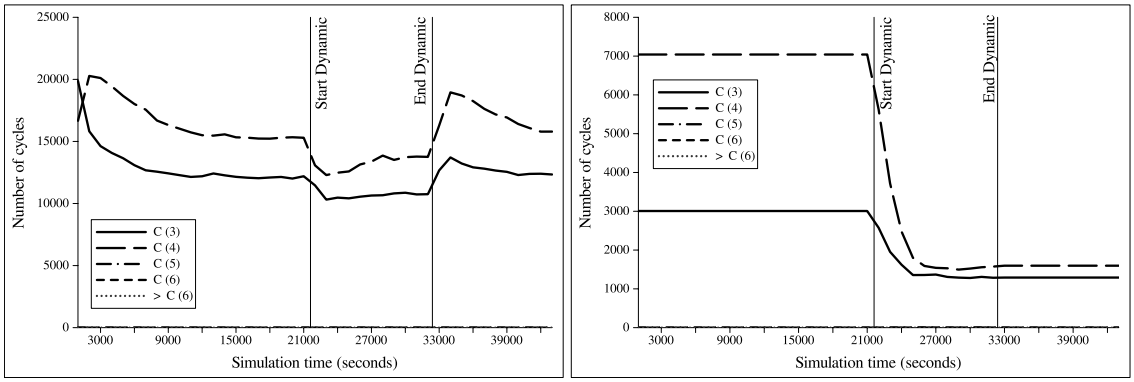
3.8 Accuracy of the results

To determine the accuracy of our data, the following process has been used. For each scenario, the results presented in this chapter refer to averages from 5 simulation runs. In each run, measurements (such as the average path length or the number of edges) have been taken every 1000 seconds, for a total of 44 during the simulated 12 hours of operation of the overlay. For each measurement point, the relative standard deviation across all runs

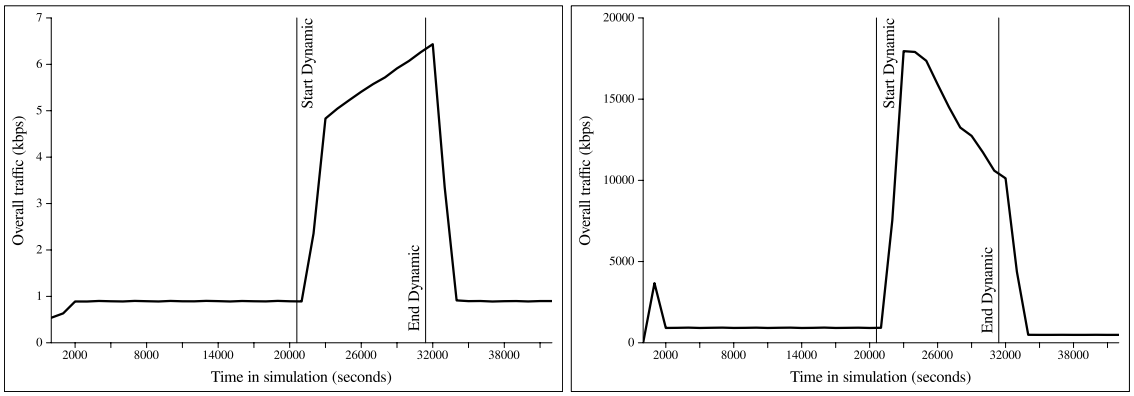
Newscast Gnutella



(a) Diameter, average path length, edge count



(b) Graph cycles



(c) Network Traffic

Figure 3.22: G - Comparison with Newscast and Gnutella (dynamic scenario)

is computed; accordingly we consider the average, as well as the maximum values of the latter in order to characterize the overall accuracy of the data collected throughout our experiments.

In all scenarios, the relative standard deviation for each measurement has been found to

be very low, highlighting the stability of the algorithm and the reproducibility of results. The average of the relative standard deviation across all measurements has also been determined to be negligible. The following statistical information can be extracted for the most relevant measurements:

- **Average path length**: the average of all of the scenario's average relative standard deviation across all periodic measurements is 4.6%, with the highest values being 12.2% (in the sensitivity analysis with BlåtAnt-R and $D = 7$), 11.4% (in the sensitivity analysis with BlåtAnt-R and $mo = 8$), 10.5% (in the sensitivity analysis with BlåtAnt-R and $\pi = 15$), and 10.4% (in the sensitivity analysis with BlåtAnt-R and $\mu = 1\%$). These scenarios can be considered as extreme ones, as either the information collected by nodes was limited (as with $D = 7$, $\pi = 15$, or $\mu = 1\%$), or the constraints were too loose ($mo = 8$) to force a stable converging behavior of the overlay optimization process.
- **Edge count**: the average of all of the scenario's average relative standard deviation across all periodic measurements is 1.66%, with the highest values being 14.2% (in the sensitivity analysis with BlåtAnt-R and $D = 7$), and 10.6% (in the sensitivity analysis with BlåtAnt-R and $mo = 8$).
- **Graph cycles**: the number and length of detected graph cycles are volatile values, and an average relative standard deviations of 33% across all measurements has been determined. It is noteworthy to say that the largest deviations are found in large cycles (of length greater than 5), hence the ability of both algorithms to detect and break small cycles is still valid.
- **Traffic**: the average relative standard deviation is at most 0.1% in all scenarios, which proves that the algorithm has predictable bandwidth consumption.

3.9 Algorithm analysis

The optimization algorithm implemented by BlåtAnt through the *Connection* and *Disconnection* rules described in Section 3.3 relies on graph traversal and single-pair shortest-path resolution algorithms. In this section, the time complexity of a centralized implementation of the algorithm and of the considered fully distributed versions is evaluated.

Centralized algorithm In a centralized implementation of the algorithm, two phases are required: one involving the disconnection of nodes to break up cycles smaller than the predefined threshold, and one to connect nodes in order to bound the diameter of the network. Each phase needs to be repeated as long as either the *Disconnection* or the *Connection* rule apply. In the first phase, all nodes are iteratively processed, the paths between all neighbors are computed, and eventually the *Disconnection* rule is applied. Conversely, during the connection phase, the shortest paths between all pairs of nodes are computed to determine whether the *Connection* rule applies. Because all operations are executed sequentially and neither phase requires the details of each path (traversed nodes) but only the distance between nodes, a breadth-first traversal technique can be employed,

leading to a complexity of $O(N + Nm)$ for each distance evaluation in a network of N nodes with at most m neighbors per node. Since each node needs to be processed, the overall complexity of each phase is $O(N^2)$.

BlåtAnt-R Each node implementing BlåtAnt-R locally employs an approach similar to the centralized algorithm, although on considerably smaller graphs based on local and partial information. In particular, the size of the partial network constructed from the cache table α is determined by the size table $\lceil|\alpha|\rceil$ and the maximum number of neighbors per node m. Concerning the *Connection* rule, the complexity for evaluating the whole table using the breadth-first algorithm is $O(\lceil|\alpha|\rceil^2 m^2)$. In contrast to the centralized solution, to safely apply the *Disconnection* rule the list of traversed nodes must be determined in order to identify the master of each cycle. Accordingly, a breadth-first traversal cannot be employed to determine the distance between neighbors of each node. Supposing that the Djikstra ([93]) algorithm is employed instead, the worst-case time complexity to evaluate the whole cache table is $O\left(\binom{m}{2}(\lceil|\alpha|\rceil m\ log(\lceil|\alpha|\rceil m) + \lceil|\alpha|\rceil m)\right)$. Several efficiency improvements can be implemented to further reduce the computational load on each node; for example, unreachable nodes can removed from the partial graph as soon as they are detected, reducing the time spent for subsequent distance evaluations.

BlåtAnt-S In contrast to the -R version, BlåtAnt-S determines distances solely on the position of the elements within information vectors collected by *Discovery Ants*. Accordingly, the time complexity is $O(l_v)$ each time a node processes an incoming information vector.

Through an empirical investigation of BlåtAnt-R we have observed a linear growth of the processing times required by the connection phase (evaluation of distances in order to create additional links) in relation to the size of the α table. More specifically, we measured 0.24 ms with $\lceil|\alpha|\rceil = 28$, 0.31 ms with $\lceil|\alpha|\rceil = 36$, and 0.7 ms with $\lceil|\alpha|\rceil = 84$. Also on BlåtAnt-R a less-than-linear growth was found regarding the maximum number of neighbors m, with 0.24 ms with $m = 8$, 0.28 ms with $m = 16$, and 0.29 ms with $m = 24$. In this respect, it should be noted that the increase of the maximum number of neighbors has not resulted in a significant increase in the actual edges in the network, meaning that the majority of the nodes retained as small a number of neighbors as necessary to maintain a bounded diameter. This phenomenon also influenced the average time spent during the disconnection phase (evaluation of distances between neighbors in order to break small cycles): with respect to both $\lceil|\alpha|\rceil$ and m a less-than-linear growth instead of a quadratic one was observed. More specifically, the measured times were 0.2 ms with $\lceil|\alpha|\rceil = 28$, 0.26 ms with $\lceil|\alpha|\rceil = 36$, and 0.5 ms with $\lceil|\alpha|\rceil = 84$, and 0.2 ms with $m = 8$, 0.22 ms with $m = 16$, and 0.22 ms with $m = 24$ respectively. Regarding BlåtAnt-S the measured growth related to an increase of l_v was found to be linear, validating our formal analysis. As expected, the processing times for both the connection and the disconnection phases were also considerably shorter than with the -R version, ranging from 0.02 ms with $l_v = 15$ up to 0.03 ms with $l_v = 45$ for the connection phase, and < 0.01 ms for the disconnection phase in all experiments up to $l_v = 45$.

3.10 Summary

In this chapter the BLÅTANT algorithm was thoroughly detailed and evaluated. By means of a fully distributed, bio-inspired algorithm, the topology of the overlay is optimized to bound the diameter as well as the girth. Whereas the first goal diminishes search response times by limiting the maximum number of hops traveled by queries, the second one mitigates the problem of redundant message transmission by reducing actively breaking small cycles in the overlay. The fundamental principles of the algorithm have been discussed, and two implementations were presented. An in-depth analysis of the behavior of the algorithm in several network conditions, conducted by means of a simulator, validates the proposed approach and demonstrates the suitability of BLÅTANT for real-world deployments.

Chapter 4

Resource Discovery

Contents

Resource discovery is a service that allows users and applications to find the resources shared by remote systems connected to the overlay by means of a querying mechanism. An important aspect to define the qualities of a discovery mechanism is its ability to resolve queries with minimal bandwidth consumption, while providing satisfactory hit rates. In [151], the authors identify several requirements for successful resource discovery in dynamic environments, such as fully decentralized operation, support for attribute-based search, scalability and ability to cope with highly dynamic environments. In accordance

with our view presented in Chapter 3, that dealt with the problem of constructing an optimized unstructured peer-to-peer overlay using a fully distributed algorithm, the aim of this chapter is to investigate efficient decentralized search methods to further exploit the properties of such system. We thus propose a fully distributed search mechanism [53, 54] inspired by existing techniques, especially routing indices, replication and clustering. In this regard, our effort is to provide each node with a cache containing addresses of other peers in the network that share similar resources or services, and forward resource discovery queries toward those nodes whenever possible, in order to increase the number of hits at minimal cost. More precisely, the presented solution builds loosely coupled clusters, and is complementary to other techniques aimed at improving search in unstructured overlays, such as replication, teeming and selective forwarding.

The rest of this chapter completes our review of search in unstructured peer-to-peer systems presented in Chapter 2 with an additional discussion of similar techniques that relate to our approach. This is followed by an overview of the proposed local caching mechanism and by a thorough validation by means of simulations.

4.1 Enhancing semantic-aware resource discovery

Semantic-aware resource discovery exploits the information concerning both the resources and the queries in order to improve search efficiency. Of particular interest for our research are two semantic based techniques, namely routing indices [83] and clustering [114]. Furthermore, replication can be exploited to increase the likelihood of finding matching resources and reduce search delays [195]. In this respect, the information might be either *naturally* replicated, as it is the case with popular content in file sharing networks, or *proactively* replicated as the result of a replication algorithm. Routing indices employ semantic information to forward queries toward nodes that are more likely to provide the requested service, while clustering solutions group semantically similar resources with the goal of making forwarding more efficient as well as increasing the hit (or recall) rate once a hit has been found.

Examples that employ semantic information to route queries include [301, 73], where it is suggested to use local indices to forward search queries toward nodes that are more likely to satisfy them. The forwarding policy is normally based on *satisfaction* indices that are evaluated based on past experiences, namely successful query responses. Building on the principles of the routing indices paradigm, different propagation strategies can be implemented, as suggested in [151]. A similar solution is employed in [227], to implement a grid information service based on peer-to-peer technologies that uses routing indices to direct queries toward the closest known node that might fulfill the request. Figure 4.1 illustrates an example of resource discovery querying for a resource on node E: by using local information in the routing table, the query is first forwarded toward node B, and finally to node R that provides a match.

In contrast to routing indices, which are concerned with query routing, clustering mechanisms focus on organizing and replicating information in order to group them into semantically similar groups. From this point of view, an interesting and self-organized solution geared toward grids is ANTARES [114]: by employing a swarm intelligence algorithm, clusters of references to nodes sharing similar resources are created. Ant inspired mobile agents

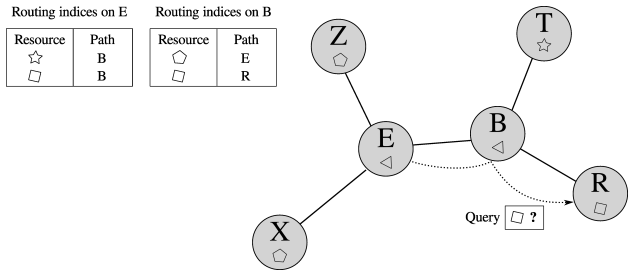


Figure 4.1: Resource discovery with routing indices

wander across the network, and transfer resource descriptors following a simple algorithm: on a node, if an ant is not carrying any descriptor, it randomly picks the one that is less similar to the others and then continues wandering; conversely, if the ant is carrying a descriptor, it will release it with a probability proportional to the similarity between the carried descriptor and the descriptors stored by the node. The outcome of this process is that descriptors that are similar will be likely on the same node or on nearby nodes. Resource discovery in ANTARES is started as a blind random walk search; when a node that shares resources similar to what is being queried is found, the search becomes informed and forwarding is done toward nodes that are the most similar to the target specified in the query. The benefits of clustering are twofold: on one side during resource discovery semantic information can be used to route the query toward a matching node; on the other side, when a matching node is found, additional hits can be resolved nearby with limited bandwidth consumption by contacting nodes in the neighborhood. Figure 4.2 depicts an example of resource discovery querying for resources similar to the ones referenced by descriptors stored on R: (a) the random walk process progresses toward node R, and then (b) flooding the neighborhood allows finding additional results.

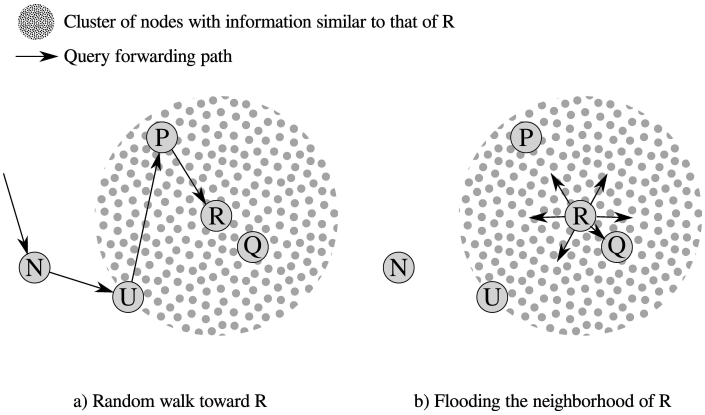


Figure 4.2: Resource discovery with clustering

A solution inspired by both routing indices and clustering is presented in [267]: each node in a Gnutella-like network maintains a list of shortcuts to other nodes that share similar interests. These shortcuts are discovered by performing searches using a flooding protocol, and are subsequently used to find additional shortcut candidates. For resource discovery, peers try to use the available shortcuts and fall back to flooding only if none of

the shortcuts has the requested content. In a similar way, the solution presented in this chapter builds on the principle of creating and maintaining local caches on each node, that contain addresses of other nodes in the overlay that share similar resources. These caches represent small clusters that can be exploited during resource discovery: more specifically, when a node matching the query is found, the forwarding of the query continues for some additional steps toward nodes in the cache, as the referenced nodes are more likely to provide additional results. Content in the cache is obtained by means of proactive resource discovery queries as well as by exchanging it with other nodes using an epidemic protocol.

4.2 Proactive caching

While the overlay topology maintained by BlåtAnt enables optimized communication by actively bounding the maximum distance between each pair of nodes, and also by reducing the number of redundant paths between nodes, obtaining satisfactory resource discovery hit rates by broadcasting a query on the network still requires visiting a large number of nodes. The aim of the proposed resource discovery approach is to increase the hit rate by exploiting cached information in order to minimize the average bandwidth consumed to obtain each result. Because transferring data across the overlay generates additional traffic, we target a positive trade-off between the increased number of hits and the additional bandwidth related to resource discovery.

4.2.1 Resource profiles

Each node in the overlay shares some information or resources with other nodes: the characteristics of shared content can be referred to as the *resource profile* of a node. A resource profile can be represented by a vector of tuples that describe the different aspects of the resource: for example, a node in a computing grid is characterized by the services offered, the CPU architecture, the amount of memory, etc. The information contained in a profile need not to be static; in particular, it is possible to distinguish between static and dynamic aspects: whereas the former are concerned with characteristics that are not likely to change across time (as for example, the CPU architecture), the latter focus on values that change across time (for example, the available memory, which depends on the status of the node, the current active tasks, and the scheduling policy). Accordingly, a dynamic caching mechanism that can take into consideration possible changes in the availability of resources is required.

4.2.2 Profile similarity

Information in the cache contains references to nodes sharing similar contents or resources, and are thus likely to match the same queries. In order to determine if two resource profiles are similar we use a *similarity function* to express the distance between profiles as a real value. We propose here two similarity functions, namely a *cosine similarity* function and a *difference vector* one.

Cosine similarity Given two nodes n_i and n_j, their resource profile vectors p_i and p_j, a suitable scalar product operation, and a norm $\|.\|$, we consider a *cosine similarity function* $\Lambda(p_i, p_j) \in [0, 1]$, such that

$$\Lambda(p_i, p_j) = \begin{cases} \frac{p_i \cdot p_j}{||p_i|| ||p_j||} & \text{if } \frac{p_i \cdot p_j}{||p_i|| ||p_j||} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The scalar product and the norm have to be defined such that the profiles are equivalent iff $\Lambda(p_i, p_j) = 1$, and *similar* iff this value is close to 1 according to a user-defined threshold.

Difference vector similarity The similarity value can be simplified if values in the resource profile map onto a discrete ordered sets. In this case, the similarity function can be computed by first creating two vectors, one for each profile, that map the profile values to the ordinals in the discrete domains, and then computing the maximum of the absolute value of the components of the vector. For example, given the following discrete sets:

$$memory = [512MB, 1GB, 2GB, 4GB, 8GB],$$

$$cpu = [1GHz, 2GHz, 3GHz]$$

and two resource profiles $A = [2GB, 2GHz]$ and $B = [8GB, 3GHz]$, the resulting vectors are:

$$A' = [3, 2], B' = [5, 3]$$

The similarity function is then computed as the maximum absolute value of the components within the difference vector, i.e:

$$max(|A' - B'|) = max(|3 - 5|, |2 - 3|) = max(2, 1) = 2$$

Depending on the considered application, two resource profiles can be considered similar if the value of the function is 1, 2, or a bigger value.

4.2.3 Similar peers cache

Each node keeps a cache table of size c_{size} storing identifiers and timestamps of other nodes with a similar profile. The timestamp is used to determine the age of an entry. The cache is updated at regular intervals by starting *proactive* resource discovery queries to search for other nodes in the network having a similar profile. Results from proactive queries are stored in the table and replace existing entries. Similarly to routing indices, the shortcuts contained in the cache form a second-level overlay, where each node's neighborhood is composed of peers with similar resource profiles. Figure 4.3 illustrates an example overlay and the cache on node A, which contains references to nodes that share similar resources according to the defined resource domain.

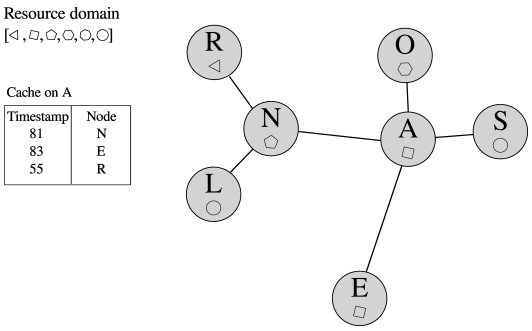


Figure 4.3: Example overlay and detail of the local cache on node A

4.2.4 Cache merging

Maintaining up to date cache information through proactive resource discovery queries may lead to high network overhead. We thus introduce a cache merging mechanism that enables nodes to share their cache contents with peers having similar profiles. This avoids flooding the network with proactive queries, in favor of a pairwise exchange of a small number of node identifiers.

The process itself is inspired by the Newscast [159] epidemic algorithm. At regular intervals, each node randomly chooses a peer from within its cache contents and initiates a merging procedure. The initiating peer requests the content of the remote cache, merges them with the local cache, and retains at most the $c_{size} - 1$ entries with the highest timestamp (i.e. the most recent information). Both the initiating node and the remote node will then replace their own caches with the resulting set. Finally, the initiating node will add the remote peer identifier, along with an updated timestamp to its cache. Conversely, the remote peer will add the initiating node's identifier and updated timestamp to its cache. It should be noted that such simple merging mechanism could be replaced by a more advanced merging scheme in future work.

4.2.5 Enhanced resource discovery

Resource discovery is performed using a limited and probabilistic flooding algorithm. Limited flooding implies that nodes keep track of received queries, and avoid forwarding queries that have already been processed. Probabilistic flooding means that, at each step, the query is forwarded only to a subset of all neighbors. In our approach, the subset is constructed by uniformly sampling the neighborhood set. We consider the query as *successful* when at least one node matching the query is found; conversely, each resource found counts as a *hit*. Accordingly, the *hit rate* (*recall rate*) measures the percentage of successfully discovered resources out of all matching ones.

The peer cache itself is exploited by non-proactive searches to enhance the hit rate: when a matching resource is found, instead of stopping the search the query *jumps* to the node cache and continues for an additional number of steps. In this way, there is a high probability of reaching additional hits because of the way the cache has been constructed.

Figure 4.4 depicts an example of resource discovery exploiting the local cache. When the query reaches node A, a match is found, hence the forwarding continues using the

shortcuts available in the cache, leading to an additional hit on node E.

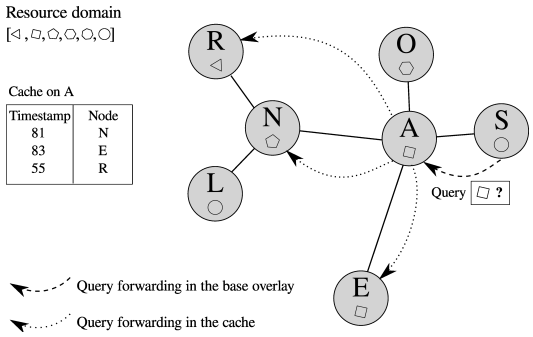


Figure 4.4: Resource discovery exploiting local caches

4.3 Evaluation

We conducted a detailed experimentation of the proactive caching, and compared its performance against basic resource discovery. For evaluation purposes, and in line with the thesis evaluation scenario, a grid network has been considered. In this section the details of the simulation tests are presented and discussed.

4.3.1 Simulation setup

Simulation of resource discovery is performed by randomly choosing both a starting node and a search profile. A set of 5 simulation runs of 12 hours were evaluated: 5 search queries are regularly started every 60 seconds, beginning at 30 minutes into simulation and ending at 12 hours, resulting in a total of 3450 queries per run. The results detailed in the next section thus represent an average over the latter number of requests.

4.3.2 Peer-to-Peer Overlay

The underyling overlay is constructed and maintained by BlåtAnt-S, which was chosen for its lower network overhead compared to BlåtAnt-R. Similarly to BlåtAnt ants, resource discovery queries also contribute in reinforcing β and γ pheromone trails as they propagate across the network. As with the reference scenario **F0** discussed in Chapter 3, the network is bootstrapped starting from an initial random lattice consisting of 10 *well-known* nodes. In the first phase of the evaluation, additional 1271 nodes connect to the overlay, up to a total of 1281 nodes. Overlay parameters, as well as the simulated rates and intervals of connections and disconnections are as in the reference scenario. Accordingly, the expected average path length in the overlay is 8, although the TTL of resource discovery queries has been set to 5 in order to highlight the benefits of the cache mechanism while retaining a reasonable traffic overhead. To evaluate resource discovery in dynamic network conditions, the overlay is modified at runtime by having new nodes joining the overlay every 4 seconds in the period between 6 and 9 hours into simulations. Upon disconnection, nodes either leave the overlay properly or abruptly.

In order to assess the impact of different overlay management algorithms on the performance of the proposed caching mechanism, we also experiment with topologies constructed and maintained by NEWSCAST and GNUTELLA. Concerning the former, each node manages a NEWSCAST cache table of 20 entries, which is merged with other peers every 10 minutes on average. Conversely, in GNUTELLA experiments each node maintains at most 10 neighbors (excepted for *well-known nodes*), and ping messages are forwarded every 30 seconds at a distance of 7 hops in the overlay to at most 4 neighbors at each step.

4.3.3 Evaluation scenarios

As our evaluation aim at assessing the performance and robustness of the proposed resource discovery mechanism, different evaluation scenarios experimenting with different parameter values have been considered. In particular we performed an analysis of the *hit rate* (also known as *recall rate*), which represents the percentage of discovered resources matching the query out of all matching ones available in the network, of the generated traffic, as well as of the sensitivity of the algorithm toward parameter variations. Accordingly, a number of evaluation scenarios (a listing of which is available in Table 4.2) have been simulated. The parameters that have been considered in our analysis, as well as the default values employed in our experiments (unless otherwise stated), are detailed in Table 4.1.

	Value	Description
TTL	5	Resource discovery query time-to-live (hops in the overlay)
FW	4	Probabilistic forwarding sample size (number of neighbors)
$M - Int$	15	Cache merge interval (in minutes)
$P - Int$	45	Proactive queries interval (in minutes)
$C - TTL$	3	TTL while traveling within the cache (in hops)
$C - FW$	3	FW while traveling within the cache (number of cache entries)
$P - TTL$	4	Proactive queries TTL (in hops)
$P - FW$	3	Proactive queries FW (number of neighbors)
–	*None*	Replication strategy (*None*, *one-hop*, *5-hops*)

Table 4.1: Summary of resource discovery evaluation parameters

By means of our evaluation we aim at assessing the influence of these parameters on both the hit rate and the consumed bandwidth.

Scenario	Focus of the evaluation
A	Benefits of proactive caching
B	Benefits of proactive caching with replication
C	Sensitivity
D	Comparison on NEWSCAST and GNUTELLA overlays

Table 4.2: Summary of the resource discovery evaluation scenarios

4.3.4 Resource profiles

Upon creation, each node is assigned a random static resource profile that does not change during evaluation. Profiles are comprised of several fields that describe both hardware and software properties of the machine. In particular, we consider the implemented architecture (e.g. AMD64, POWER, etc.), available memory, available disk space, and operating system (e.g. LINUX, SOLARIS, etc.). Values for each field are chosen with different probability distributions, as follows:

- **Architectures** are chosen according to the list published on the *TOP500 Supercomputing Sites* (www.top500.org) at the time of the writing of this thesis. The probability distribution is as follows: AMD64 87.2%, POWER 11%, IA-64 1.2%, SPARC 0.2%, MIPS 0.2%, NEC 0.2%;
- **Available Memory and Disk Space** are both independently and uniformly chosen as either 1, 2, 4, 8, or 16 Gigabytes;
- **Operating Systems** installed on each node are based on the aforementioned *TOP500* list, with the following distribution: LINUX 88.6%, SOLARIS 5.8%, UNIX 4.4%, WINDOWS 1%, BSD 0.2%.

The simulator generates resource discovery queries with random profiles according to the aforementioned distribution, that will be matched by nodes on the overlay. To compute profile similarity, the architecture and operating systems are considered as discriminant aspects, thus two profiles with different values are always considered as *non similar* (similarity value equal to 0). On the other side, a similarity value can be computed for profiles with matching operating system and architecture, using the early mentioned difference vector similarity function. More precisely, given two resource profiles a, b, and the corresponding values for memory and disk space a_{mem}, b_{mem}, respectively a_{disk}, b_{disk} we consider *a similar* to *b* if $\frac{b_{mem}}{2} \leq a_{mem} \leq 2 \cdot b_{mem}$ and $\frac{b_{disk}}{2} \leq a_{disk} \leq 2 \cdot b_{disk}$. It is important to note this value for similarity is not commutative.

4.3.5 Traffic Evaluation

To evaluate the amount of bandwidth consumed by resource discovery, the following traffic estimations have been considered:

- **resource discovery queries / replications**: 5 KBytes;
- **resource discovery query replies**: 128 bytes;
- **replication**: 5 KBytes *per* hop;
- **cache merge**: 1064 bits *plus* 176 bits/cache entry;
- **ping**: 704 bits $(2 * (320 + 32)$ bits ICMPv6).

These estimations are based on the actual information carried by each ant, and include both the size of an IPv6 header (320 bit), a UDP header (64 bit), as well as a 4 bits

packet type identifier and 144 bits source identifier (128 bits IPv6 identifier *plus* 16 bits port number). The obtained traffic results are based on an average cost over the total number of 3450 queries, and include the overlay management, the proactive caching task (if applicable), replication (if enabled), and resource discovery. The bandwidth consumed by the overlay management algorithm and by proactive caching does not depend on the resource discovery activity, and it should thus be considered as a fixed cost shared among all queries.

4.3.6 Scenario details

The rest of this section discusses the algorithm parameters used in each scenario according to the focus of the evaluation. A detailed overview of the default parameter values used in each scenario is shown in Table 4.1, unless otherwise specified; the corresponding results are presented in Section 4.4.

A - Benefits of proactive caching Scenarios **A** evaluate the benefits of the proposed proactive caching scheme. In order to setup a baseline for comparison, several simulations without caching that employ a simple probabilistic flooding protocol and different query forwarding strategies have been performed. More specifically, we experimented both with fixed query TTLs equal to 5 and varying number of contacted neighbors $(3, 4, 5, 8)$, as well as with TTL varying between 5 and 9 and fixed number of neighbors equal to 4. The same experiments have been repeated with proactive caching enabled. Proactive resource discovery queries are started every 45 minutes (*P-Int*), while cache merging happens on each node with an average period of 15 minutes (*M-Int*). Each proactive query is forwarded up to a distance of 4 hops in the overlay (*P-TTL*): at each forwarding step, 3 neighbors are contacted (*P-FW*). The cache on each node stores at most 5 entries (*C-size*). Once a hit is found, resource discovery queries may travel at most 3 hops in the overlay (*C-TTL*), contacting 3 peers at each step (*C-FW*). From this set of experiments, the one employing a forwarding strategy of 5 hops and 4 neighbors is considered as baseline for all other scenarios, because, as it will be made clear by the results, it provides one of the lowest cost per hit.

B - Benefits of proactive caching with replication To assess the influence of replicated contents on the benefits brought by proactive caching, scenario **B** experiments with two different replication strategies: one-hop replication and replication at a distance of 5 hops. While the first strategy represents a typical choice in peer-to-peer systems, the second one is tailored for the BlåtAnt overlay constructed with $D = 5$, because the size of cycles in the graph should enable traveling for 5 hops away from a node without following redundant paths. The considered query forwarding and proactive caching parameters are as in scenario **A**.

C - Sensitivity Whereas previous scenarios aim at assessing the improvements derived by our proactive caching approach, sensitivity analysis scenarios focus on determining how algorithm parameters values affect the performance of the system.

- **C1 - Sensitivity to hops traveled in the cache (C-TTL, C-FW)**: to evaluate the impact of cache navigation strategies we conduct several experiments with different C-TTL and C-FW. The value of C-TTL is changed from the default 3 hops, to 2 and 5 hops, whereas the C-FW is changed from the default 3 neighbors, to 1, 2, and 5.

- **C2 - Sensitivity to the cache merge interval (M-Int)**: cache merges enable information sharing between nodes, and help cleaning old entries from caches, thus preventing references to missing nodes that have already left the network. We assess the performance of the algorithm pertaining to the frequency of merges by varying the merge interval from the default of 15 minutes to 7m 30s, 30m and 60m.

- **C3 - Sensitivity to the proactive query interval (P-Int)**: proactive queries are the primary mechanism used to update the cache. We gauge the benefits of more or less frequent proactive querying by experimenting with different intervals, namely 15m, 30m, 45m (the default value used throughout the rest of the experiments), and 1h 30m.

- **C4 - Sensitivity to the proactive query spread (P-TTL, P-FW)**: this scenario experiments with different forwarding strategies concerning proactive queries. Specifically, we change the value of *P-TTL* from the default of 4 hops to 3 and 5, and the value of *P-FW* from 3 neighbors to 2 and 4.

- **C5 - Sensitivity to network stability**: all previous experiments are conducted on a network with dynamic characteristics, where the overlay is modified at runtime by having new nodes joining the overlay every 4 seconds in the period between 6 hours and 9 hours into simulations. In this set of simulations we aim at assessing the influence of such network dynamics on the performance of resource discovery. In particular, the results obtained in scenario **A** are dissected to obtain average results before the dynamic phase (i.e. before 6 hours in to simulation), during the dynamic phase (i.e. between 6 and 9 hours into simulation), and after the dynamic phase (i.e. after 9 hours into simulation).

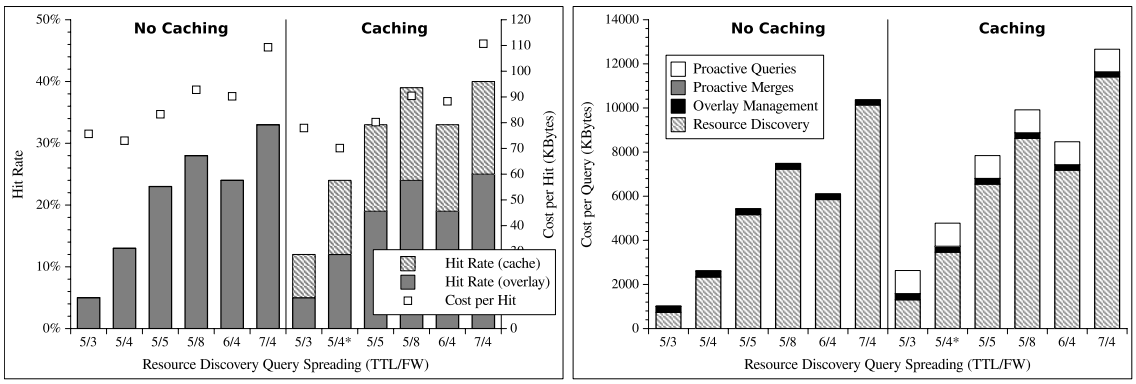
D - Comparison with Newscast and Gnutella overlays Similarly to scenario **A**, the improvements in the hit rate introduced by proactive caching are evaluated in two different overlays, namely Newscast and Gnutella, to assess the influence of the selected overlay on the behavior of our solution.

4.4 Results

Based on the previously discussed evaluation scenarios and having detailed the considered parameters, we present and analyze here the corresponding results, which aim at assessing the efficiency of the proposed resource discovery approach and the sensitivity of the caching algorithm to variation of parameters. In the presented graphs, experiments marked with a $*$ indicate that the baseline experiment's parameters have been used.

4.4.1 A - Benefits of proactive caching

As shown in Figure 4.5 (a) our proactive caching strategy significantly improves the hit rate, which is doubled in the *5/3* (i.e. TTL=5, FW=3) and *5/4* query forwarding strategies, and accounts for about 40% of the hits in other experiments. Cache merges generate a negligible part of the traffic (6 KBytes per query), whereas the impact of proactive queries on the network cost is about 10% (Figure 4.5 (b)), and totals about 1 MByte per query. The traffic generated by resource discovery queries forwarded on the overlay slightly increases when caching is enabled, because of the additional forwarding steps that are performed within the cache. The benefits of proactive caching are noteworthy in the *5/4* experiment, where the hit rate achieved with caching matches that of the *5/5* experiment without caching, but with an overall query cost reduced by 650 KBytes per query.



(a) Hit rate / Cost per hit

(b) Cost per query

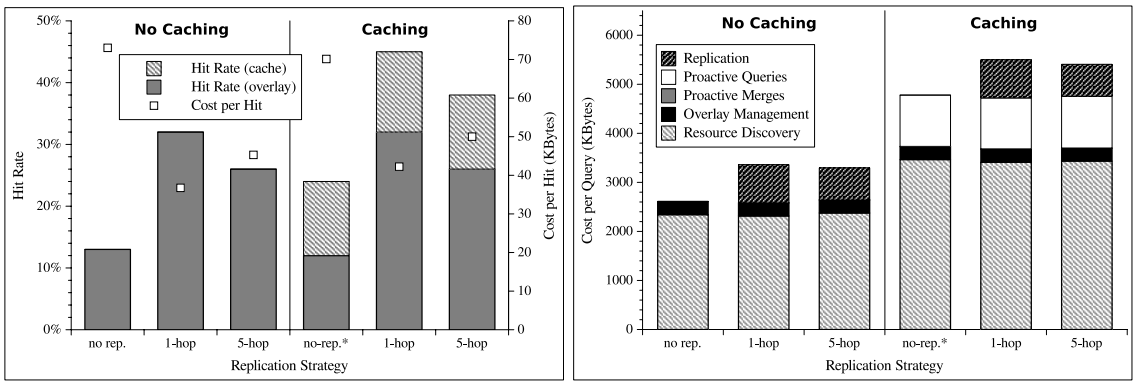
Figure 4.5: A - Benefits of proactive caching

4.4.2 B - Benefits of proactive caching with replication

As shown in Figure 4.6 (a), replication improves the query hit rate by increasing the odds of finding results in the overlay. In this regard, *one-hop* replication provides the best hit rate, both with and without caching. Even with replication, proactive caching improves substantially the performance of resource discovery: in particular, the hit rate increases from 13%, without neither replication nor caching, to 45% when both are employed. As illustrated in Figure 4.6 (b), the cost of proactive caching is comparable to that of replication, and the improved performance accounted to caching remains at the same levels across the different replication strategies. This result enables us to claim that the set of results obtained from the cache overlay and that of results from replicas in the overlay are disjoint, thus the benefits of caching are independent from the replication strategy employed.

4.4.3 C1 - Sensitivity to hops traveled in the cache (C-TTL, C-FW)

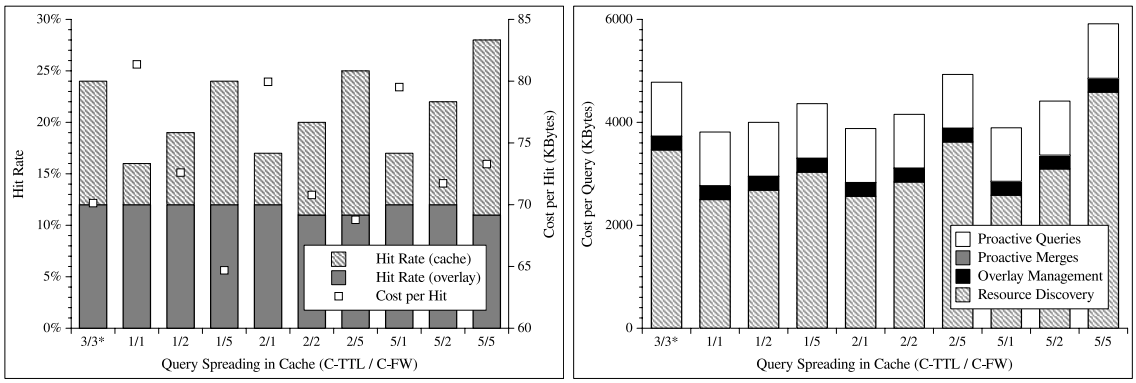
Figure 4.7 (a) shows the impact of different cache navigation strategies (i.e. different C-TTL and C-FW) on the hit rate and the bandwidth required for each result (cost per hit). From the analysis of the results it is clear that the more nodes are visited through the cache, the higher the hit rate becomes. Nonetheless, by focusing on the cost per hit, it emerges



(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.6: B - Benefits of proactive caching with replication

that the best strategy involves forwarding the query for just one hop in the overlay, to all nodes referenced in the cache (*1/5*). This result highlights the fact that similar peers have a higher probability to remain close to a node in the cache overlay, thus the advantages of letting the query be forwarded farther in the cache do not scale proportionally with the distance traveled.

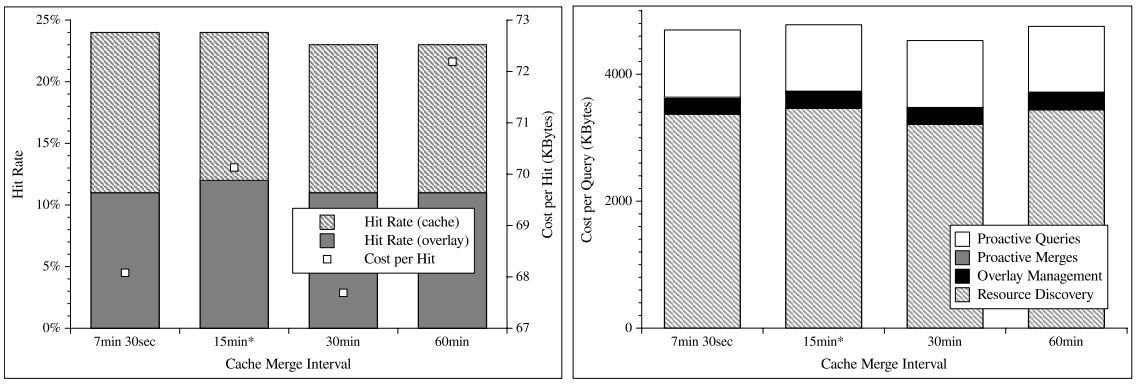


(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.7: C1 - Sensitivity to hops traveled in the cache (C-TTL, C-FW)

4.4.4 C2 - Sensitivity to the cache merge interval (M-Int)

Cache merges allow for increasing the amount of information stored in the caches with a lower bandwidth consumption than proactive queries. With cache merges peers share the discovered resources and remove old entries from the cache, which could point to missing nodes. Nonetheless, as shown in Figure 4.8, changing the merge frequency does not result in significant improvement or degradation of the resource discovery performance, although a slight benefit is observed when the merge interval is below or equal to 15 minutes. As the merging process consumes a negligible amount of bandwidth, more frequent cache merges have no negative impact on the network overhead and should be favored.

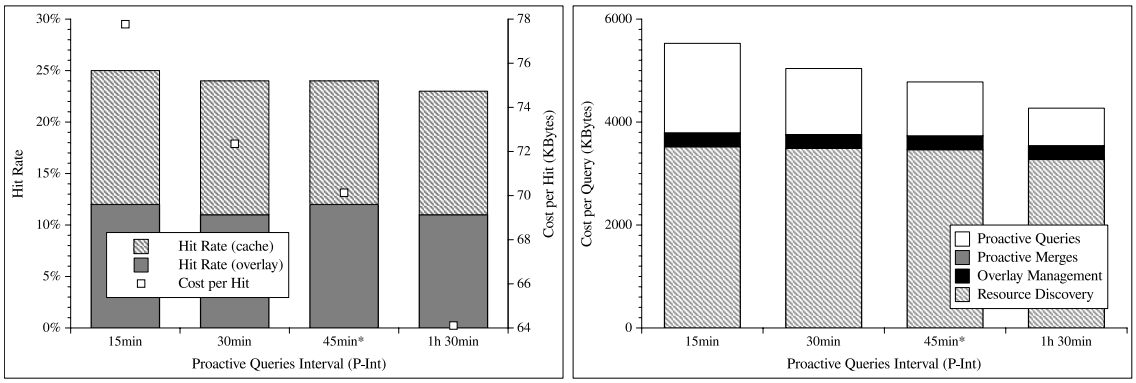


(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.8: C2 - Sensitivity to the cache merge frequency (M-Int)

4.4.5 C3 - Sensitivity to the proactive query interval (P-Int)

As shown in Figure 4.9 (a), more frequent proactive querying increases the hit rate, signaling that better information is stored in the cache. However, a counter-effect of shorter intervals is an increased cost affecting each query. The difference between the strategies concerning the hit rate is nonetheless minimal, varying from 23% when queries are started every 1h 30m to 25% when the interval is reduced to 15m. Pertaining to the network overhead, from Figure 4.9 (b) it is evident that more frequent proactive queries substantially increase the overall traffic.

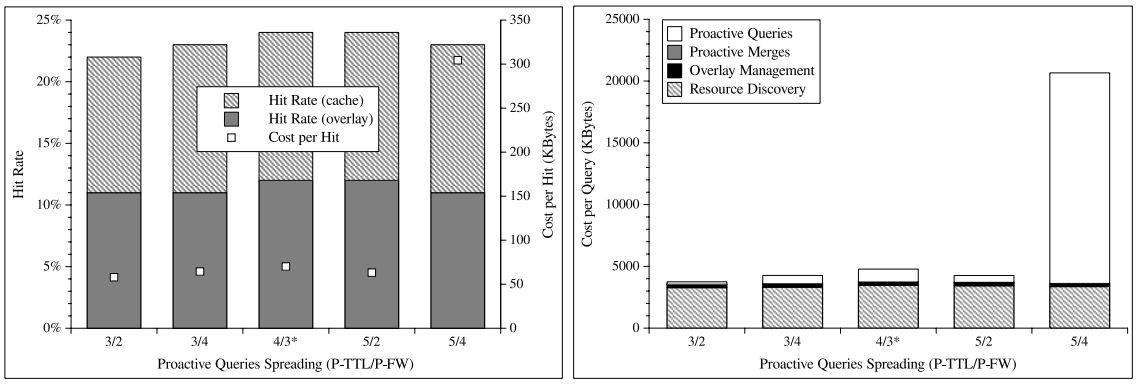


(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.9: C3 - Sensitivity to the proactive query frequency (P-Int)

4.4.6 C4 - Sensitivity to the proactive query spread (P-TTL, P-FW)

By letting proactive queries travel deeper in the network, more hits can be found, hence better cache contents are collected. Results depicted in Figure 4.10 show nonetheless that the small benefits of increased proactive query P-TTL and P-FW do not compensate for the additional bandwidth consumption.

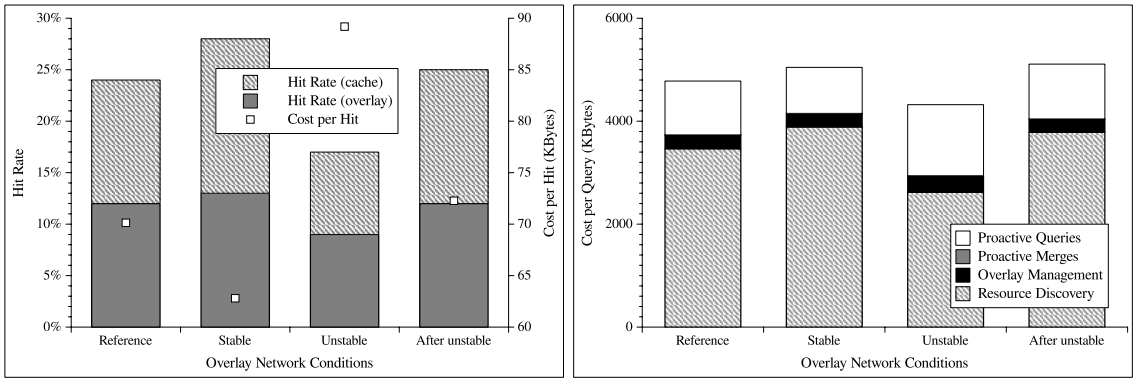


(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.10: C4 - Sensitivity to the proactive query spread (P-TTL,P-FW)

4.4.7 C5 - Sensitivity to network stability

The stability of the network, in terms of frequency of connections and disconnections of nodes, influences significantly the outcome of resource discovery operations. In particular, in an unstable network queries may get discarded if the node that currently processes them disconnects from the network. Additionally, cache contents may refer to nodes that have already left the overlay, which hinders the benefits of cache forwardings. In previous scenarios the network conditions were modified during the simulation, with new nodes joining the overlay every 4 seconds in the period between 6 hours and 9 hours into simulations. In this scenario we consider the three phases of previous experiments separately, namely by analyzing the stable one before 6 hours into the simulation, the unstable one from 6 hours to 9 hours, and the phase after the unstable conditions from 9 hours until the end of the experiment (12 hours). As expected, Figure 4.11 shows that the best hit rate (27%) can be achieved in a stable network; on the contrary, during the dynamic phase of the simulation, unstable network conditions lower the hit rate to 17%, as well as increasing the cost per hit to about 90 KBytes. After the completion of unstable conditions, the performance of resource discovery in terms of hit rate and cost per hit quickly return to satisfactory levels.

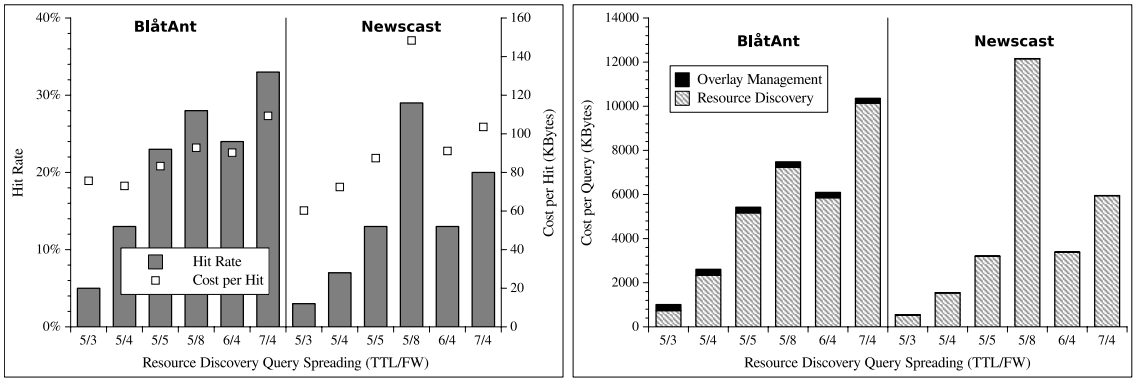


(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.11: C5 - Sensitivity to network stability

4.4.8 D - Comparison with NEWSCAST and GNUTELLA overlays

This last set of experiments aims at judging whether the benefits of the proposed proactive caching scheme can be maintained on other peer-to-peer overlays. The NEWSCAST and GNUTELLA scenarios replicate the same resource discovery settings as the baseline experiment, although with varying forwarding strategies are employed.



(a) Hit rate / Cost per hit

(b) Cost per query

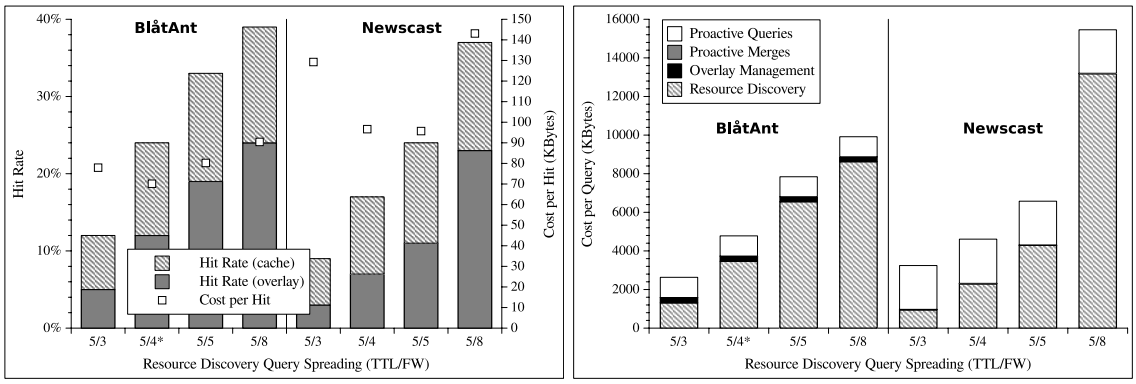
Figure 4.12: D - Comparison with NEWSCAST (without caching)

NEWSCAST During the simulations, the observed average path length in overlays maintained using NEWSCAST was between 5 and 6 during stable network conditions and around 8 in unstable conditions. Although these values are lower than the ones registered for BLÅTANT-S, the hit rate achieved with (Figure 4.13) or without (Figure 4.12) caching is lower than the latter. The reason for these results is the fact that NEWSCAST topologies contain a larger number of links and redundant paths: if a query is forwarded through such paths, nodes that have already been visited are encountered, thus worsening the performance of resource discovery. A similar issue can be observed with proactive caching queries, whose traffic is higher with NEWSCAST than with BLÅTANT-S. However, it is interesting to note that the contribution of the cache mechanism to the hit rate accounts for a similar percentage with both overlays, namely around 7% in the *5/3* experiment and 15% in the remaining experiments.

GNUTELLA As shown in Figure 4.15, the bandwidth required to maintain the GNUTELLA overlay is considerably higher than with BLÅTANT-S. This negatively influences the overall cost of the resource discovery process. Moreover, the hit rate in the GNUTELLA overlay is lower than with BLÅTANT-S in all but the *5/8* scenario, both with or without caching. However, as observed with NEWSCAST, the contribution of the cache mechanism to the hit rate accounts for a similar percentage with both overlays.

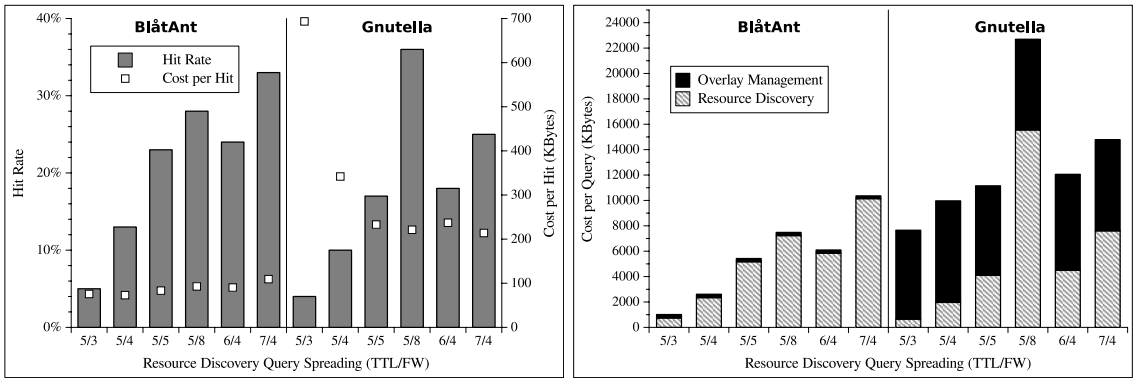
4.5 Accuracy of the results

The presented data refer to an average over 5 simulation runs. For the hit rate and the cost per query, an average over a total of 17150 queries over all runs in each scenario was con-



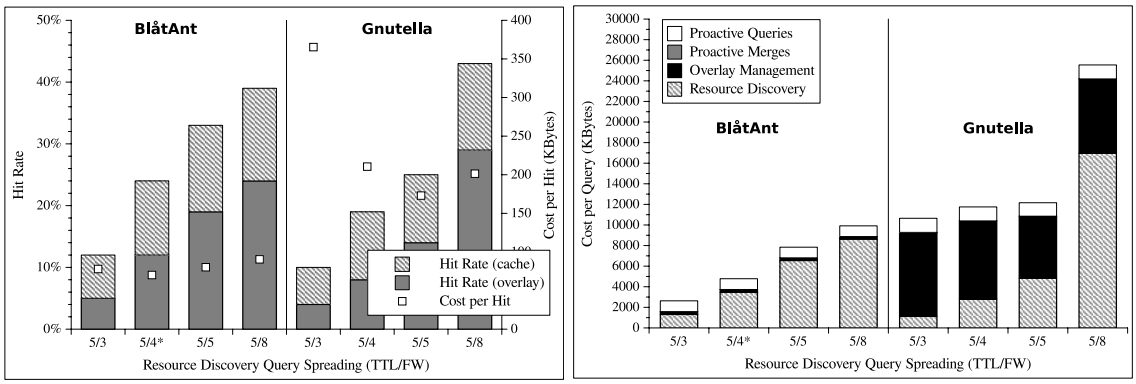
(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.13: D - Comparison with NEWSCAST (with caching)



(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.14: D - Comparison with GNUTELLA (without caching)



(a) Hit rate / Cost per hit (b) Cost per query

Figure 4.15: D - Comparison with GNUTELLA (with caching)

sidered. The observed relative standard deviations are minimal and do not invalidate our findings. More specifically, the average relative standard deviation for the hit rate across all scenarios was 2.52%, and for the hit rate contributed by the cache 1.63%. Concerning

network traffic, the obtained relative standard deviations for the query cost and the hit cost are 1.44% and 2.16% respectively.

4.6 Summary

In this chapter we presented a technique to improve resource discovery in unstructured overlays using local shortcut caches. Caches are maintained by periodically executing proactive resource discovery in order to retrieve identifiers of other nodes with similar resource profiles, that are thus likely to fulfill the same queries. To further improve the performance of our system, while limiting the network bandwidth consumption, we incorporated in our approach epidemic exchange of information between caches. Resource discovery queries are broadcasted on the network using a probabilistic flooding protocol; when a matching node is reached, search continues through cache shortcuts, providing additional results with limited cost. We evaluated our approach through extensive experimentation and assessed its merits compared to traditional flooding methods. We have been able to realize improvements in the hit rate with little impact on the generated traffic. Furthermore, an analysis of the benefits of our scheme on different peer-to-peer overlays proved its independency and validated its applicability on diverse peer-to-peer overlays with consistently substantial improvements of the hit rate.

Chapter 5

Meta-Scheduling

Contents

Previous chapters have dealt with the problem of connecting remote sites by means of a peer-to-peer overlay as well as with information retrieval. Building on the solutions that we have accordingly introduced, in this chapter we consider the problem of efficiently exploiting the computational power of loosely interconnected computers. The term that is used to define such large scale distributed systems devoted to solving massive computing tasks is *grids*. In this context, *grid computing* refers to all activity carried on a grid.

Grid computing leverages the capabilities of a large number of geographically dispersed sites by lowering the barriers of entry for both exploiting and contributing resources [119].

Grids can be used to solve computationally intensive problems that would not be efficiently solved by a single resource, possibly because of time or space limitations. In this respect, a grid ideally provides a flexible infrastructure that scales efficiently as the number of participating resources increases [81]. In contrast to peer-to-peer networks, grids consist of more powerful resources and better connected infrastructures, and rely on persistent management services [116]. A widely accepted notion of the grid considers it as a pool of federated resources within distributed virtual organizations that manage accessing policies and provide transparent on-demand computing [120, 119]. Grids have been successfully deployed in several scientific scenarios [72], and have attracted a noteworthy research stream aimed at improving the underlying infrastructures in terms of accessibility [194], efficiency [125] and reliability [165].

As pointed out in [253], effective grid computing depends on how efficiently tasks are assigned to available resources. Grid schedulers must decide which resources to schedule a job on, based on the available information about their status. In this regard, grid scheduling and allocation strategies must conform to the demands of the users (i.e. QoS agreements such as response time, cost, etc.), and balance them according to the usage policies set by resource providers (i.e. security, execution efficiency, resource utilization, etc.). Grid scheduling complexity is further exacerbated by the fact that there exist two levels of operation, namely *local-scheduling* and *meta-scheduling*. More specifically, local-scheduling is concerned with managing local tasks' execution policies and resources on every computing node, whereas meta-scheduling provides high-level coordination and orchestration between different local schedulers by assigning tasks to the appropriate computing nodes, typically within a virtual organization. From this point of view, in order to achieve optimal scheduling at both levels, the trade-off between fulfilling the requirements set by local usage policies, and by virtual organizations must be addressed. In this regard, meta-scheduling is often hindered by the limited availability of up-to-date information about grid nodes, which can result in less-than-optimal decisions.

To satisfy the aforementioned issues, currently deployed grid infrastructures [118, 243] rely on centralized or hierarchical schemes to support all the activities required to run the grid: resource discovery, resource and data management, meta-scheduling, as well as security services. The business requirements imposed by virtual organizations inherently support such an organizational model, although it is important not to neglect the concrete demand for flexible, autonomic, and self-manageable grids, in order to reduce deployment costs, increase reliability, and meet dynamic users' needs [23].

Grid systems have an inherent heterogeneous, dynamic and distributed nature [37]; as noted in [164], current designs must face several challenges that currently limit the prospects and full benefits of grid computing. Among the concerns highlighted in [164], our research focuses on problems related to schedulers' interoperability and to reliance on centralized meta-scheduling solutions. In this regard, we aim at enabling fully decentralized meta-scheduling to effectively cope with the challenges that raise barriers to a wider adoption of grids. Our vision is supported by the multitude of network applications that have already recognized and exploited the advantages of distributed and decentralized approaches. Recent advances in the underlying network technologies (i.e. ubiquity, link bandwidth, etc.) also contribute towards this direction, and an established shift toward decentralized solutions has been observed also within grid architectures, with the emergence

of decentralized resource discovery mechanisms [142, 259], fully distributed load-balancing solutions [61], and decentralized meta-scheduling algorithms [272].

In agreement with our vision of a grid supported by decentralized services, this chapter presents a fully distributed meta-scheduling protocol, named ARiA, that supports coordination between nodes to enable efficient global dynamic scheduling across multiple sites. The meta-scheduling process is performed *online*, and takes into account the availability of new resources as well as changes in actual allocation policies. Moreover, the proposed approach aims at addressing the scalability and adaptability of grids, to optimally exploit dynamically changing grid resources. Scalability concerns both the size of the grid and the actual load. On one side new grid nodes must seamlessly merge into the grid system; on the other side, jobs must be distributed over all suitable nodes to avoid hot spots, as long as requirements are met. We refer to adaptability as the ability of scheduling and rescheduling tasks according to global or local scheduling policy changes. In this respect, a balance between adaptability and stability is required to avoid coupling situations that have an adverse effect on performance,

5.1 Grid Meta-Scheduling

The benefits of large-scale distributed computing largerly depend on the ability of the grid middleware to manage large sets of heterogeneous resources, and perform optimal task allocation on these resources. From this point of view, in order to meet the expectations users and resource owners, grid scheduling must allocate jobs on the most suitable machines and avoid overloading just a few of the most capable ones. To this extent, meta-scheduling services play an important role that delineate the capabilities and performance of a grid infrastructure.

This chapter focuses on decentralized scheduling mechanisms, namely by enabling fully distributed meta-scheduling across heterogeneous nodes, while additionally providing dynamic load-balancing support by rescheduling jobs across nodes whenever possible. To better understand the issues raised by grid job allocation, in this section we review related work concerning both meta-scheduling and load-balancing, and discuss the transition from centralized approaches toward decentralized ones.

While fully decentralized cooperative grid solutions bear advantages over their centralized counterparts, interoperability of the diverse systems involved is often hindered by infrastructural or organizational problems, such as lack of standardization [112]. Although a number of projects have been started to promote collaboration between projects, and to implement standards for facilitating the communication between different platforms [230, 44], interoperability remains one of the open issues for future generation grids [220, 110]. As discussed in [134], to alleviate these issues, collaborative scheduling solutions should avoid enforcing control over local resources by establishing a clear separation between global and local resource management. Furthermore, resource management and scheduling should rely on adaptive decision-making in order to cope with unprecedented situations. Moreover, meta-scheduling should also be concerned with load balancing through dynamic reallocation of jobs. Unfortunately, whereas task allocation on a single site benefits from the availability of local, complete, and precise information about available resources, decentralized approaches have to tradeoff between information and network traffic.

Centralized Scheduling Traditional grid models [118, 243] rely on centralized or hierarchical meta-schedulers that have a global view of the resources shared on the grid or by their virtual organization. Research has come up with very efficient centralized meta-scheduling mechanisms [17] that can take full advantage of a global view of the grid and provide optimal allocation of tasks on resources. It should be noted that centralized scheduling does not necessarily require a corresponding central information repository, but can rely on distributed information systems [232]. Nonetheless, these approaches still contain bottlenecks for scalability of the system, as well as single points of failure that may affect the robustness of the grid as a whole. An extensive literature review of centralized scheduling mechanisms is outside the scope of this work; an in-depth analysis of related state-of-the-art can be found in [96].

Decentralized Scheduling The design of decentralized and adaptive scheduling algorithms is considered in [256], with nodes performing load-balancing within a limited set of neighbors. Two strategies are proposed, namely transferring jobs at precise intervals or depending on their arrival time; both strategies have the goal of achieving similar total execution time on all nodes. In the direction of reducing the average response time, [112] proposes an adaptive decentralized mechanism that employs an evolutionary fuzzy algorithm to select the best site for job delegation among the set of all possible candidates.

The ORGANIC GRID [68] introduces a novel paradigm that redefines the grid as self-organized biologically inspired complex system of agents providing decentralized scheduling for heterogeneous tasks on a large number of resources. Nodes are organized as a tree, with the root being the job originating node, and faster nodes located closer to it; nodes can push tasks down the tree depending on the actual load of their children.

Collective intelligent behavior of mobile agents has been also exploited in [61] to support grid task load-balancing in a fully distributed environment. Job requirements and resources are profiled using a performance analysis tool called PACE [217], and matched to appropriate resources by the agents. Recognizing the importance of decentralization and self-organization for the future of grid systems, [104] presents a distributed grid scheduling framework where nodes group into communities according to resource similarities and disseminate their actual state. The scheduling process is decentralized and makes use of information about remote nodes in order to find the best resources to fulfill a request.

The distributed meta-scheduling model presented in [272] operates on the principle of submitting a job to the least loaded sites and subsequently revoking it on all but the one that has commenced its execution. An evident drawback of this model is the overloading of a large number of schedulers with jobs that are frequently cancelled. Another decentralized scheduling and load-balancing technique is detailed in [25], which depends on nodes retaining jobs or submitting them to their neighbors according to a heuristic on local load. A different approach is taken in [179], where the selection of a target neighbor for job delegation is driven by the available bandwidth; this is made possible by the adoption of a simplistic model that considers all tasks as identical and focuses on the time required to transfer data.

The potential of applying peer-to-peer technologies to support decentralized grid scheduling is highlighted in [109], with a fully distributed solution where nodes perform a gossip-based exploration of the network for the purpose of generating an optimal schedule on

the discovered resources. Peer-to-peer gossiping protocols are also employed in [103], but with the goal of disseminating the state of the available resources across the grid; this information is cached by remote nodes and used to optimally allocate incoming jobs.

The GridIS [297] scheduling algorithm employs a peer-to-peer communication model that enables resource providers to bid for the delegation of a job. Job requests are submitted to the grid through a portal that broadcasts them in an unstructured peer-to-peer overlay network. The objective of GridIS is to satisfy both resource consumers and providers, by ensuring high successful execution rates, respectively fair allocation of benefits. Similarly to GridIS, the work presented in [99] makes use of a structured peer-to-peer overlay network to discover nodes wishing to carry out a job; furthermore, rescheduling is exploited to avoid starvation of jobs failing to be executed.

In contrast to the aforementioned research approaches, we aim at supporting fully distributed task meta-scheduling by means of a lightweight coordination protocol which takes into account the dynamicity and heterogeneity of resources. Among the distinctive features that differentiate our solution, we highlight the fact that it does not require detailed scheduling information from other nodes, and that it promotes asynchronous peer-to-peer interaction between nodes as well as overall self-organization. In this regard, we assert that our solution contributes to the previously mentioned drive towards flexible and autonomic grids.

5.2 ARiA Protocol

The ARiA protocol [55] aims at providing fully distributed task meta-scheduling across a heterogeneous grid. The name ARiA (*air* in Italian, denoting the aim to be lightweight) comes from the initials of the different message types defined in our protocol, namely REQUEST, ACCEPT, INFORM, and ASSIGN (Table 5.1). An additional STATUS message is employed by the protocol to support synchronized execution of interdependent task pools in advance reservation scheduling. In the following we detail the operational phases of the protocol, as well as the information exchanged between nodes by means of the aforementioned messages.

5.2.1 Assumptions

One of the fundamental design principles of the ARiA protocol is that of being agnostic to schedulers, namely not requiring nor depending on any particular local scheduling policy. Moreover, to emphasize the idea of promoting fully distributed operation, it is assumed that grid nodes are connected by means of a peer-to-peer overlay network. The algorithm nonetheless requires that direct communication between any pair of nodes could be established. According to these premises, and for evaluation purposes, we base our experiments on a self-organized overlay maintained by BlåtAnt-S, as it accounts for a lower bandwidth consumption than the -R version.

ARiA supports all phases of the job execution, from submission to completion, and exploits the time-to-execution to perform dynamic rescheduling of jobs across grid nodes, thus achieving better global throughput and load-balancing. To obtain resources for job delegation, a specific REQUEST message is defined by the protocol: this task can either

be accomplished by processing such message on a suitable grid information system or by broadcasting it on the network using a dedicated fully distributed resource discovery protocol. Because of its fully distributed design, job submission can be performed from any node; furthermore, execution may occur on any node whose resource profile matches the job requirements. For simplicity, we do not allow nodes to decline incoming jobs that have been already accepted, and while every node may hold several jobs within its scheduling queue, only one job at a time can be executed. Batch jobs are assumed to be independent, while advance reservations can be made for jobs composed of interdependent tasks. Moreover, to avoid checkpointing issues, preemption and migration of running jobs are not considered, while security issues are also out of the scope of this research.

To describe resource and job profiles the protocol does not specify any particular formats: actual implementations may choose to use one of the available job description schemas such as JSDL [115]. In accordance with this view, also the matching logic determining whether a task can be executed on a specific node is left to specific implementations, which may choose to define job acceptance rules based not only on profile matching but also according to security, or accounting policies. Finally, execution of tasks and transmission of task-related data between nodes are not within the focus of this research. In this regard, the evaluation provided in the following will assume that jobs are responsible for transferring the required data on the node where execution takes place.

Table 5.1: Protocol Messages and Fields

ACCEPT

Node's address	Job UUID	Cost

REQUEST

Initiator's address	Job UUID	Job Profile

INFORM

Assignee's address	Job UUID	Job Profile	Cost

ASSIGN

Initiator's address	Job UUID	Job Profile

STATUS

Job UUID	Status value

5.2.2 Job Submission Phase

The first phase of the protocol covers the submission of jobs and their initial handling by the node that each job was submitted to. Because the protocol aims at achieving optimal grid-level meta-scheduling, submitting a job to a particular node does not ensure that execution will take place locally, unless such a requirement is specified in the job description.

To univocally track jobs scheduled on the grid, each job is assigned a universal unique identifier (UUID). Nodes receiving job submissions from users or applications are referred to as *initiators* for these jobs. In order to find candidates for the execution of a job, initiators issue resource discovery queries across the grid by means of REQUEST messages. These messages can either be sent to a grid information system or broadcasted to a random

subset of nodes of a peer-to-peer overlay. The submitting node then waits for a predefined timelapse for incoming query replies. When pools of interdependent tasks are submitted to a grid node, each task is independently managed by means of separate REQUEST messages.

Besides the initiator's address and the job UUID, a REQUEST message contains the profile of the resources required to carry out the job, which also specifies an Estimated job Running Time (ERT) according to a grid-level accepted baseline regarding computing power. The estimated running time can be computed by means of a job profiling tool such as PACE [217]. Job profiles may also define additional job execution constraints, for example to prevent execution of a job outside the boundaries of a virtual organization.

5.2.3 Job Acceptance Phase

In a fully distributed implementation, upon reception of a REQUEST message, a node determines whether the requirements of the job profile match its own resources. If the request cannot be satisfied, the message is further forwarded on the peer-to-peer overlay, otherwise a *cost* value for the job based on actual resources and current scheduling is computed. The cost information is sent back to the job's initiator by means of an ACCEPT message. If the REQUEST message is processed by the grid information system, the latter would either reply according to available information or forward the message to each matching node, which would then reply directly to the job initiator.

The cost value depends on the adopted local scheduling state, with lower values being used to indicate better offers. The initiator evaluates incoming ACCEPT responses, and selects the best qualified node (i.e. the node providing the lowest cost). The job is delegated to the latter, which is referred to as the current *assignee*, by sending an ASSIGN message. In order to keep track of the scheduling status of each job, the initiator and the assignee both store a reference to each other: whenever the assignee changes, the initiator is notified.

Currently, three cost functions have been considered, namely *Estimated Time To Completion* (ETTC), *Negative Accumulated Lateness* (NAL) and *Total Delay Time* (TDT) for batch, respectively deadline and advance reservation schedulers. As we assume that deadline scheduling offers and advance reservations ones are not mixed with batch ones, values produced by different functions do not necessarily need to be comparable.

Estimated Time To Completion (ETTC) This function defines the cost for a job j as:

$$\mathrm{ETTC}_{cost}(j) = ETTC_j$$

where $ETTC_j$ corresponds to the relative time that the job is expected to finish according to the local scheduling policy and actual load of the node (determined by the scheduling queue).

Negative Accumulated Lateness (NAL) Targeted at deadline scheduling algorithms, it computes the cost for a job j and an existing local scheduling queue Q as follows:

$$\mathrm{NAL}_{cost}(j) = \sum_{job \in Q'} \delta_{(job,Q')} * |\gamma_{job}|$$

with

$$Q' = Q \cup \{j\}$$

$$\gamma_{job} = deadline_{job} - ETC_{job}$$

$$\delta_{(job,S)} = \begin{cases} -1 & \gamma_{job} \geq 0, \forall\ job \in S, \\ 0 & \gamma_{job} \geq 0 \wedge \exists\ w \in S\ :\ \gamma_w < 0, \\ 1 & \text{otherwise} \end{cases}$$

ETC_{job} refers to the absolute time that the job is expected to finish according to the local scheduling policy and the actual load of the node (determined by the scheduling queue Q'), while $deadline_{job}$ is the upper time limit for job completion; hence γ_{job} represents the lateness of the job. If no deadline is missed, the cost function returns a negative result, with smaller values indicating better scheduling solutions.

Total Delay Time (TDT) This function is used to evaluate the opportunity of allocating a time-slot in advance reservation scheduling. The cost value is determined by the sum of all the estimated delays for scheduled jobs; if the sum is zero, the cost is the negative value of the sum of free time between jobs in order to have schedules with longer idle times represent better choices. For an existing local scheduling queue Q the cost is thus:

$$\mathrm{TDT}_{cost}(Q) = \begin{cases} -idle_Q & \gamma_Q = 0, \\ \gamma_Q & \gamma_Q \neq 0 \end{cases}$$

with

$idle_Q$ = sum of idle time between scheduled jobs

$$\gamma_Q = max(0, \sum_{job \in Q} (EST_{job} - ARS_{job}))$$

EST_{job} = estimated job start according to local schedule

ARS_{job} = advance reservation slot beginning time

Whereas positive values account for the inability of the node to cope with reservations, negative values are an inverse value of the idleness of a node, hence smaller values indicate better scheduling options.

5.2.4 Dynamic Rescheduling Phase

An important aspect of the ARiA protocol is the dynamic rescheduling of jobs. This supports the scalability and adaptability of the meta-scheduling mechanism by enabling job re-allocation to reflect possible changes in the state and availability of resources. This can typically be the result of new nodes connecting to the grid, or existing jobs terminating earlier than predicted or being cancelled.

At the time of job assignments, *assignees* represent the initiators' perceived optimal solutions for job execution; however, it should be expected that better alternatives may potentially arise in the future. Accordingly, the *assignee* attempts to find candidates for rescheduling of jobs in its queue while their execution has not yet started. For this purpose, INFORM messages, which are either processed by a grid information system or disseminated across the network, are employed. Because the rescheduling process is executed periodically, a fully distributed implementation should make use of a low-overhead discovery protocol to avoid excessive bandwidth consumption. In our evaluation, a fully distributed probabilistic flooding protocol is used.

The structure of INFORM messages relates to that of REQUEST messages, in that they both contain a full description of the job's profile. The goal of INFORM messages is to discover nodes that might carry out the execution of the job at a lower cost than the current *assignee*. For this reason, INFORM messages also carry the actual cost value, as computed by the aforementioned cost calculation functions. Nodes will typically generate INFORM messages for a set of jobs in their queue according to a selection mechanism. For batch schedulers jobs with the largest waiting times are preferentially selected, for deadline schedulers jobs with the least lateness are chosen, whereas for advance reservation schedulers tasks with the largest delays are considered.

The behavior of a node upon reception of INFORM messages is similar to the one concerning REQUEST messages, with the node first checking whether it can satisfy the job's requirements and then evaluating the corresponding cost for execution. Unlike REQUEST messages, an ACCEPT reply will only be sent to the current *assignee* if a lower cost can be guaranteed. Thresholds may be introduced to prevent rescheduling when the benefit does not justify the additional overhead, for example if the execution time is only reduced by a small fraction or if the actual job transfer time surpasses the benefit to be gained from the rescheduling operation.

The rescheduling process is completed when the current *assignee* receives the ACCEPT message and accordingly reassigns the job to the new *assignee* by means of an ASSIGN message. To ease tracking of jobs, and enable failsafe mechanisms in the event of an *assignee*'s crash, rescheduling actions are notified to the job's initiator by means of a STATUS message with value SCHEDULED.

5.2.5 Job Execution Phase

The last concern of the protocol is to manage the execution of jobs. Whereas in batch, deadline, and simple advance reservation scheduling each job can be started as soon as it reaches the head of the scheduling queue, the scheduling of pools of dependent tasks is more complicated. More precisely, tasks in each pool depend on each other, and thus need to be concurrently executed. This situation prevents scheduling of multiple dependent tasks in the same job queue if the available resources on the node preclude their parallel execution. Moreover, the meta-scheduling protocol must implement a mechanism to synchronize the start of the execution of each task in a pool.

ARiA deals with this issue by means of STATUS messages: when a job is ready for execution, the initiator is notified with a STATUS message with value READY. The job initiator waits until all tasks in a pool are ready, and then notifies the corresponding

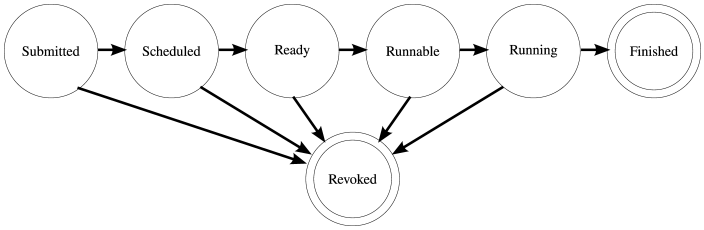


Figure 5.1: Job scheduling and execution states

remote nodes with a RUNNABLE notice. Nodes receiving such notification may change the execution state of the corresponding jobs to RUNNABLE, so that the scheduler can begin their execution, subsequently changing their status to RUNNING. A node can also revoke the execution of a pool, by sending a STATUS message with value REVOKE, to the job initiator, which then relays it to all job tasks' assignees. Jobs are automatically revoked when a node has one task ready for execution and one or more dependent tasks in the same queue that cannot be concurrently started. Figure 5.1 illustrates the job execution states and all possible transitions. To prevent denial of service attacks from a misbehaving node, each node involved in the execution of a task (initiators and assignees) can ask for its revokation, for example if it delays other tasks for a too long period.

5.2.6 Example

To better understand the different phases of the protocol we propose here a simple example of the submission, acceptance, and dynamic rescheduling steps of a single task job. We consider an overlay composed of 13 nodes, depicted in Figure 5.2, and a fully distributed implementation of the protocol.

At step 1, a job is submitted to node A, which becomes the initiator of that job and is responsible for the initial delegation. Next, a resource discovery operation is started by sending REQUEST messages on the network (step 2). All nodes matching the job profile compute the estimated cost according to their scheduling policy, and reply to the initiator with an ACCEPT message: in this example we suppose that replying nodes are B,F, and P (step 3). The lowest cost offer (in this example, the one submitted by node B) is chosen by the initiator at step 4, and the job is assigned by means of a ASSIGN message. The assignee (B) replies with a STATUS message with value SCHEDULED to signal that the job has been correctly scheduled.

Because the resource availability on the network may change, before the start of the execution, B tries to reschedule the job, by searching for lower cost offers: accordingly, INFORM messages are transmitted on the network (step 5). Each receiving node checks whether its resource profile matches the job description and whether it can provide a lower execution cost. If both conditions hold, an ACCEPT message is sent to the current assignee (step 6), and the rescheduling process is concluded by transferring the job to the new assignee (step 7), and notifying the initiator of the change of assignee by means of a STATUS message from Y with value SCHEDULED. As long as execution of the job has not yet commenced, several rescheduling operations can take place.

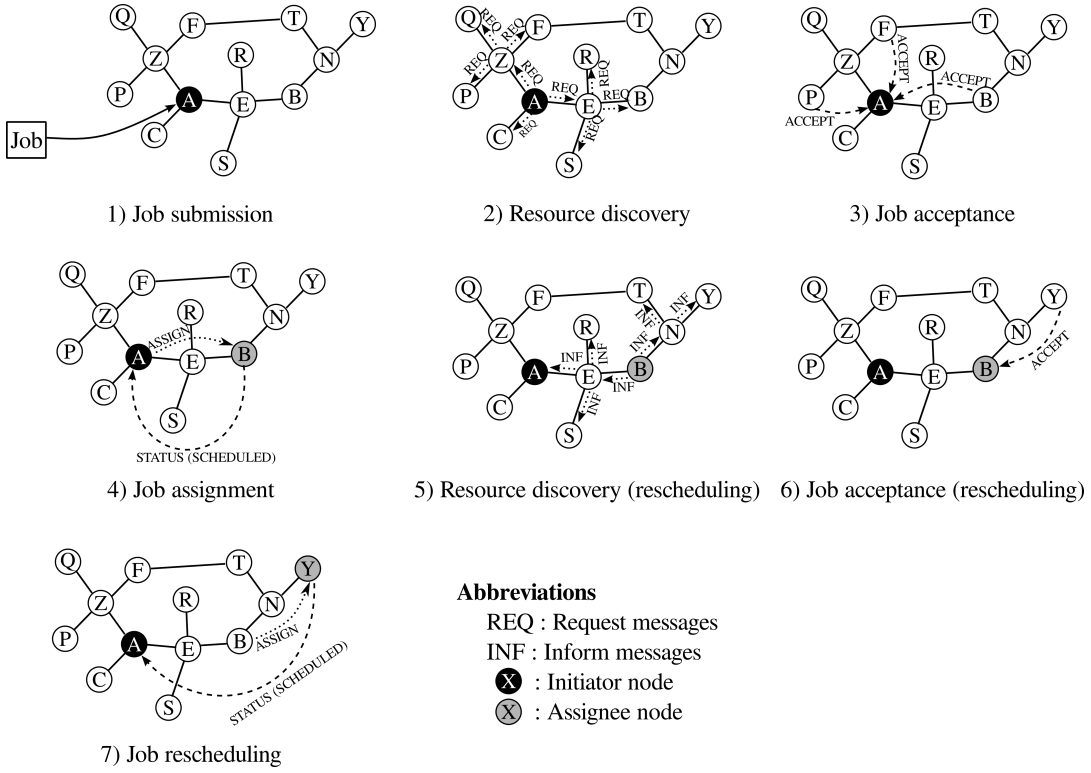


Figure 5.2: Job submission, acceptance and rescheduling example

5.3 Evaluation

To evaluate the behavior of the ARiA protocol in a grid environment, an in depth analysis by means of simulations was performed. To take into account all the characteristics of the meta-scheduling problem, several aspects are considered: scheduling optimality, adaptiveness, scalability, consumed network bandwidth, and load-balancing. More specifically, our evaluation focuses on measuring the average total execution time, the traffic generated by protocol messages, the number of idle nodes, of delayed jobs in advance reservation scheduling, and of missed deadlines. From this point of view, our analysis aims at both assessing the qualities of the dynamic meta-scheduling protocol, as well as providing a sensitivity analysis of the main protocol parameters, in order to understand their influence on the aforementioned assessment metrics. This section introduces the evaluation setup and the details of each of the considered scenarios, a summary of which can be found in Table 5.2.

All scenarios are evaluated on a custom event-based simulation platform, where communication latency between nodes is based on realistic timing as in Chapter 3. For each scenario 5 simulation runs were performed.

5.3.1 Overlay network

We assume that grid nodes are connected by means of an unstructured peer-to-peer overlay, and that nodes trust each other and interact directly among them. Accordingly, we employ

Scenario	Focus of the evaluation
A	Benefits of dynamic rescheduling with batch schedulers
B	Robustness and load balancing capabilities
C	Scalability
D	Benefits of dynamic rescheduling with deadline schedulers
E	Benefits of dynamic rescheduling with advance reservation
F	Sensitivity

Table 5.2: Summary of the meta-scheduling evaluation scenarios

an overlay of 500 nodes constructed and maintained using BLÅTANT-S. The algorithm parameters values are as defined in the baseline scenario presented in Chapter 3, although for our evaluation purposes the network is maintained stable; an exception is the scenario focusing on the scalability, where an expanding overlay growing up to a size of 700 nodes is employed.

5.3.2 Grid resources

Evaluation of the protocol is conducted on an overlay of heterogeneous resources, where the capabilities of each node are determined by its profile. Resource profiles are comprised of different fields that include both hardware and software properties of the machine. Similarly to the evaluation of resource discovery presented in Chapter 4, the following aspects have been considered: the implemented architecture (e.g. AMD64, POWER, etc.), available memory, available disk space, and operating system (e.g. LINUX, SOLARIS, etc.). Upon initialization, the simulator randomly assigns a profile to each node according to a probability distribution defined as follows:

- **Architectures** are chosen according to the list published on the *TOP500 Supercomputing Sites* (www.top500.org) at the time of the writing of this thesis. The probability distribution is as follows: AMD64 87.2%, POWER 11%, IA-64 1.2%, SPARC 0.2%, MIPS 0.2%, NEC 0.2%;
- **Available Memory and Disk Space** are both independently and uniformly chosen as either $1, 2, 4, 8$, or 16 Gigabytes;
- **Operating Systems** installed on each node are based on the aforementioned *TOP500* list, with the following distribution: LINUX 88.6%, SOLARIS 5.8%, UNIX 4.4%, WINDOWS 1%, BSD 0.2%.

To account for heterogeneity in the computational capabilities of each node, each system has an associated real value *performance index* p between 1 and 2, that compares its computing power to a baseline reference. The latter corresponds to the hardware configuration used to calculate the Estimated job Running Time (ERT). The simulator uses this index to derive the Estimated job Running Time on a particular node (that is referred to as ERT^p). More specifically, the ERT^p is defined as the ERT divided by the performance index p.

5.3.3 Grid jobs

The resource requirements for submitted jobs are defined according to the characteristics defined by resource profiles. User submitted jobs are created by means of a random generator, and submitted to random nodes in the overlay that subsequently initiate their delegation by sending REQUEST messages on the network. Each job is characterized by parameters defining the resources required to execute the job. This information is matched against grid resources profiles, and includes the required architecture, memory, disk space, and operating system. The values of each job parameter are randomly chosen. To evaluate the impact of the distributions of requests on the performance of the meta-scheduling mechanism, two probability distributions have been considered: the first one, considers the same probability distribution as used for node profiles, while the second one is based on a uniform probability distribution. Whereas the former distribution is employed in all of our evaluation scenarios, the latter is considered only for a sensitivity analysis. Job descriptors also define an ERT, which is randomly assigned according to a normal distribution $\mathcal{N}(\mu, \sigma)$ with $\mu = 2h30m,\ \sigma = 1h15m$, using a lower bound of $1h$ and an upper bound of $4h$ to avoid extreme cases.

In a real grid, the ERT only provides a rough estimation of the actual job running time. Accordingly, in our simulation each node computes an Actual Running Time (ART) by purposely introducing estimation errors. The ART for a job j (which is unknown until execution completes) on a node with performance index p is derived from ERT, ERT^p, and a relative error ε as follows:

$$ART_{j,\varepsilon} = ERT_j^p + drift_{j,\varepsilon}$$

with

$$drift_{j,\varepsilon} = \mathcal{U}_{[-1,1]} * ERT_j * \varepsilon$$

In our evaluation we assume an accuracy of $\pm 10\%$ of the Estimated job Running Time ($\varepsilon = 0.1$).

Unless otherwise specified, in all scenarios jobs are submitted starting from 20 minutes up until 3 hours 7 minutes into the simulation. A new job is submitted to a random node in the overlay at 10 seconds intervals, resulting in a total of 1000 jobs submitted to the grid in each scenario. For deadline scheduling scenarios, jobs' deadlines are set to an absolute time equal to the current time *plus* their ERT *plus* an additional random interval following the aforementioned normal distribution, with $\mu = 15h,\ \sigma = 7h30m$, hence, the deadline is set 15 hours after the expected absolute completion time. In advance reservation scenarios, the reservation start is set 15 hours after the submission time on average, based on the same distribution as for deadlines. In advance reservation of task pools, each job is composed of a pool of interdependent tasks, the size of which is chosen uniformly at random in the range $[1, 4]$; because of their interdependency, jobs in the same pool can be executed only if simultaneously started.

5.3.4 Traffic Evaluation

To evaluate the amount of bandwidth consumed by the meta-scheduling protocol, the following traffic estimations have been considered for the ARiA messages overhead:

– **REQUEST, INFORM, and ASSIGN**: 5 KBytes;

– **ACCEPT and STATUS**: 1 KByte.

Our evaluation focuses on a fully distributed implementation of the meta-scheduling mechanism, thus REQUEST and INFORM messages are disseminated on the network using a probabilistic flooding protocol. Concerning broadcasting strategies, REQUEST messages are forwarded on the network at a distance of 9 hops, to at most 4 neighbors at each step. Conversely, in scenarios with dynamic rescheduling, INFORM messages are generated for at most 2 scheduled jobs candidate for rescheduling every 5 minutes, and are forwarded at a distance of 8 hops, to at most 2 neighbors at each step. These values are based on the properties the underlying peer-to-peer overlay management algorithm and the parameters set for its construction, and guarantee a near optimal operation without overloading the network.

5.3.5 Local Scheduling Policies

The ARiA protocol aims at providing a meta-scheduling service that is independent of the local scheduling policy implemented by each node. Hence, we assume that different schedulers are available. In our simulations, the scheduling policy is randomly assigned to each node upon creation, and in this respect, the following scheduling policies have been considered:

- **First-Come-First-Served (FCFS)**: incoming jobs are appended to the scheduling queue according to the local arrival time (i.e. reception of an ASSIGN message);
- **Shortest-Job-First (SJF)**: the scheduling order depends on the jobs' ERT, with shorter jobs being executed first;
- **Earliest-Deadline-First (EDF)**: used only for deadline scheduling, this policy prioritizes jobs with an earlier deadline (as specified in their profile);
- **Fair Advance Reservation (FAR)**: used in advance reservation scenarios, this policy enables the allocation of time-slots for executing a task. If collisions between allocations happen, the earliest submitted job is given priority;
- **Fair Pool Advance Reservation (FPAR)**: similar to FAR, this scheduling policy supports dependency between tasks within the same pool. When a job is ready for execution, the assignee informs the initiator by means of a STATUS message with value READY: jobs can be started only when all tasks in the pool are ready for execution (i.e. when the assignees receive a STATUS with value RUNNABLE).

For batch scheduling scenarios FCFS and SJF are used: these schedulers are interoperable because these schedulers share the same cost function (as defined in Section 5.2.3). In our evaluations, we use the term *Mixed* to refer to scenarios where each node is randomly assigned a scheduling policy between FCFS and SJB. Unless otherwise specified, in our batch scheduling experiments the Mixed policy is employed. In deadline scenarios the EDF scheduling policy is employed; conversely, in advance reservation scenarios FAR and FPAR are used.

5.3.6 Scenario details

In the following we detail the goals of each evaluation scenario and the main parameter values used to assess the behavior of the protocol according to these goals. The corresponding results are presented in Section 5.4.

A - Benefits of dynamic rescheduling with batch schedulers To quantify the benefits that can be achieved with dynamic rescheduling, scenarios **A** experiment with different local batch scheduling policies (FCFS, SJF, Mixed) and measure the average waiting and execution times, as well as the number of completed jobs during the simulation.

B - Robustness and load balancing capabilities The robustness of the meta-scheduling protocol is assessed by means of low and high load situations in scenarios **B**. More specifically, in low load situations the job submission rate is halved to one job every 20 seconds, with jobs submitted from 20 minutes to 5 hours 54 minutes into the simulation. Respectively, for high load situations the submission rate is doubled, with one submission every 5 seconds, starting from 20 minutes up to 1 hours 45 minutes into the simulation.

C - Scalability Scenarios **C** gauge the scalability by means of a dynamically expanding network. Starting from the original network of 500 nodes, new nodes are added every 50 seconds starting from 1 hours 23 minutes, increasing its size to 700 nodes at approximately 4 hours 10 minutes into the simulation. These new nodes represent newly available grid resources that can take part in the scheduling and rescheduling process. The evaluation of these scenarios aims at determining the load-balancing effect amongst available resources achievable by means of dynamic job rescheduling.

D - Benefits of rescheduling with deadline schedulers Concerning deadline scenarios, the focus of the evaluation is on the protocol's ability to match jobs' deadlines. Two policies are considered: with the first, deadlines are set 15 hours after the estimated completion time on average; with the second, the available time to complete the job is reduced to 2 hours 30 minutes after the estimated completion time on average. All jobs in these scenarios employ the EDF scheduler.

E - Benefits of rescheduling with advance reservation The benefits of dynamic rescheduling in advance reservation scenarios is assessed by measuring the number of delayed jobs and the average delay. In this regard, both simple (one task) reservations, as well as advance pool reservations are considered. The considered local schedulers are FAR and FPAR, for simple reservations and pool reservations respectively.

F - Sensitivity To better understand the behavior of ARiA under different circumstances and to assess how main variables influence the outcome of the scheduling process, a sensitivity analysis is conducted. More specifically, the following parameters and evaluation conditions are considered:

- **F1 - Sensitivity to ERT precision**: the scheduling and rescheduling decisions depend on the estimated running time of each job. To determine the influence of the accuracy of such an estimation we evaluate the behavior of the protocol by varying the error introduced in the simulation. Two sets of experiments with a Mixed batch scheduling policy are considered. In a first set of experiments, the relative error of the Actual Running Time is increased from ±10% to ±25% ($\varepsilon = 0.25$). Subsequently, we employ an optimistic estimation where the ERT is always lower than the actual time, with $\varepsilon = 0.1$, and $drift_{j,\varepsilon}$ is replaced with $|drift_{j,\varepsilon}|$). Finally, we conduct experiments where the estimation matches the ART ($\varepsilon = 0$).

- **F2 - Sensitivity to jobs candidate for rescheduling**: the number of jobs that each node tries to reschedule at once determines the amount of INFORM messages broadcasted on the network. The goal of this evaluation is to assess the benefits of considering rescheduling for a different number of jobs. In particular, we change the default value of 2 candidate jobs in the queue, to 1 and 4 respectively.

- **F3 - Sensitivity to job submission node**: in all other scenarios jobs are submitted to a random node in the grid, to simulate geographically dispersed users accessing their local grid nodes. To determine if the node that acts as job broker influences the outcome of the scheduling process we simulate a grid where only a single node is responsible for job submission, and compare the results regarding the average completion time and the traffic with ones obtained for our baseline strategy.

- **F4 - Sensitivity to job profiles distribution**: to assess the influence of the job profiles distribution on the scheduling performance we experiment with a uniform distribution instead of the one matching the actual distribution of resources.

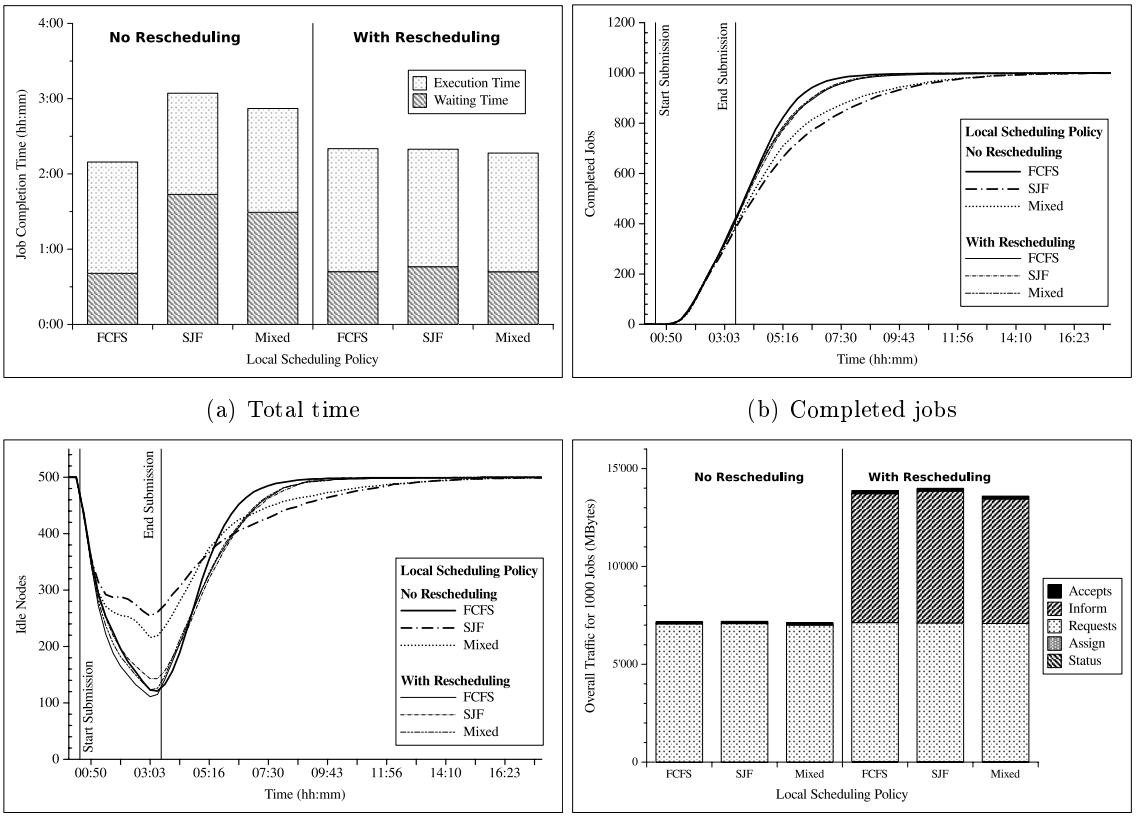
5.4 Results

Having detailed the parameters of the considered evaluation scenarios, we present and discuss here the corresponding results. First, a discussion on the benefits of the dynamic rescheduling mechanism of the ARiA protocol, its scalability, and its effectiveness to address the load-balancing problem is presented. This is followed by an analysis of deadline and advance reservation scheduling scenarios. Finally, the results of the sensitivity analysis pertaining to different aspects of our meta-scheduling approach are discussed. The presented job completion times refer to an average over all 1000 submitted jobs.

5.4.1 A - Benefits of dynamic rescheduling with batch schedulers

Figure 5.3 (a) shows the total execution time achieved on batch schedulers. The SJF and Mixed scenarios demonstrate the benefits of dynamic rescheduling, although it is noteworthy to highlight the comparative optimality of the FCFS policy without dynamic rescheduling. This result is attributed to the fact that FCFS preserves the optimality of the initial delegation by not modifying the scheduling order upon new submissions. On the contrary, with SJF submission of a job with shorter ERT than already scheduled jobs modifies the expected completion time for all jobs with longer ERT. Another interesting

fact concerns the composition of the total job completion time of SJF and Mixed; more specifically, while dynamic scenarios exhibit larger execution times, there is a reduction in the completion time, which proves the effectiveness of the rescheduling phase in providing shorter waiting times and its ability to distribute jobs to nodes based on actual waiting queues length rather than just on computational power. Similar observations about the benefits of dynamic rescheduling with SJF and Mixed policies can be made with regards to the evolution of completed jobs, shown in Figure 5.3 (b).



(a) Total time

(b) Completed jobs

(c) Idle nodes

(d) Traffic

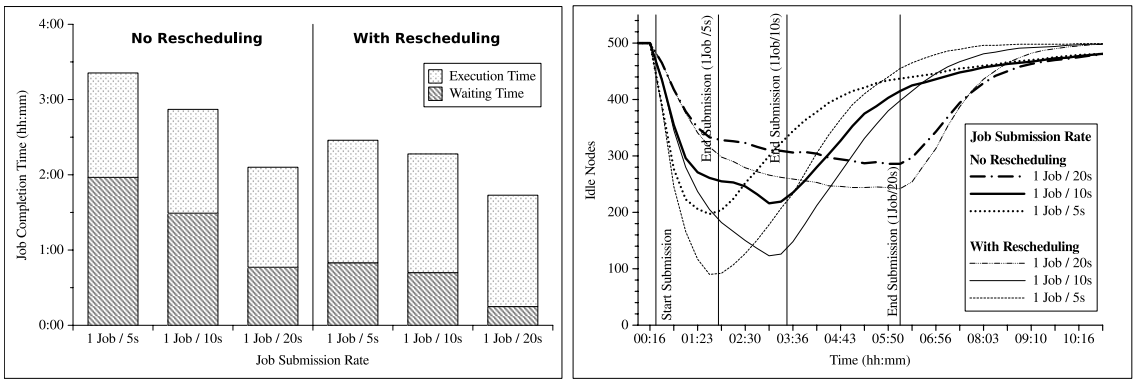
Figure 5.3: A - Benefits of dynamic rescheduling with batch schedulers

As illustrated in Figure 5.3 (c), dynamic rescheduling helps achieving better resource utilization when using either the SJF or Mixed scheduling policies. In particular, the number of idle nodes decreases by about 100, indicating an improved balancing of the overall grid load.

The resulting network overhead is shown in Figure 5.3 (d): for all considered scheduling policies the rescheduling operations double the traffic, from an average of 7000 MBytes to 14000 MBytes. The largest part of the traffic is attributed to REQUEST and INFORM messages, whereas other messages account for only a negligible part of the overall traffic. Although the traffic increase is important, it is compensated by the achievable benefits of reduced execution times.

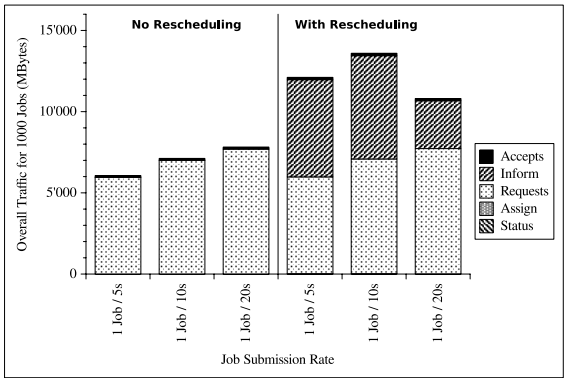
5.4.2 B - Robustness and load balancing capabilities

The robustness of our meta-scheduling protocol in respect to the total execution time *versus* the frequency of job submissions is demonstrated in Figure 5.4 (a). Even when the submission rate is doubled from 1 job every 10 seconds (as in other experiments) to 1 job every 5 seconds, the benefits of dynamic rescheduling are noticeable, with a reduction of the average completion time from 3h 21m to 2h 27m. Conversely, with a slower submission rate of 1 job every 20 seconds, dynamic rescheduling lowers the average completion time from 2h 06m to 1h 43m.



(a) Total time

(b) Idle nodes



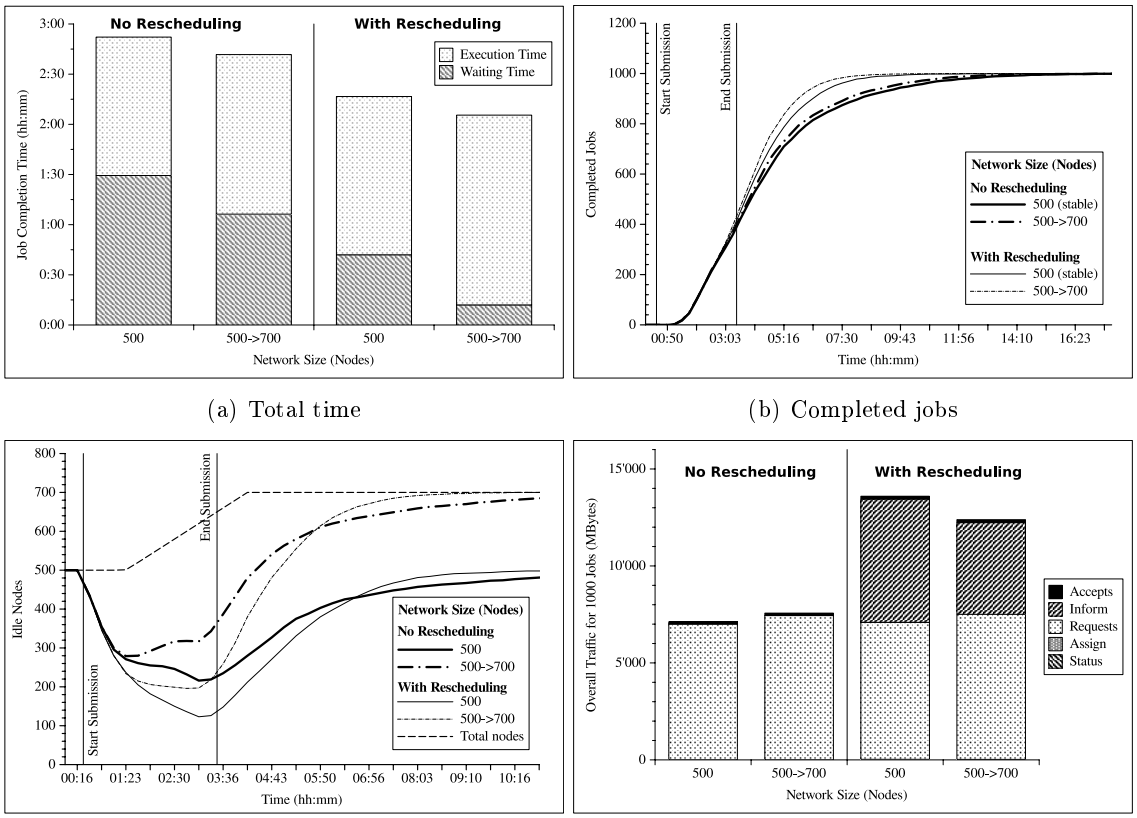
(c) Traffic

Figure 5.4: B - Robustness and load balancing capabilities

Figure 5.4 (b) depicts the resource utilization in all experiments. As noted in scenarios **A**, dynamic rescheduling enhances the load balancing across grid nodes by making use of about 100 nodes more. Finally, Figure 5.4 (c) shows the overall bandwidth consumption necessary to execute the 1000 jobs submitted to the grid. It is interesting to note that a lower submission rate results in noticeably less rescheduling traffic (INFORM messages). The reason behind this is the ability of starting job execution earlier because queues are less loaded as more time passes between each submission; hence, the number of jobs candidate for rescheduling is lowered.

5.4.3 C - Scalability

In Figure 5.5 (a) we assess the scalability of ARiA pertaining to the reduction of the total execution time in an expanding grid. As expected, dynamic rescheduling enables better usage of newly available resources, and reduces the total execution time from 2h 41m to 2h 5m. As it emerged in scenarios **A**, the reduction of the waiting time accounts for a shorter total completion time, although the execution time increases. This result is supported by the analysis of the evolution of the number of completed jobs shown in Figure 5.5 (b).



(a) Total time

(b) Completed jobs

(c) Idle nodes

(d) Traffic

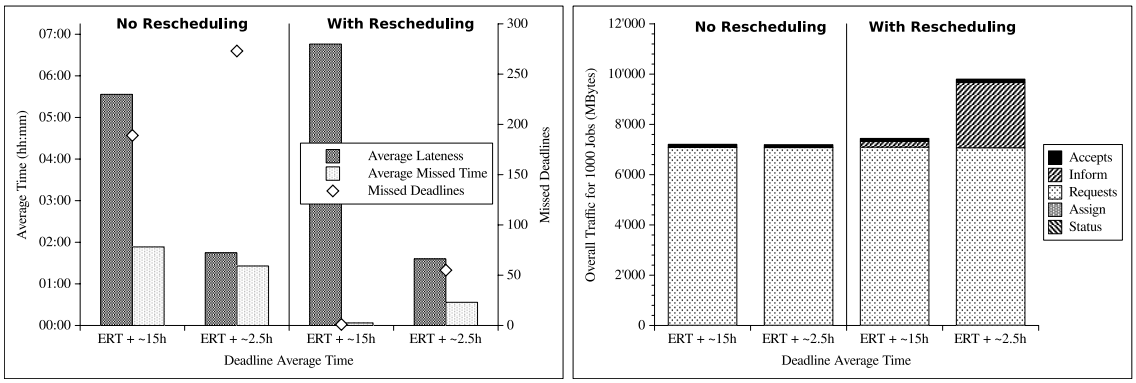
Figure 5.5: C - Scalability

The load balancing effect is demonstrated in Figure 5.5 (c): as the size of the network increases, rescheduling leads to the utilization of up to 100 additional nodes. The network traffic results shown in Figure 5.5 (d) reveal an interesting behavior of the meta-scheduling protocol: as the network size is increased, the overall traffic generated by INFORM messages is reduced. The reason for this is the increased availability of nodes that can start job execution sooner, thus reducing the number of rescheduling opportunities.

5.4.4 D - Benefits of rescheduling with deadline schedulers

Pertaining to deadline scheduling, important performance metrics are the number of deadlines, the lateness (i.e the time left from completion to the deadline), and the missed time (i.e. the time, if any, past the deadline). As shown in Figure 5.6 (a), dynamic rescheduling

significantly reduces the occurrence of missed deadlines. In particular, their number is decreased from 189 to 1 when the deadline is set 15 hours after the estimated completion time, and from 273 to 55 when the deadline is set 2.5 hours after the estimated completion time. With deadlines set at *ERT + 15h*, the average total lateness is also increased from 5h 33m to 6h 45m, meaning that more time is left between completion times and deadlines; with deadlines at *ERT + 2.5h* a slight decrease can be observed, from 1h 44m to 1h 36h, but the total lateness for successful jobs increases as more deadlines are fulfilled. In all experiments, the average missed time is decreased substantially when dynamic rescheduling is employed, going from 1h 53m to 3m with *ERT + 15h*, and from 1h 25m to 33m with *ERT + 2.5h*. Finally, Figure 5.6 (b) shows a significant increase in the number of INFORM messages as tighter deadlines are enforced.



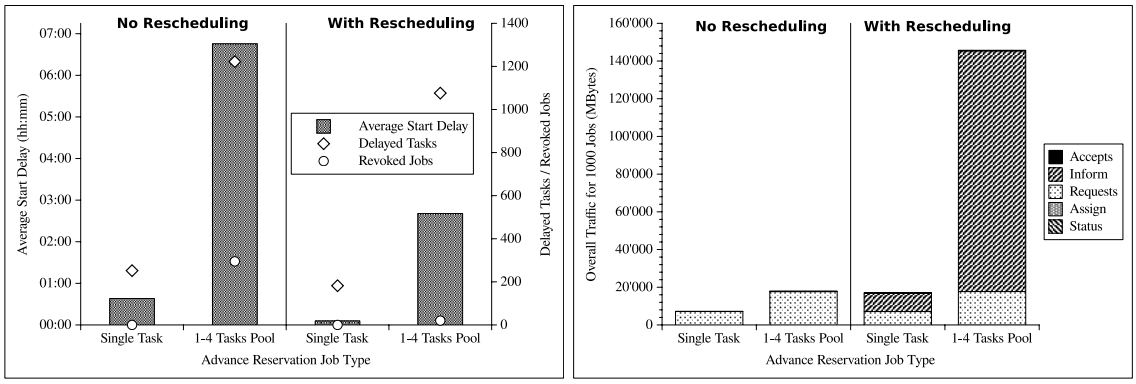
(a) Deadlines (b) Traffic

Figure 5.6: D - Benefits of rescheduling with deadline schedulers

5.4.5 E - Benefits of rescheduling with advance reservation

In advance reservation scheduling the system must ensure that allocated time slots are enforced, and that jobs can start executing on time. However, when multiple reservations are made, collisions may happen and some reservations might need to be delayed. Because our protocol strives to provide a *best effort* meta-scheduling service, an important metric to assess its benefits is the average delay time across all reservation slots. In contrast to deadline scheduling, the execution of a job cannot be arbitrarily started, but has to wait until the reservation start time; additionally, with task pools, each task has to wait until all dependent tasks are ready. Each one of the 1000 submitted jobs is composed of 1 to 4 dependent tasks; in each simulation run a total of 2495 tasks on average was scheduled on the grid. Because each task is scheduled independently, dynamic rescheduling is performed on a per-task basis rather than per-job.

As shown in Figure 5.7 (a) the average delay is significantly reduced when dynamic rescheduling is employed, from 38m to 5m per job in single task reservations, and from 6h 45m to 2h 40m per task in task pool reservations. Moreover, concerning the scheduling of task pools, a substantial decrease in the number of revoked jobs, that are reduced from 295 down to 20, can be observed. Concerning network overhead, Figure 5.7 (b) highlights the significant increase in the bandwidth consumed by INFORM messages, which bespeaks



(a) Delays (b) Traffic

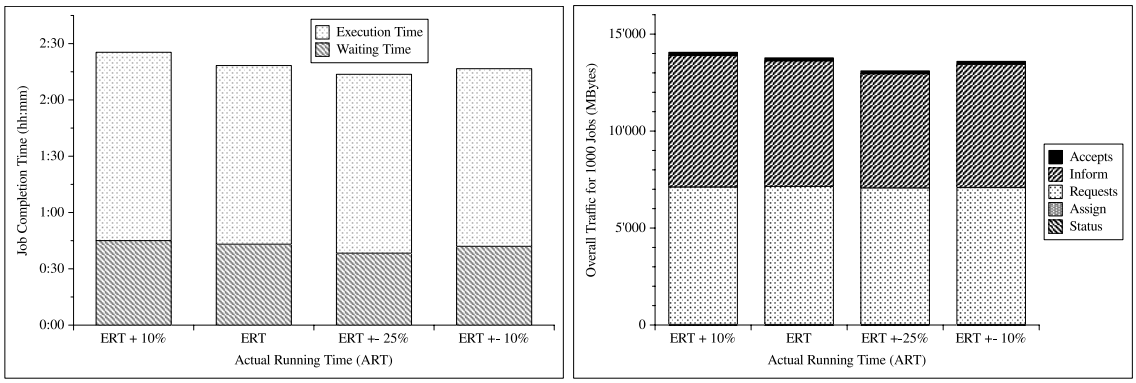
Figure 5.7: E - Benefits of rescheduling with advance reservation

for a more unstable behavior of the protocol in the considered conditions.

5.4.6 F - Sensitivity

Results of the sensitivity analysis will lead to a better understanding of the meta-scheduling protocol and lay more solid foundations for further research.

F1 - Sensitivity to ERT precision An important assumption of the protocol is the availability of an accurate job running time estimation. Because of estimation errors, the actual running time might be higher or lower than the ERT. The results shown in Figure 5.8 (a) provide an insight on the performance of the dynamic rescheduling mechanism implemented by ARiA. The balanced nature of the introduced error accounts for the homogeneity of the average completion time across all experiments. The stability of the protocol is confirmed by the overall traffic generated during the experiments (Figure 5.8 (b)), which remains stable and consistent across all simulations.

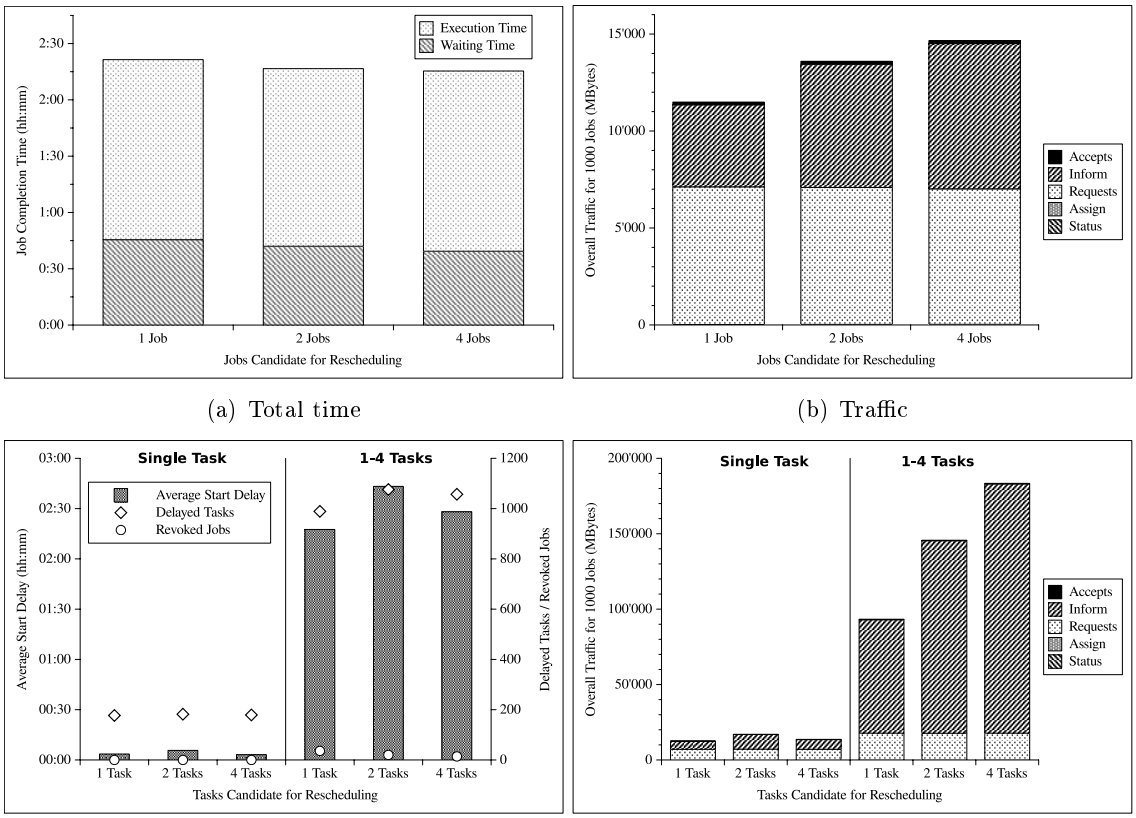


(a) Total time (b) Traffic

Figure 5.8: F1 - Sensitivity to ERT precision

F2 - Sensitivity to jobs candidate for rescheduling The behavior of the dynamic rescheduling phase is determined by the number of jobs that could potentially be reassigned. In this regard, we are interested in evaluating the impact of different rescheduling strategies with two relevant scheduling policies, namely batch and advance reservation scheduling. More precisely, we consider the influence of a different number of jobs candidate for rescheduling on the average job completion time and on the number of delays, for batch and advance reservation schedulers respectively.

The results depicted in Figure 5.9 show no noteworthy variation in the total completion time when batch schedulers are concerned, and negligible differences with single task advance reservation scheduling. On the contrary, with task pool reservations rescheduling of 1 and 4 tasks achieves a lower average job start delay than the default policy of 2 tasks. In this context, rescheduling 1 task seems to provide the best performance, although the number of revoked jobs is slightly increased.



(a) Total time

(b) Traffic

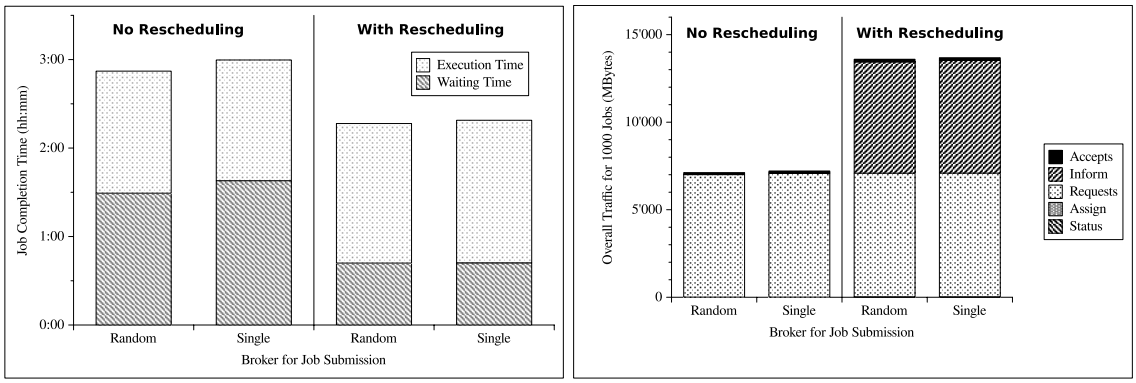
(c) Delays

(d) Traffic

Figure 5.9: F2 - Sensitivity to jobs candidate for rescheduling

Concerning the network overhead, the bandwidth consumption to be accounted to INFORM messages significantly increases as more jobs are considered for the rescheduling phase. From this point of view, selection of 1 candidate emerges as the best option in all scheduling policies.

F3 - Sensitivity to job submission node In all previous scenarios jobs are submitted to a random node in the grid, thus favoring an even distribution of the requests across all available resources. In order to assess the influence of this choice on the performance of the meta-scheduling protocol, experiments where all 1000 jobs are submitted to only one single broker were conducted. Concerning the average total completion time, Figure 5.10 (a), highlights no significant difference between the two submission strategies when rescheduling is enabled, and only a small increase is noticeable when no rescheduling is allowed. On the contrary, an analysis of the traffic, illustrated in Figure 5.10 (b), shows equivalent results.



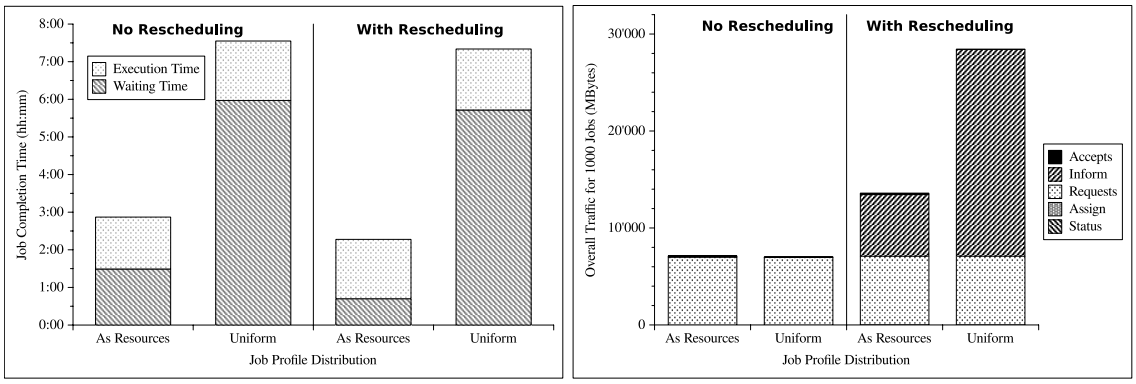
(a) Total time (b) Traffic

Figure 5.10: F3 - Sensitivity to job submission node

F4 - Sensitivity to job profiles distribution The last set of experiments focuses on the distribution of job profiles. Instead of generating job requests according to the simulated distribution of resources, experiments with uniformly distributed requests have been performed. The results concerning the average total completion time are illustrated in Figure 5.11. It is evident that a uniform distribution worsens the performance of the meta-scheduling process, and almost denies all benefits of the dynamic rescheduling phase. This degradation of the performance can be attributed to the large number of jobs that require very rare resources: in this case, the queues on nodes sharing such resources quickly becomes overloaded, leading to substantial increase in waiting times.

5.5 Accuracy of the results

Results detailed in this chapter represent an average over 5 simulation runs for each scenario. Time for completion graphs are computed on an average on 1000 jobs in each run. The obtained performance data has proven to be very stable, with minimal variations across all runs and scenarios. Concerning the average total time for completing a job, the relative standard deviation is 2.6%; conversely, for the average waiting time it is 5.98%, and for the average execution time 1.48%. Pertaining to deadline scheduling, the most relevant relative variations of missed deadlines were found in scenario **D** with dynamic rescheduling enabled, with a deviation of 97% on an average of 1 job with $ERT + 15h$,



(a) Total time (b) Traffic

Figure 5.11: F4 - Sensitivity to job profiles distribution

and of 43% on an average of 55 jobs with $ERT + 2.5h$; without dynamic rescheduling the obtained variations were in the range of 13.10%, on an average of 189 jobs with $ERT + 15h$, and 5.8% on an average of 273 jobs with $ERT + 2.5h$.

5.6 Summary

In this chapter, we focused on the problem of efficiently allocate tasks across geographically distributed resources. We presented a fully distributed grid meta-scheduling protocol named ARiA that aims at improving the efficiency of heterogeneous grids, as well as addressing the related scalability and adaptability concerns. From this point of view, our work reflects the vision of next-generation grids that strive to evolve into reliable, flexible, autonomic, and self-manageable systems that require minimal user intervention and reduced deployment costs.

The proposed meta-scheduling protocol, named ARiA, is based on simple messages exchanged between grid nodes over a peer-to-peer overlay, and does not depend on the actual implemented local scheduling policies. This enables better integration with existing grid middlewares and facilitates its adoption. The central point of our work is the support for dynamic rescheduling of jobs, which enables optimal job reallocation under dynamic conditions, for example by making use of newly available resources and by taking into account changes in resource utilization.

Throughout extensive experimental evaluation, we validated the behavior of the protocol and assessed significant results concerning the effectiveness of our approach. In particular, we achieved shorter average execution times with batch schedulers, a reduced number of missed deadlines, and decreased delays in advance reservation scheduling. The protocol also demonstrated its ability to enhance load-balancing amongst the nodes. Finally a traffic analysis pinpointed an acceptable bandwidth consumption when compared to the acquired benefits, thus suggesting the viability of our approach in real-world deployments.

Chapter 6

SmartGRID

Contents

SmartGRID is a novel grid middleware that aims at supporting grid applications running over a set of non-dedicated resources. From this point of view, SmartGRID follows the idea of implementing a desktop grid that brings together the power of a large number of personal systems voluntarily shared by their users to provide on-demand access to computing resources. In contrast to traditional grid infrastructures, the SmartGRID vision is geared toward peer-to-peer interaction between systems, and promotes self-organization and adaptiveness.

To abstract from the volatile and unreliable nature of the considered underlying resources, a multi-layered architecture has been considered. More specifically two main concerns have been identified, namely that of resource monitoring and of high-level task management. Accordingly the middleware is composed of two loosely coupled functional layers, the Smart Signaling Layer (SSL) and the Smart Resource Management Layer (SRML), which are connected by means of a Datawarehouse Interface (DWI). This design promotes a clear separation between low-level communication amongst peers and high-level task allocation activities.

SmartGRID also differs from common grid middleware platforms, such as Globus [117], in that it operates in a fully distributed and self-organized way, hence lowering the need for supervised operation and reducing deployment effort. Moreover, for low-level activities, such as resource discovery and communication, bio-inspired solutions are employed in order to achieve the required characteristics of adaptiveness, robustness and self-organization.

The work presented in this thesis mainly focuses on the SSL; in particular, the overlay management algorithm and the resource discovery mechanisms are employed to provide basic services to the grid and enable sharing of resources among nodes. In addition, the meta-scheduling protocol concerns the SRML. In the following we thus group all the

components presented in previous chapters and describe how they are integrated within the SmartGRID framework. In section 6.1 the details of the multi-layered architecture are presented, while in section 6.2 the platform used for the SSL is introduced and a discussion about its strengths is presented.

6.1 SmartGRID

SmartGRID is a distributed grid middleware that aims at providing stable, robust, and efficient resource management over a heterogeneous and volatile pool of geographically sparse resources. The architecture is composed of three layers (Figure 6.1): the Smart Resource Management Layer (SRML), the Smart Signaling Layer (SSL), and a datawarehouse interface for loosely coupled interaction. The SRML is in charge of managing user requests and job scheduling by exploiting information gathered from the SSL, which provides resource discovery and low-level communication between nodes.

In the considered scenario, each SmartGRID node runs an instance of both the SRML and the SSL. Concerning the software aspect, the SRML is implemented by MaGate nodes [149, 148], whereas the SSL is based on Solenopsis nodes [52]. The design of the node is modular, to allow for easy replacement of single components and reuse of existing modules for different purposes.

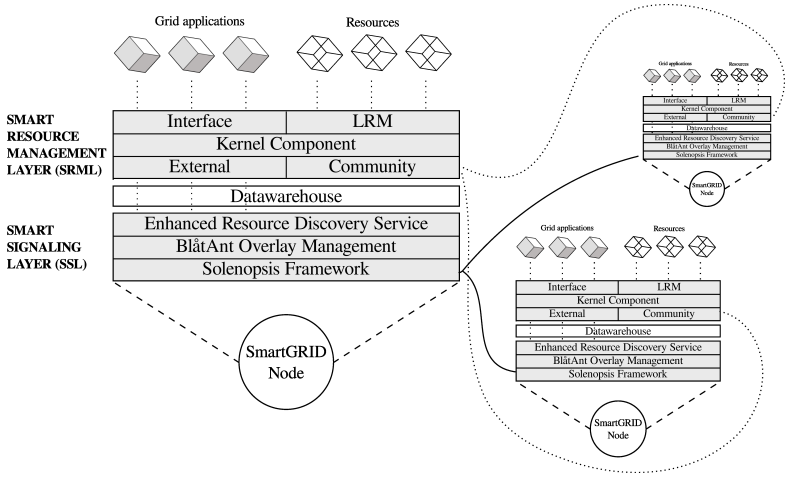


Figure 6.1: SmartGRID node architecture

Smart Signaling Layer The Smart Signaling Layer (SSL) manages low-level communication between grid nodes as well as providing resource discovery. In this context, several of the components presented in this thesis have been integrated in the SSL, as shown in Figure 6.1. In particular to manage the overlay connecting SmartGRID nodes, the BlåtAnt algorithm is employed. To support this functionality from a practical point of view, a custom runtime platform tailored for the deployment of bio-inspired ant algorithms called Solenopsis (presented in detail in the forthcoming section) has been developed. Finally, to support efficient resource discovery the approach presented in Chapter 4 has been employed.

Smart Resource Management Layer The Smart Resource Management Layer (SRML) is in charge of supervising the usage of resources and mediating interaction between the user and the system by providing an interface for task submission and tracking. The SRML exploits information from the SSL to efficiently schedule tasks either on local resources or on remote nodes. Accordingly, the SRML interoperates with the existing scheduling infrastructure and obeys to local and remote resource usage policies. At the SRML level, each node is managed by a software application called MAGATE [148]. As shown in Figure 6.1, the MAGATE itself is comprised of different components that enable interaction with local resources, remote MAGATE, external services as well as users and applications. In the following, a brief review of each component is provided.

- **Kernel Component**: represents the core of the MAGATE, and provides the logic to analyze job descriptions, monitor system load, take scheduling decisions, and coordinate the operation of the other components.
- **Local Resource Management (LRM) Component**: connects to low-level resource management systems and middlewares such as GLOBUS [117] or UNICORE [21].
- **External Component**: offers a plug-in mechanism that enables the integration of additional services, such as resource discovery or hardware monitoring. In the context of SMARTGRID, the Datawarehouse is interfaced by means of an external component.
- **Interface Component**: deals with job submissions from different sources, such as grid users and applications.
- **Community Component**: manages connections between MAGATES, in order to support external job scheduling requests and job transfer requests.

Datawarehouse The SSL and SRML communicate through a datawarehouse, which provides both an asynchronous communication channel and a temporary storage. In the context of the SMARTGRID middleware, the datawarehouse also helps maintaining clear separation of concerns between the two functional layers.

Although the main contribution of this thesis fall within the Smart Signaling Layer, research spans over all layers. In particular, the overlay management algorithm introduced in Chapter 3 and the resource discovery protocol in Chapter 4 concern the SSL, while the meta-scheduling framework presented in Chapter 5 concerns the SRML.

6.2 SOLENOPSIS

SOLENOPSIS [1] [52] is a framework for the deployment of fully distributed ant algorithms composed of a programming language and an execution environment. The framework

[1] Solenopsis Invicta, also known as Red imported fire ant, is a particularly aggressive species of ant originally from South America.

was developed specifically for the SmartGRID project, in order to fulfill the need for a software platform to base the Smart Signaling Layer on.

6.2.1 Related Work

Several platforms exist aimed at supporting the development of ant algorithms; noteworthy examples are the Swarm Simulation System [209], MASS [146] and Anthill [27]. In this section we briefly review these systems and highlight their characteristics:

Swarm Simulation System The Swarm Simulation System [209] allows to model multi-agent discrete simulations at different levels. The framework is object-oriented, with agents being mapped as objects. Agents can interact with each other, and the whole simulation can be synchronized. The platform itself offers different tools for algorithm profiling and data analysis. Unfortunately, the platform is limited to simulations and does not provide any support for fully distributed agent deployment.

Multi-agent System Simulation Framework The Multi-agent System Simulation Framework (MASS) [146] allows accurate and controllable simulations of systems composed of collaborative agents. Agents can sense the simulated environment and perform a mixture of real and simulated activities. The platform supports both discrete time and event-based simulations, but is not targeted to fully distributed deployments.

Anthill Anthill [27] is a Java framework that supports P2P application development. It provides runtime and simulation environments and it has been successfully used to implement the Messor [210] load-balancing algorithm. The runtime environment is a middleware built on JXTA [131] and allows for real-world deployment of applications. A simulation environment is also supported and enables local testing and evaluation of ant algorithms. Unfortunately the development of Anthill was stopped in year 2002.

Because both Swarm and MASS focus on the development and evaluation of multi-agent coordination in distributed systems with total accuracy, by means of a simulation environment, their architecture is not well suited for real-world distributed dynamic environments. In contrast, Anthill is not only aimed at supporting the design and analysis of P2P systems, but at the implementation of such systems in real network environments as well. To such an extent the Anthill framework is the one mostly similar to Solenopsis, although it does not support transparent and strong migration of agents.

6.2.2 Solenopsis Framework Overview

Solenopsis is comprised of several components that support both fully distributed execution (*deployment mode*), with an instance of the platform running on each of the participating hosts, as well as local execution (or simulation) of several nodes (*simulation mode*). It is noteworthy to mention that for the development of ant algorithms, there is no distinction between these two scenarios: implemented algorithms can be executed either in simulation or deployment mode, without modification; moreover, simulated nodes can

be seamlessly combined with deployed ones in a fully distributed environment to enable complex evaluation scenarios.

Deployment mode Figure 6.2 depicts the architecture of a node in a fully distributed setup. On start-up, a **configuration script** is processed by the **platform daemon**. The script contains the list of operations required to set up the node and initialize the required plug-ins, and is executed in the platform's *shell*. As a result of this phase a **node daemon** is instantiated, along with all the plug-ins needed for implementing the extended functionalities of the node (for example, the BLÅTANT algorithm) as well as for exposing access to external resources (such as the Datawarehouse). To enable communication between nodes, the platform daemon provides a **mail server** (that implements a custom protocol based on TCP/IP communication): the server allocates a uniquely identified mailbox to the node daemon which is used to send and receive ants from, respectively to, other SOLENOPSIS nodes. Actual data transfer is managed by the **mail service**. Incoming ants and locally started ants are compiled and then executed in sandboxed **virtual machines**, the execution of which is managed by a **preemptive scheduler**. The scheduler enables concurrent execution of many virtual machines without creating an excessive number of threads. When an ant requests to migrate to another node, the built-in **migration service** serializes the ant's state and transfers it to the target node, where execution is resumed.

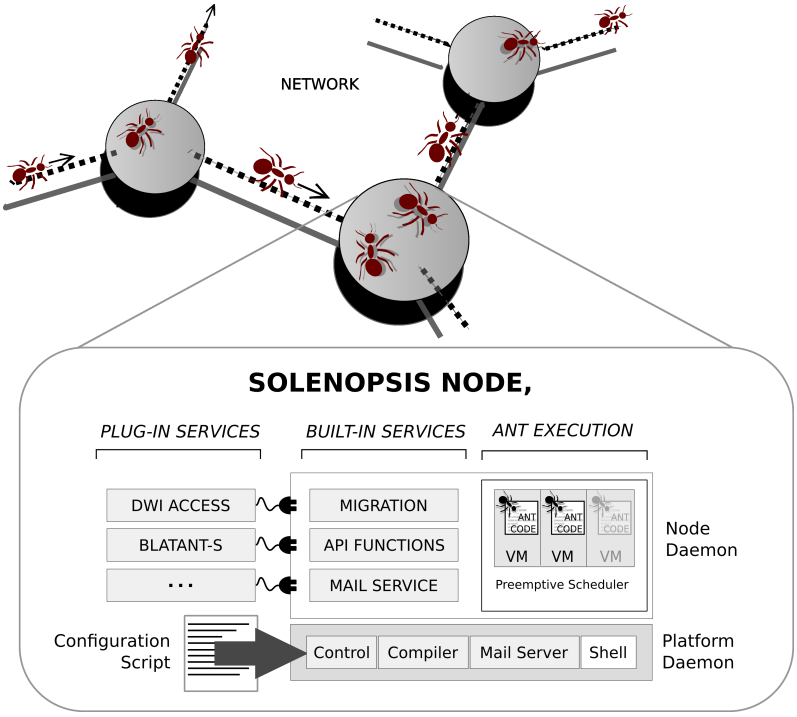


Figure 6.2: Solenopsis deployment mode with node detail

Simulation mode In simulation mode (Figure 6.3) each platform daemon manages multiple node daemons. In contrast to deployment mode, plug-in services can be shared between node daemons to reduce memory footprint. Moreover, simulation specific plug-ins

to provide useful global statistics, such as network measurements, can be enabled.

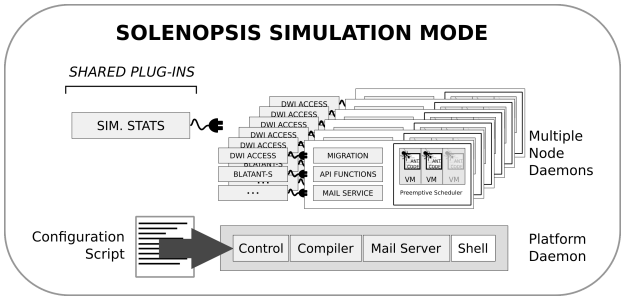


Figure 6.3: Solenopsis simulation mode

6.2.3 Ant programming language

Ants are developed in a Lisp-like language called DLisp which is compiled to a byte-code representation and executed by a stack-based virtual machine on the node. The algorithm describing the behavior of the ant as well as its runtime state are encapsulated in the ant itself. It is thus possible to create and execute different ant species in a distributed system without replacing components or functionality on each node. Moreover, as the evaluation and the deployment environment are the same, there is no need to re-implement algorithms for large-scale deployment. As the ant behavior is executed inside a virtual machine, ant code is sandboxed and only services made available by the node can be accessed.

The programming language supports different basic data types such as numbers (integers and floating point), strings, lists, dictionaries, simple closures (lambda), and nil (the only type whose semantic value is the boolean False); functions to manipulate these types are available as built-in. Moreover, macros can be developed to extend the language with custom constructs.

6.2.4 Support for transparent strong migration

One of the strengths of Solenopsis is the possibility to transparently migrate ant-agents across nodes, as part of their execution. This feature is particularly important for bio-inspired ant-algorithms, because mobility is an inherent capability of each agent. The details of transparent strong migration are shown in the example illustrated in Figure 6.4. Several steps are involved in the migration process:

1. the ant executing on a node calls the `migrate` function to migrate to another node;
2. the call is managed by the migration service running on the node;
3. a snapshot of the actual running state of the ant is requested from the control component of the node; the state includes the current program counter as well as the full execution stack;
4. the ant state is passed to the mail service to be sent to the target node;
5. the mail service serializes the received data, and forwards it to the local mail server;

6. the serialized ant state is transmitted to the receiving node's mail server;
7. the control component of the target node is instructed to create a new virtual machine instance, and restore the execution state;
8. the virtual machine is created, and scheduled for execution;
9. execution of the ant code restarts, and the instruction following the call to `migrate` is processed (the `migrate` function returns false if the migration does not succeed).

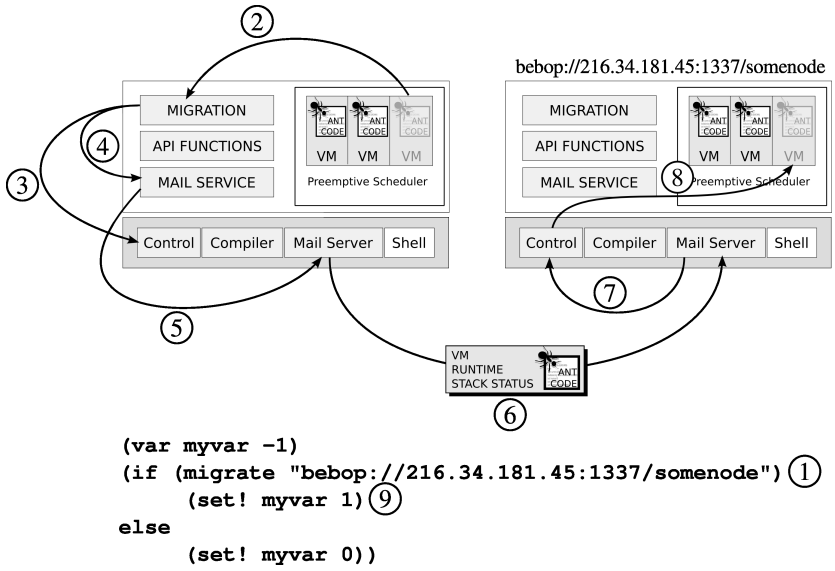


Figure 6.4: Strong transparent migration example

6.2.5 Extensibility

To integrate Solenopsis within the SmartGRID middleware a modular design approach was employed. Additional services, written in Java, can be used to extend the functionality of node services and enable access to external components and resources from the ant code. In the context of SmartGRID, the modules that are implemented as extensions to Solenopsis include the BlåtAnt algorithm, the resource discovery service, and access to the Datawarehouse to enable communication with the SRML. It is noteworthy to mention that the generic and modular design of Solenopsis enabled the integration of its core components (namely, the compiler, virtual machine, and basic services) into another project, called FlexibleRules [121], aimed at supporting the development of digital board games.

6.3 Summary

In this chapter we presented the SmartGRID, a novel grid middleware that aims at bridging the gap between applications and heterogeneous and volatile resources, by promoting a fully distributed design and self-organized operation. SmartGRID integrates

the concepts and the solutions proposed in this thesis, namely the BLÅTANT overlay management algorithm, detailed in Chapter 3, the proactive caching mechanism to improve resource discovery discussed in Chapter 4, and the ARiA protocol for fully distributed meta-scheduling, presented in Chapter 5. The framework is based on a multi-level design composed of two layers: the Smart Signaling Layer (SSL) and the Smart Resource Management Layer (SRML). The first is in charge of managing low-level aspects of the middleware, such as communication between grid nodes and resource discovery, whereas the latter deals with high-level concerns such as job scheduling and resource management. The SSL is implemented by a runtime platform called SOLENOPSIS that supports the development and deployment of ant-based distributed algorithms, whereas the SRML is composed of MAGATE components that provide an interface to grid applications, schedulers, and existing grid middlewares.

Chapter 7

Conclusions

Contents

Self-organization and adaptiveness are commonly viewed as important aspects to support reliable, efficient, and scalable distributed solutions. In this respect, the development of novel self-organized and adaptive solutions for distributed systems was the impulse that motivated the research carried out in this thesis. To this extent several essential aspects of autonomic management and exploitation of distributed systems have been studied, in particular the construction of an optimized peer-to-peer overlay with bounded diameter and girth, the implementation of a resource discovery mechanism based on local shortcut caches to increase the efficiency of flooding based protocols, and the definition of a fully distributed meta-scheduling protocol that improves task allocation across large pools of resources. Each of these aspects has been thoroughly researched, and a comprehensive literature review of existing solutions has not only motivated but also driven our research and developments. Moreover, a clear separation of concerns enables the incorporation of the solutions implemented in this thesis into a variety of situations where fully distributed and autonomic operation is required, although the target of the evaluation has been the grid scenario defined by the SmartGRID project.

The solutions that have been proposed fit extremely well into the emerging self-organization realm. In this chapter we summarize the main contributions for each of the topics considered in this thesis, and highlight their distinctive characteristics as well as future research directions.

7.1 Overlay management

The proposed overlay management algorithm, called BlåtAnt, answers the problem of maintaining an optimized overlay that enables communication between peers with a reduced number of retransmissions. By means of bio-inspired, fully distributed techniques

the overlay is managed in a self-organized and adaptive way, and is scalable as well as robust to both node and communication failures. The behavior of BLÅTANT has been empirically studied under different network conditions, and results of simulations have shown to be promising in respect to the possibility of employing bio-inspired optimization in communication networks. More specifically, the two proposed implementations of the algorithm, namely BLÅTANT-R and BLÅTANT-S, enable effective management of peer-to-peer overlays and provide satisfactory performance under churn and in the event of unexpected communication failures. The produced overhead on network resources as well as the the scalability of the system have been deemed reasonable in respect to other existing solutions (such as GNUTELLA or NEWSCAST). Hence, in response to the research problem set in Chapter 1, *"Can we exploit self-organization and bio-inspired solutions to provide an optimized peer-to-peer communication and service provisioning framework?"*, we can assert that our contribution provides a sufficient and affirmative answer.

7.1.1 Future research directions

Although the considered evaluation scenarios have proven the viability of BLÅTANT, more extensive experimentation is undoubtedly required to understand the implications of different network conditions on the robustness and reliability of the algorithm. Moreover, full scale tests in a real network would help determine the limits of our approach, better comprehend the influence of the considered parameters, and drive further improvements on the underlying logic. In this sense, an interesting research direction will consider the optimization of the overlay according to the underlying network topology, in order to reduce transmission delays and low-level traffic.

The collaborative process of detecting long paths and small cycles carried out by nodes could be improved by letting nodes exchange more information. In particular, if a node detects a cycle for which it is not responsible (hence it cannot break), a notification to the master of that cycle could be sent. Conversely, neighbor nodes could be queried in order to determine distances on the graph with more precision. Moreover, to detect some partitioning situations, nodes could analyze the information brought by *Discovery Ants* and measure its entropy. In a similar way, the optimization process could be made adaptive in respect to the perceived dynamicity of the network.

Security should also be studied in a more comprehensive manner; in particular an analysis of the robustness of the system in presence of misbehaving peers, and against targeted attacks has been neglected in this thesis, as its scope and time plan did not allow us to delve into this field. Future work should definitely consider such concerns in order to produce a solid distributed platform.

Furthermore a detailed comparison with other existing approaches (besides the ones considered in this thesis) would help us understand the benefits and weaknesses of our approach. In this regard, the lack of a common evaluation platform where different peer-to-peer algorithms could be evaluated under the same premises is considered as the most important issue that would need to be solved in the future.

7.2 Resource discovery

The second question set in Chapter 1 concerns the problem of locating information in a distributed system, and asks *"Can we improve existing resource discovery mechanism using fully distributed bio-inspired solutions?"*. The approach that we proposed in this thesis implements a local routing cache on each node that stores references to other nodes in the overlay that share similar resources. In order to reduce the overhead derived from updates to the cache, which are achieved by means of proactive queries broadcasted on the network, an epidemic algorithm was employed to merge the contents of local caches between nodes. In this regard, a bio-inspired solution like epidemic cache merges enabled us to manage local caches with minimal network overhead, without sacrificing the quality of the contained information. By means of extensive experimentation in different scenarios, the benefits of such semantic-aware techniques were evident, even when combined with another mechanism aimed at improving flooding-based resource discovery, namely replication. As determined by evaluation on different types of overlay, the improvements brought by the proposed approach are independent from the peer-to-peer topology, hence our solution can be implemented in a variety of scenarios.

7.2.1 Future research directions

It would be of merit to compare our solution with other flooding improvement techniques, and on different peer-to-peer overlays (either unstructured or structured). Different forwarding techniques should also be considered: in this context we find non-forwarding approaches presented in [303, 188] interesting. In the same sense, the forwarding strategy employed in the cache could be improved, for example by favoring routing towards nodes with higher similarity: this solution would nonetheless require similarity values to be stored in the cache itself.

Several improvements of the proactive caching scheme are possible. In particular, refinements to the similarity function could be introduced to support more complex semantics, although the one used in our evaluation to determine matching resources is appropriate for grid scenarios. Another aspect worth considering in future research works concerns the outcome of the merging process: whereas the implemented solution filters the entries to retain after a merge by selecting the most recent ones based on their age information, more useful information could be retained by employing information about the similarity.

7.3 Meta-scheduling

Efficient task-allocation concerns the last question that we asked in Chapter 1, and to answer it a fully-distributed meta-scheduling mechanism called ARiA was presented in Chapter 5. Our solution implements a lightweight protocol that enables decentralized coordination of local schedulers, without requiring each node to disclose the details of its own scheduling policy, which ensures flexibility and scalability. Empirical evaluation validated the benefits of the protocol in different conditions in terms of decreased total job execution time and improved load-balancing.

7.3.1 Future research directions

The encouraging results obtained by our evaluation provide a solid base for future developments, which should primarily focus on some issues that were consciously set aside in this thesis. In this regard, interesting future research directions include evaluation with grid schedulers in a real grid deployment. In this thesis we modeled several simple policies for batch and deadline scheduling, nevertheless real schedulers deal with more unexpected situations, that include job revocation, requirement changes, queue holding, and hardware or software failures. Also, the scope and time plan of this thesis did not allow us to dig into the problem of parallel job scheduling on the same node, having chosen to employ a simpler model where only one job at a time executes on a node. Therefore, a wider range of scheduling and execution policies could be introduced.

The logic to determine the nodes best suited to schedule tasks on could be changed to include information other than the expected cost. More specifically, as proposed in [148], node trust and reputation could be taken into account.

As for overlay management, reliability and security within the proposed scheduling framework should be studied in a more comprehensive manner. Concerning the first issue, our evaluation assumed that no failure could terminate job execution; in this sense, mechanisms to enable job recovery and resubmission in the event of a failure should be implemented in the future. Conversely, regarding security in our evaluation job allocation was performed under the premise that trust relationships exist between each participating nodes, although this cannot be assumed in a real distributed scenario.

7.4 SmartGRID

We presented the SmartGRID middleware architecture, that aims at providing a fully-distributed solution to operate a grid environment. SmartGRID builds on two functional layers, and the work presented in this thesis covers the signaling layer, which is responsible for communication between nodes and monitoring of the network. In this context, we proposed a software framework called Solenopsis, which enables the development and execution of ant-based distributed algorithms, and helps bringing together all the aforementioned functional components (overlay management, resource discovery, scheduling) into the SmartGRID architecture. By means of a simple, modular design, our solution is also flexible and easily extensible, and could be implemented in other scenarios. The promoted programming language enables fast prototyping of ant mobile agents, and is supported by a runtime environment that supports transparent strong migration across the overlay. This feature simplifies the development of mobile code, by removing the hassle of requiring explicit execution state serialization and de-serialization.

7.4.1 Future research directions

In the context of the SmartGRID project, future work will focus on evaluating all aspects of the middleware, in particular job submission and execution, in a full-scale grid environment. Furthermore, further development of Solenopsis should improve support for controlled simulation conditions, for example by considering network latency, as well

as by implementing additional measurement and statistical utilities.

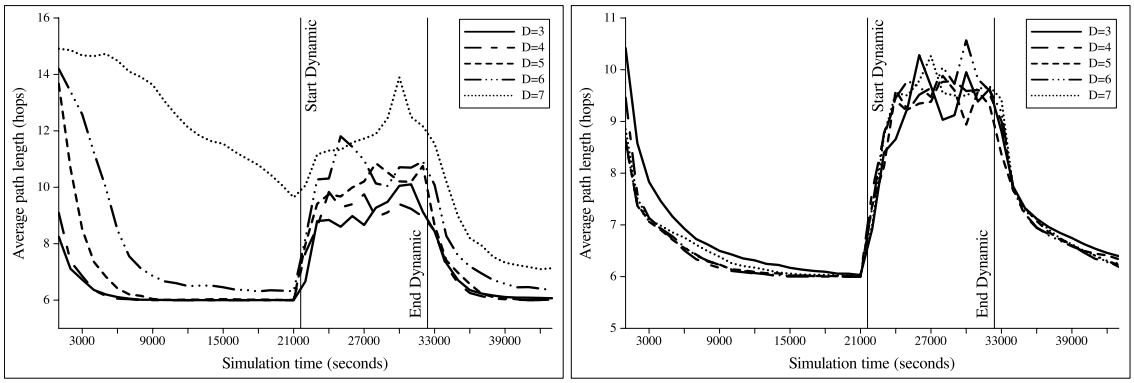
7.5 Epilogue

Self-organization can bring a decisive improvement in the performance, reliability, and robustness of distributed systems. In this context, bio-inspired unsupervised solutions can be used to achieve self-organization and optimal operation of networked systems. The proposed framework of algorithms shows that it is possible to exploit self-organized behaviors to support or improve different areas of distributed computing, namely peer-to-peer overlay management, resource discovery, and task allocation. The central part of our work, which consists of the BlåtAnt algorithm, achieves fully distributed management and optimization of a peer-to-peer overlay by means of a process that uses bio-inspired techniques. Beside that, epidemic algorithms have proven to be a simple yet efficient solution to share information between nodes, and improve search in unstructured overlays. Finally, self-organization can also be used to achieve optimal fully-distributed task allocation in grids, thus support benefits in terms of performance, scalability and robustness. Undoubtedly we do not claim that bio-inspired solutions address to a full extend the problems of self-organization and self-management of distributed systems. Nonetheless in the considered scenarios they have proved to be suitable approaches providing satisfactory performance. This further strengthens our belief that self-organized and bio-inspired techniques are worthy contenders in the field of distributed systems design.

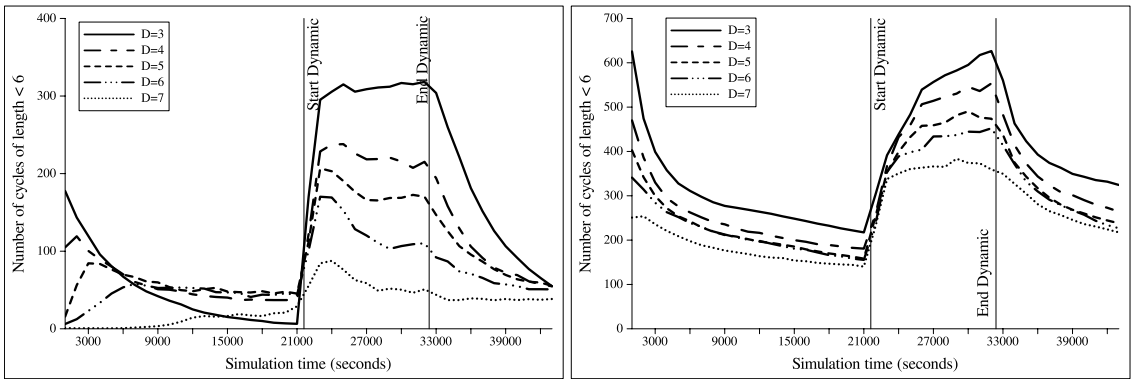
Appendix A

Detailed Results for Chapter 3

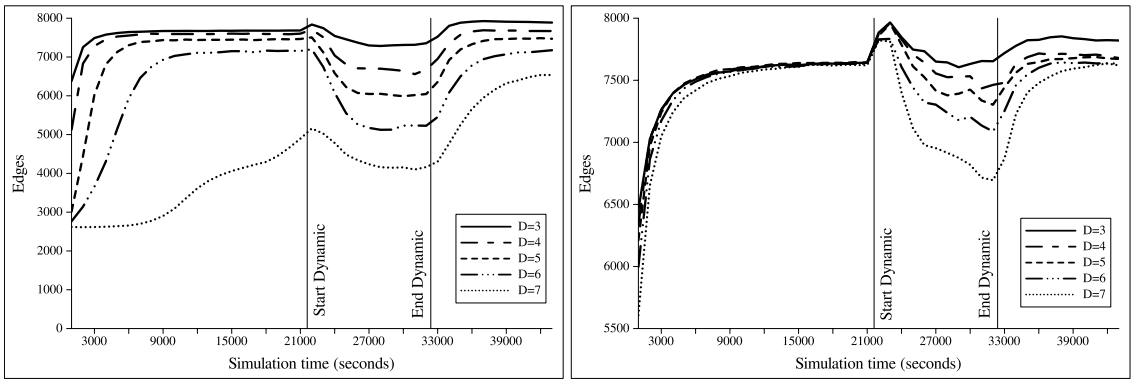
BLÅTANT-R BLÅTANT-S



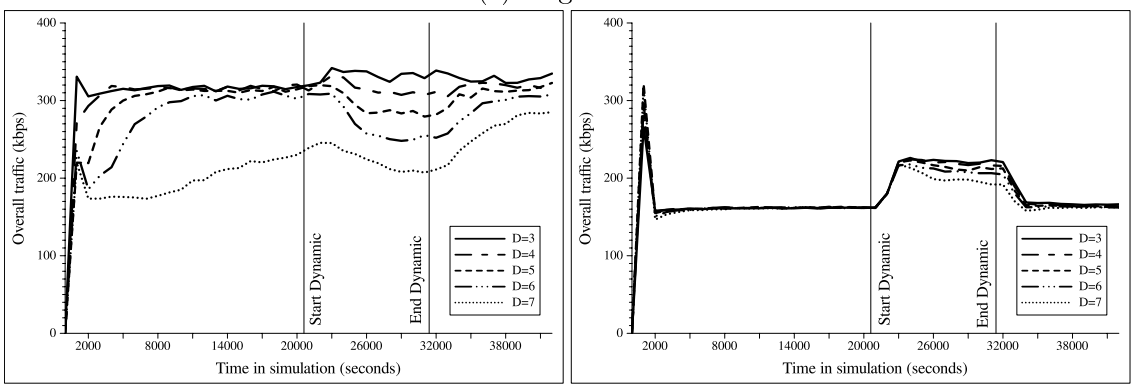
(a) Average path length



(b) Graph cycles



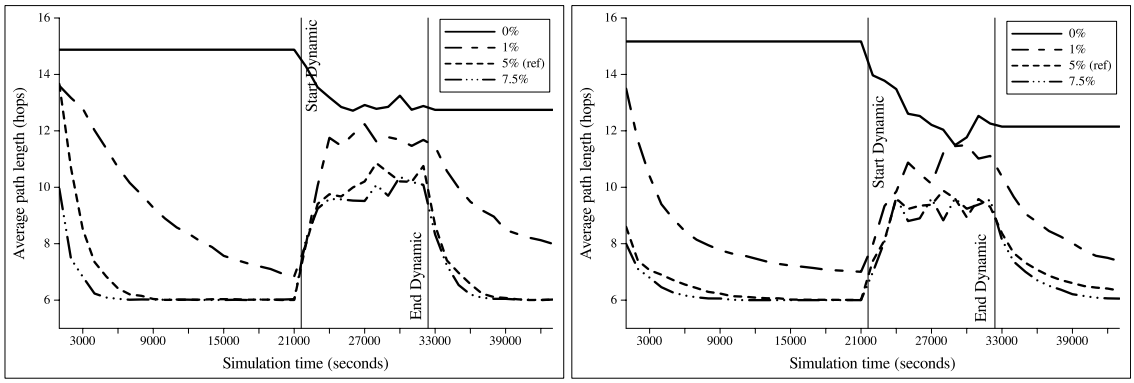
(c) Edge count



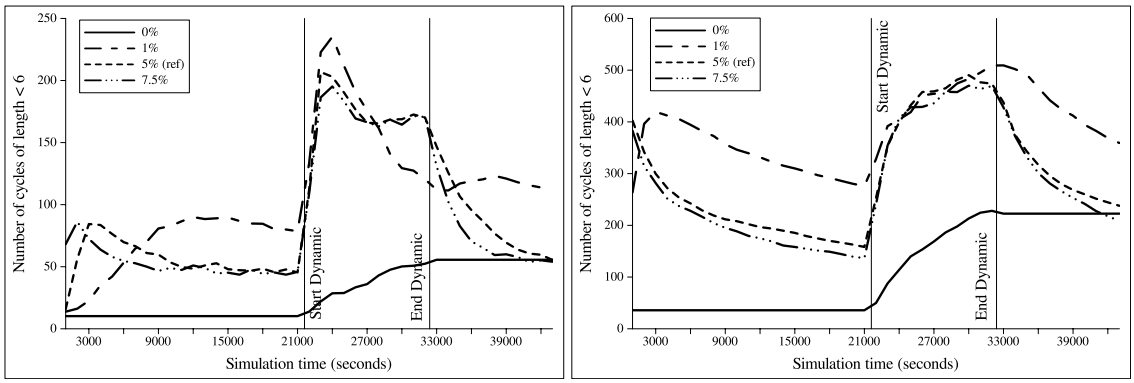
(d) Traffic

Figure A.1: F1 - Optimization parameter D - Sensitivity analysis

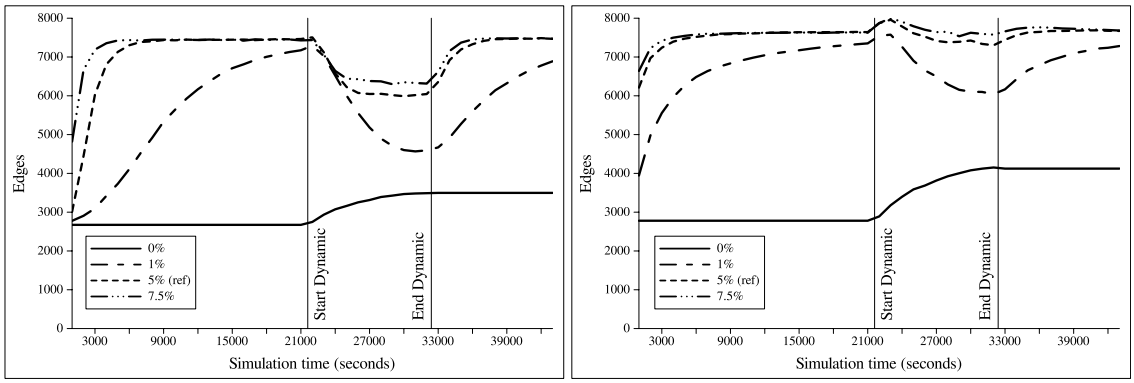
BLÅTANT-R BLÅTANT-S



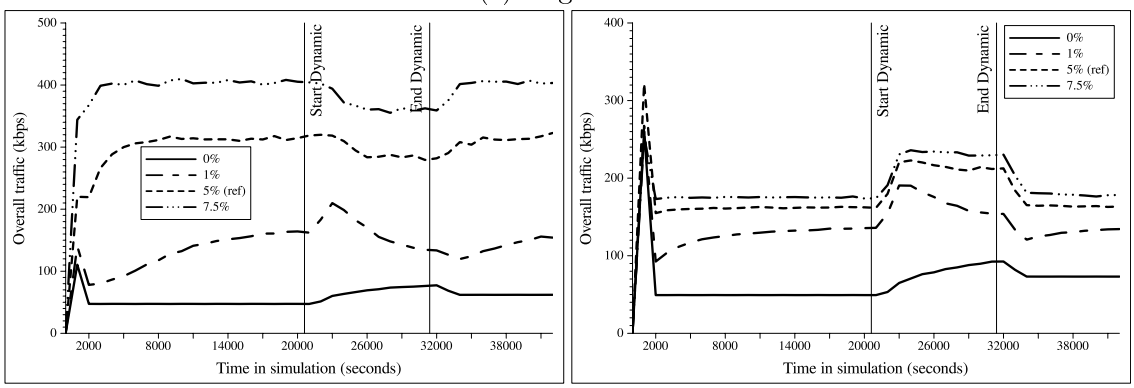
(a) Average path length



(b) Graph cycles



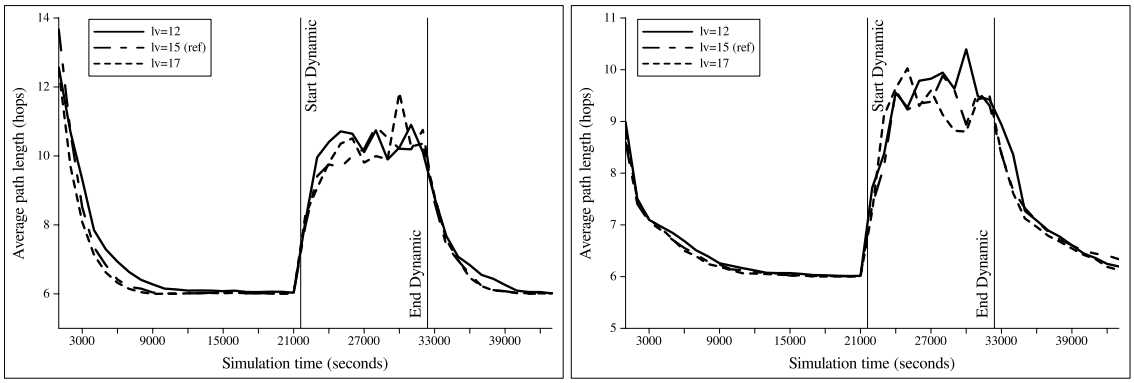
(c) Edge count



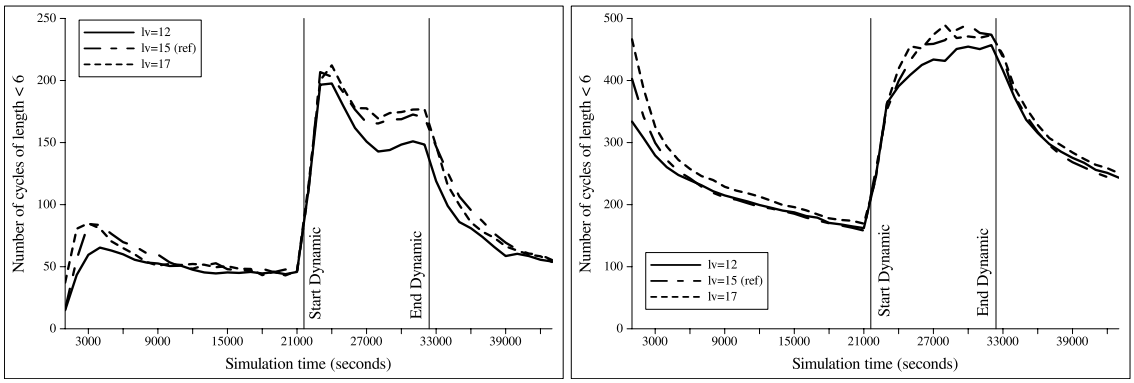
(d) Traffic

Figure A.2: F2 - *Discovery Ant* birth probability μ - Sensitivity analysis

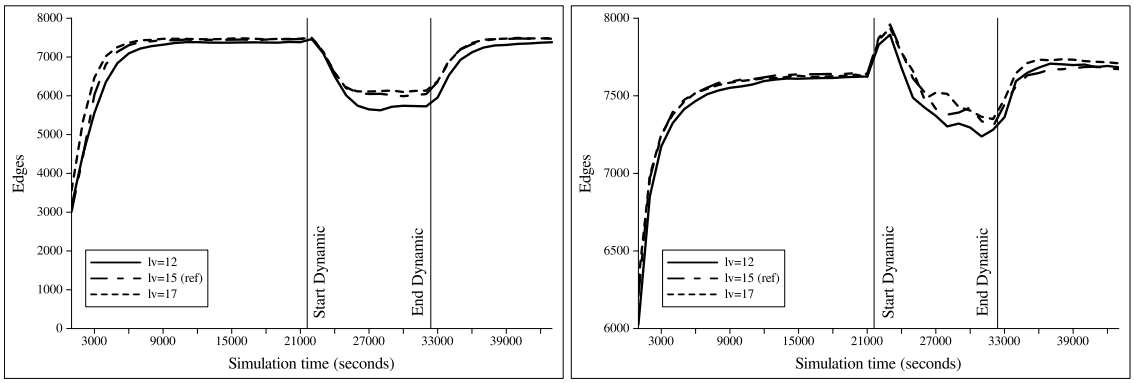
BLÅTANT-R BLÅTANT-S



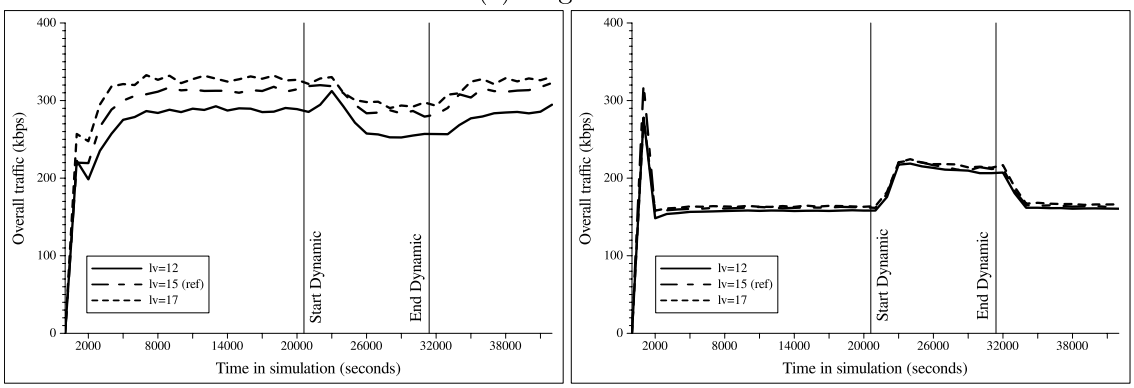
(a) Average path length



(b) Graph cycles



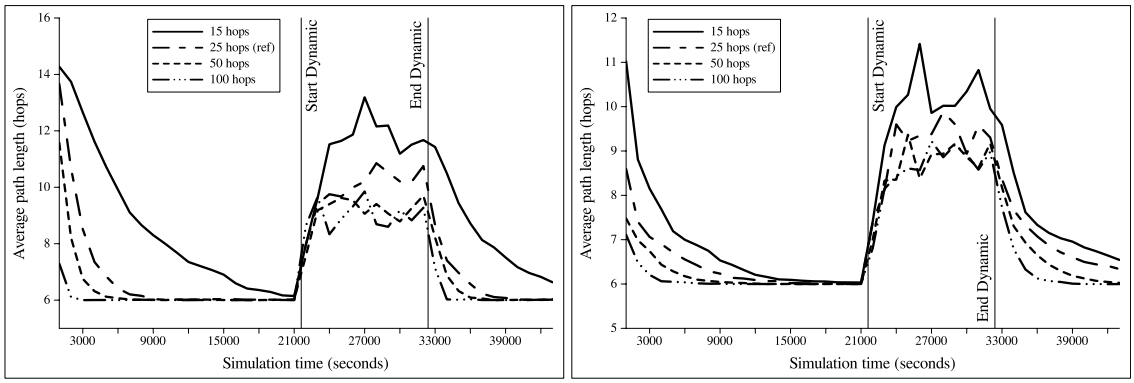
(c) Edge count



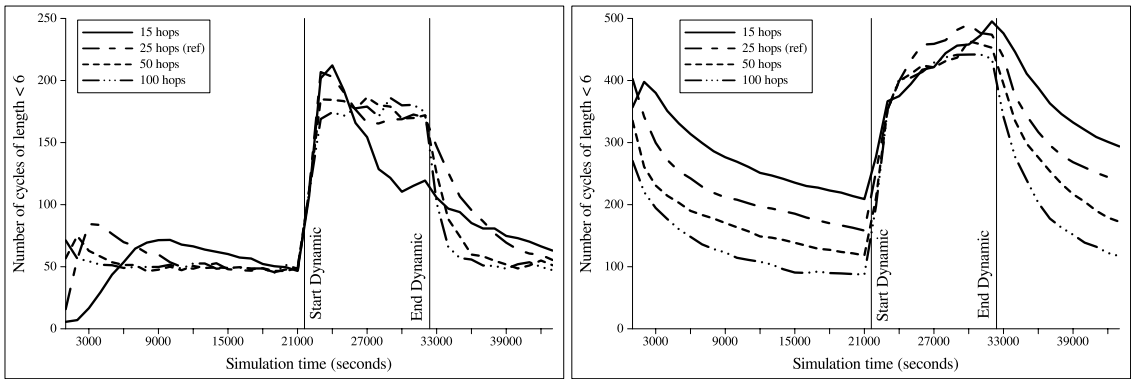
(d) Traffic

Figure A.3: F3 - Maximum length l_v of the information vector - Sensitivity analysis

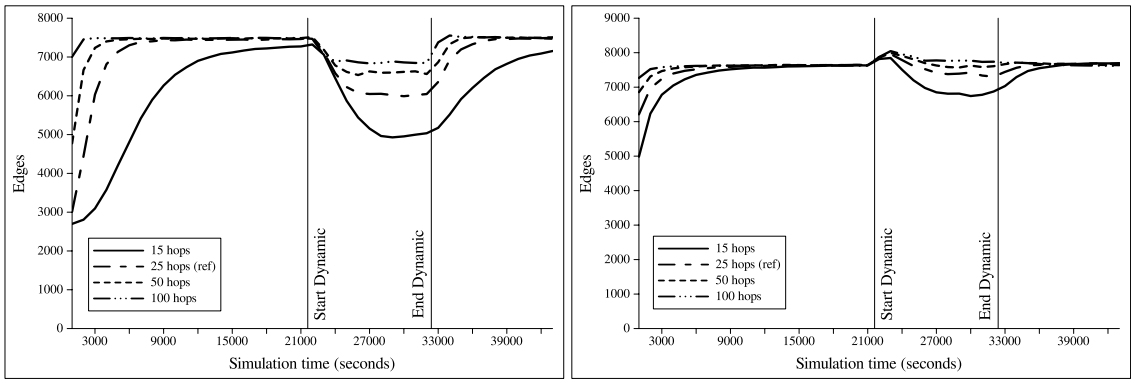
BLÅTANT-R BLÅTANT-S



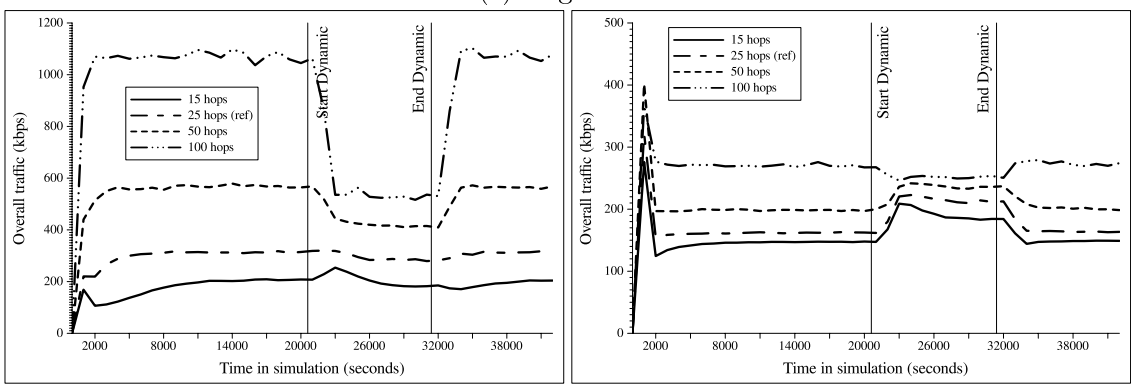
(a) Average path length



(b) Graph cycles



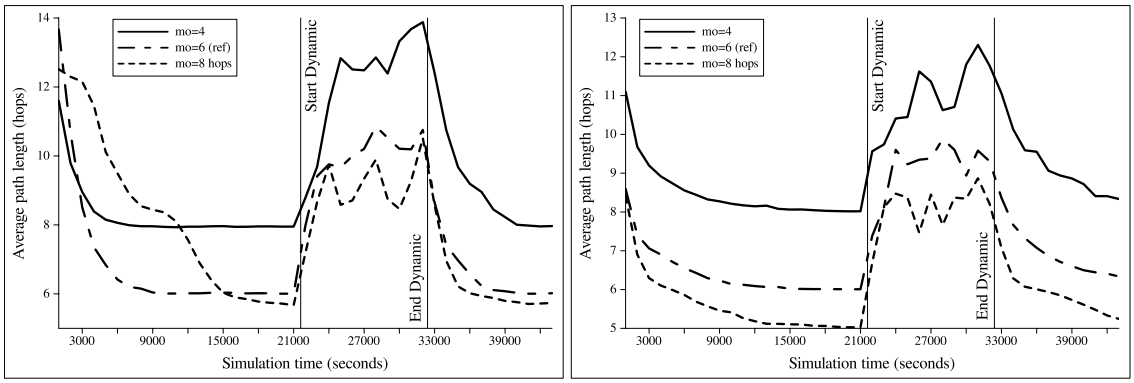
(c) Edge count



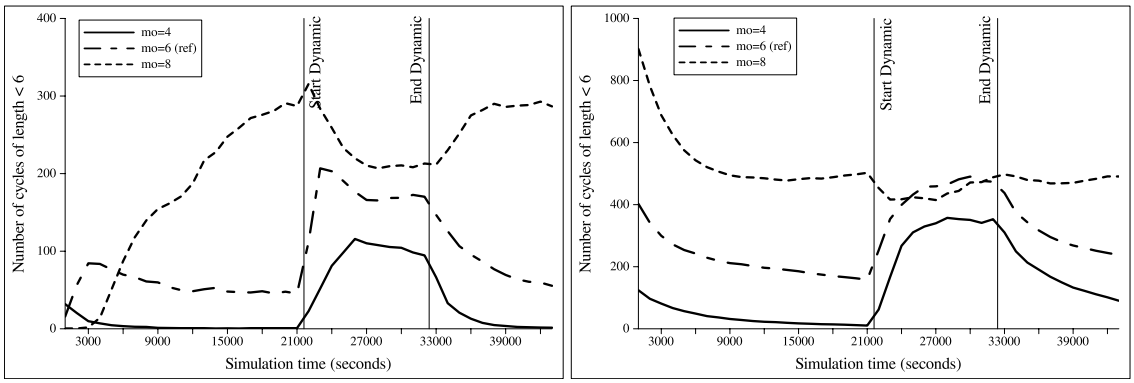
(d) Traffic

Figure A.4: F4 - Maximum number of allowed *Discovery Ant* hops π - Sensitivity analysis

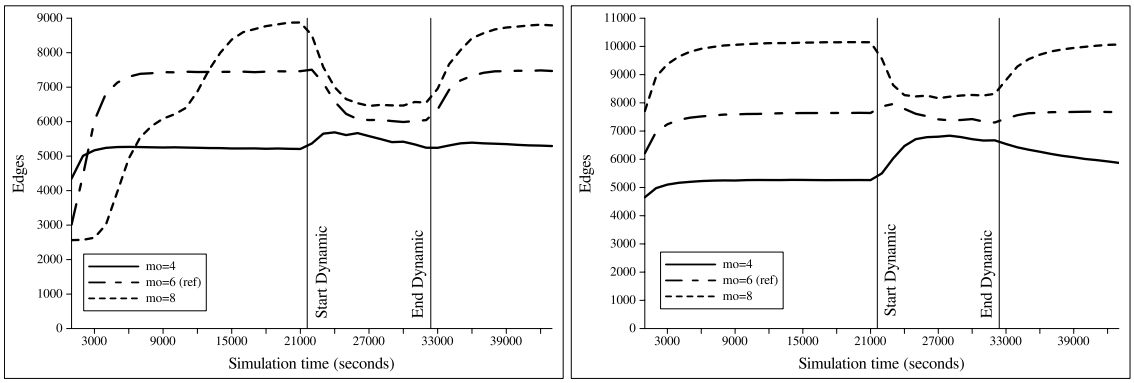
BLÅTANT-R BLÅTANT-S



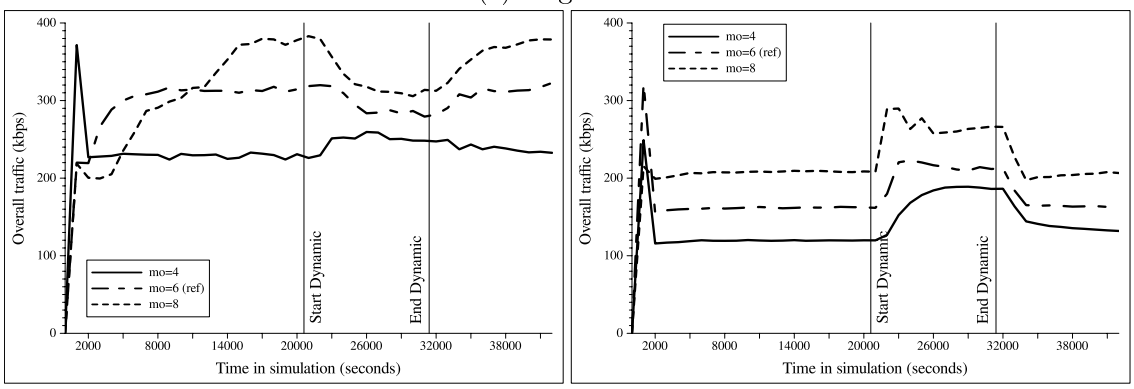
(a) Average path length



(b) Graph cycles



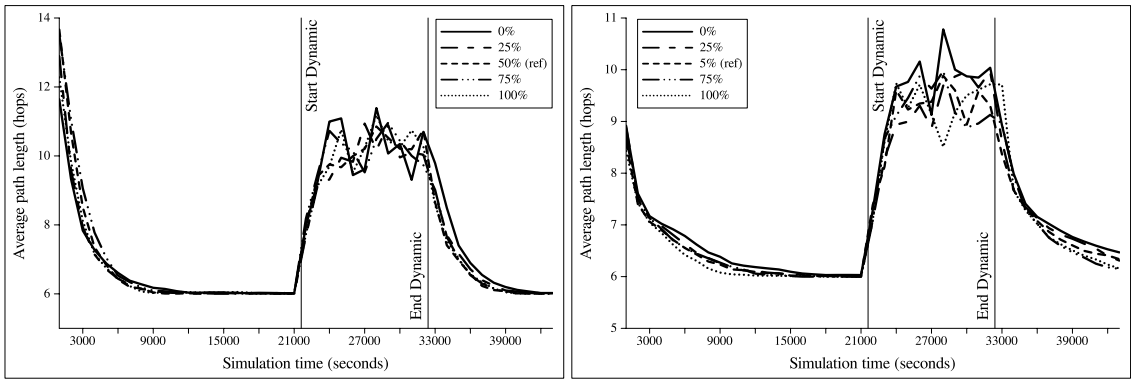
(c) Edge count



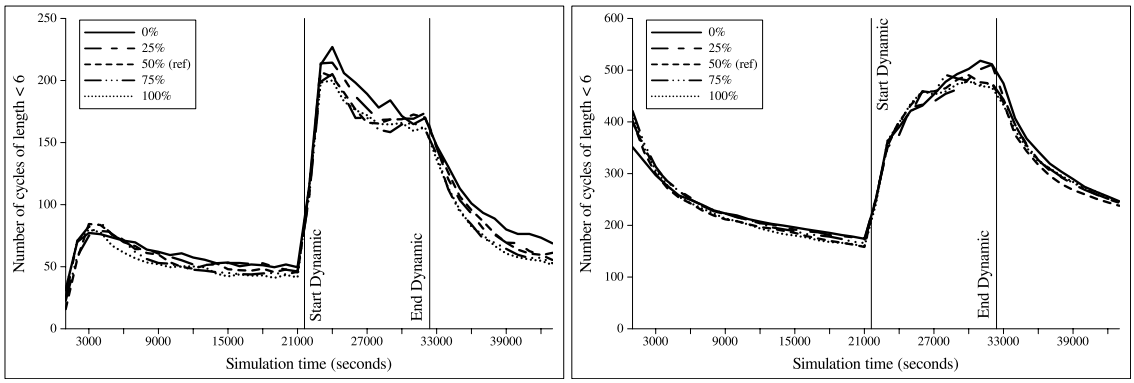
(d) Traffic

Figure A.5: F5 - Maximum per-node degree *mo* - Sensitivity analysis

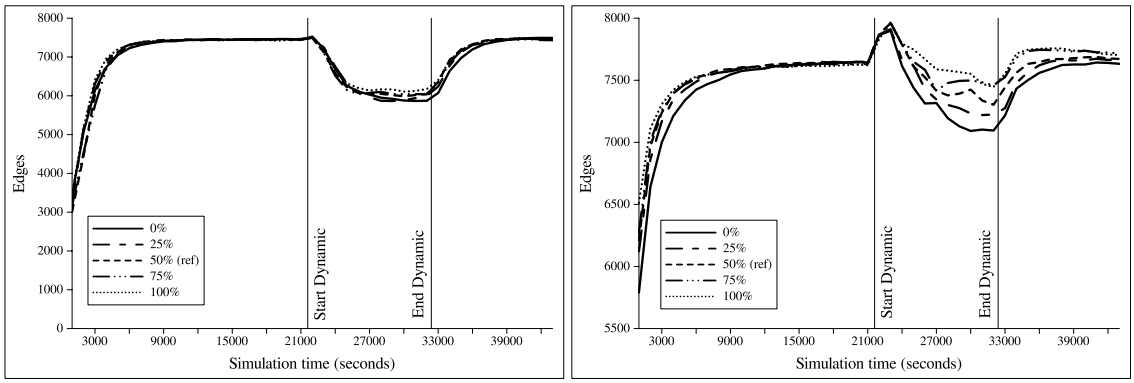
BLÅTANT-R BLÅTANT-S



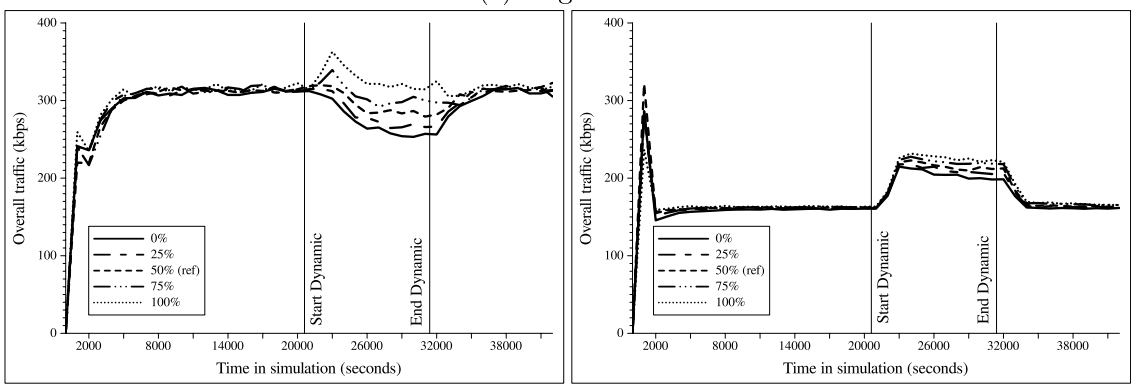
(a) Average path length



(b) Graph cycles



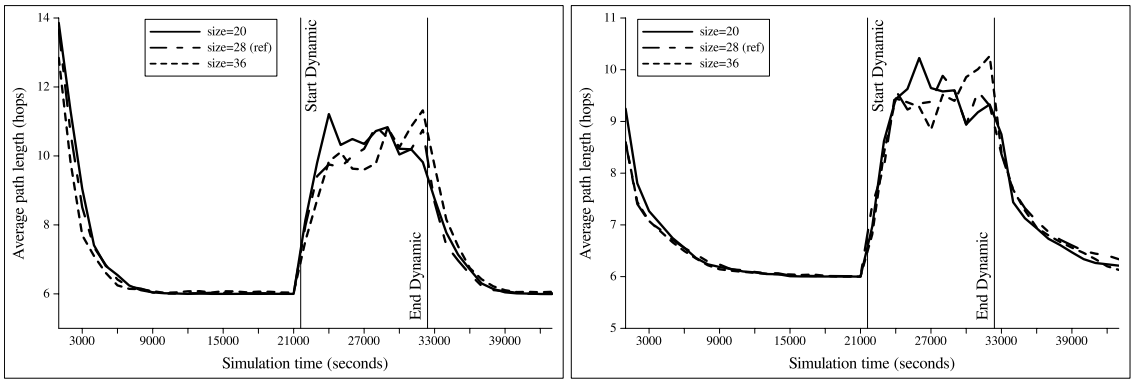
(c) Edge count



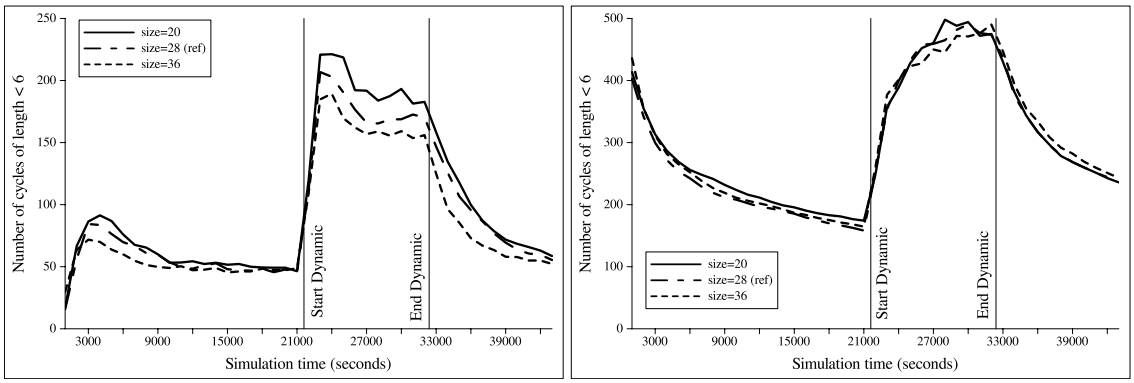
(d) Traffic

Figure A.6: F6 - Exploration *versus* Exploitation - Sensitivity analysis

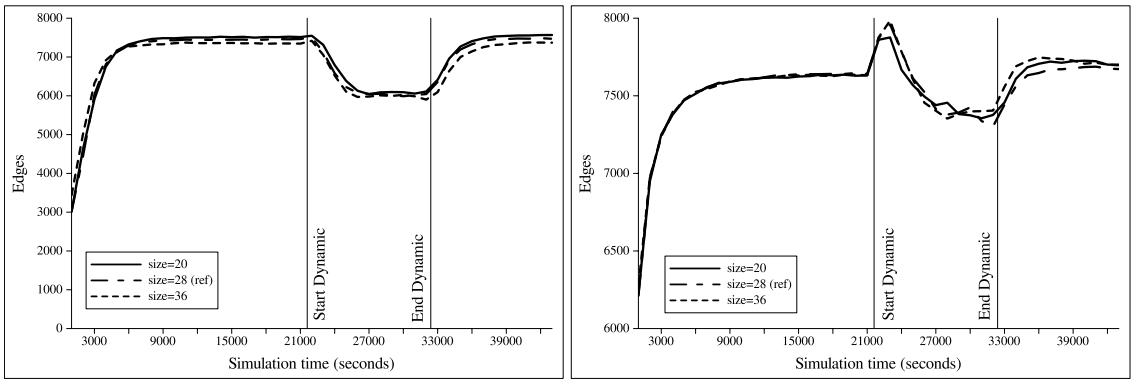
BLÅTANT-R BLÅTANT-S



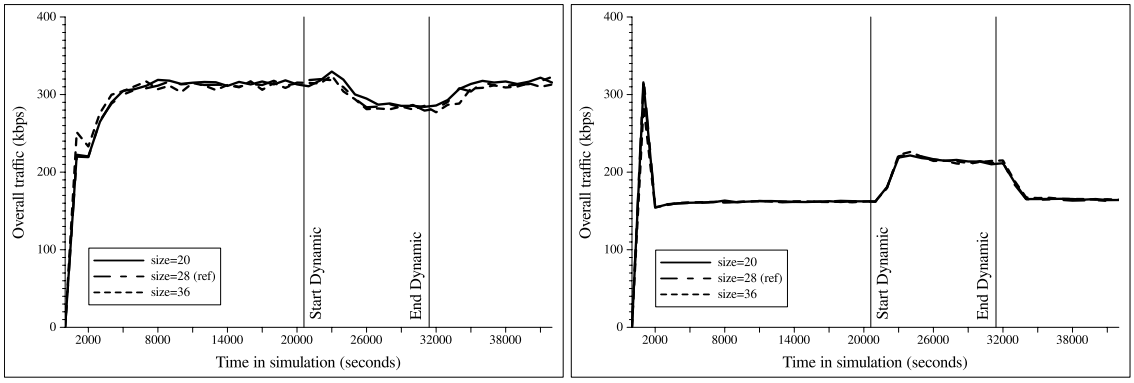
(a) Average path length



(b) Graph cycles



(c) Edge count



(d) Traffic

Figure A.7: F7 - Size of the α table - Sensitivity analysis

Bibliography

[1] Bittorrent homepage. http://www.bittorrent.org. 13

[2] gift-fasttrack, an open source implementation of the fasttrack p2p protocol. http://gift-fasttrack.berlios.de/. 37

[3] Gnutella homepage. http://www.gnutella.com. 2, 33, 62

[4] Gnutella protocol 0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf. 33

[5] Gnutella protocol 0.6 (draft). http://wiki.limewire.org/index.php?title=GDF. 33, 35

[6] Gnutella2 homepage. http://g2.trillinux.org. 35

[7] Joltid homepage. http://joltid.com/. 37

[8] Kazaaa homepage. http://www.kazaa.com/. 2, 37

[9] Napster homepage. http://www.napster.com. 13, 28, 33

[10] *Phenix: supporting resilient low-diameter peer-to-peer topologies*, volume 1, 2004. 39

[11] *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol*, 2006. 10

[12] Bittorrent website, http://www.bittorrent.com/, June 2010. 23

[13] edonkey protocol, http://sourceforge.net/projects/pdonkey/, June 2010. 23

[14] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Record*, 32(3):29–33, 2003. 24

[15] William Acosta and Surendar Chandra. Trace driven analysis of the long term evolution of gnutella peer-to-peer traffic. In *PAM*, pages 42–51, 2007. 35

[16] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physical Review E*, 64(4):046135+, Sep 2001. 31

[17] Mona Aggarwal, Robert D. Kent, and Alioune Ngom. Genetic algorithm based scheduler for computational grids. In *HPCS '05: Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, pages 209–215, Washington, DC, USA, 2005. IEEE Computer Society. 108

[18] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Comput. Netw. ISDN Syst.*, 47(4):445–487, 2005. 1

[19] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47, 2002. 27

[20] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, July 2000. 28

[21] Jim Almond and Dave Snelling. Unicore: uniform access to supercomputing as an element of electronic commerce. *Future Gener. Comput. Syst.*, 15(5-6):539–548, 1999. 131

[22] Noga Alon, Shlomo Hoory, and Nathan Linial. The moore bound for irregular graphs. *Graphs Combin*, 18:53–57, 2002. 47

[23] A. Andrzejak, A. Reinefeld, F. Schintke, T. Schütt, C. Mastroianni, P. Fragopoulou, D. Kondo, P. Malecot, G. Cosmin Silaghi, L. Moura Silva, P. Trunfio, D. Zeinalipour-Yazti, and E. Zimeo. Grid architectural issues: State-of-the-art and future trends. CoreGRID White Paper, May 2008. 106

[24] Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 33, Washington, DC, USA, 2002. IEEE Computer Society. 26, 43

[25] Manish Arora, Sajal K. Das, and Rupak Biswas. A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. In *ICPPW '02: Proceedings of the 2002 International Conference on Parallel Processing Workshops*, page 499, Washington, DC, USA, 2002. IEEE Computer Society. 108

[26] Elias Athanasopoulos, Kostas G. Anagnostakis, and Evangelos P. Markatos. Misusing unstructured p2p systems to perform dos attacks: The network that never forgets. In *in Proceedings of the 4th International Conference on Applied Cryptography and Network Security (ACNS'06*, pages 130–145, 2006. 35

[27] Özalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 15, Washington, DC, USA, 2002. IEEE Computer Society. 132

[28] Erwin M. Bakker, Jan van Leeuwen, and Richard B. Tan. Prefix routing schemes in dynamic networks. *Comput. Netw. ISDN Syst.*, 26(4):403–421, 1993. 16

[29] Roberto Baldoni, Adnan, Sirio Scipioni, and Sara Tucci-Piergiovanni. Churn resilience of peer-to-peer group membership: A performance analysis. *Lecture notes in Computer Science*, 3741:226–237, December 2005. 41

[30] Shane Balfe, Amit D. Lakhani, and Kenneth G. Paterson. Trusted computing: Providing security for peer-to-peer networks. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 117–124, Washington, DC, USA, 2005. IEEE Computer Society. 3

[31] Albert-Laszlo Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means*. Plume, reissue edition, April 2003. 28

[32] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999. 28

[33] Sujoy Basu, Sujata Banerjee, Puneet Sharma, and Sung-Ju Lee. Nodewiz: peer-to-peer resource discovery for grids. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 1*, pages 213–220, Washington, DC, USA, 2005. IEEE Computer Society. 43

[34] Dominic Battre, Andre Hoing, Martin Raack, Ulf Rerrer-Brusch, and Odej Kao. Extending pastry by an alphanumerical overlay. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 36–43, Washington, DC, USA, 2009. IEEE Computer Society. 26

[35] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990. 26

[36] Gerardo Beni. From swarm intelligence to swarm robotics. In *Swarm Robotics*, pages 1–9. 2005. 50

[37] F Berman, G C Fox, and A J G Hey. The grid: past, present, future, 2003. 106

[38] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on*, volume 2735 of *Lecture Notes in Computer Science*, pages 256–267. Springer Berlin / Heidelberg, February 2003. 41

[39] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353–366, 2004. 26, 43

[40] Peter Biddle, Paul England, Marcus Peinado, and Bryan Willman. The darknet and the future of content distribution. In *ACM Workshop on Digital Rights Management*, November 2002. 35

[41] Nabhendra Bisnik and Alhussein Abouzeid. Modeling and analysis of random walk search algorithms in p2p networks. In *HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 95–103, Washington, DC, USA, 2005. IEEE Computer Society. 31

[42] Nabhendra Bisnik and Alhussein A. Abouzeid. Optimizing random walk search algorithms in p2p networks. *Comput. Netw.*, 51(6):1499–1514, 2007. 31

[43] Roland Bless, Christian Hübsch, Sebastian Mies, and Oliver P. Waldhorst. The Underlay Abstraction in the Spontaneous Virtual Networks (SpoVNet) Architecture. In *Proc. 4th EuroNGI Conf. on Next Generation Internet Networks (NGI 2008)*, pages 115–122, Krakow, Poland, April 2008. 15

[44] Richard Boardman, Stephen Crouch, Hugo Mills, Steven Newhouse, and Juri Papay. Towards grid interoperability. In *All Hands Meeting 2007, OMII-UK Workshop*, September 2007. 107

[45] Béla Bollobás. The diameter of random graphs. *Transactions of the American Mathematical Society*, 267(1):41–52, 1981. 27

[46] Béla Bollobas. *Random Graphs*. Cambridge University Press, 2001. 27

[47] Béla Bollobás and Oliver Riordan. Robustness and vulnerability of scale-free random graphs. *Internet Mathematics*, 1(1), 2003. 28

[48] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999. 3, 50

[49] Amos Brocco, Fulvio Frapolli, and Béat Hirsbrunner. Blatant: Bounding networks' diameter with a collaborative distributed algorithm. In *Sixth International Conference on Ant Colony Optimization and Swarm Intelligence*, pages 275–282. ANTS, Springer, September 2008. 50

[50] Amos Brocco, Fulvio Frapolli, and Béat Hirsbrunner. Shrinking the network: The blatant algorithm. Technical Report 08-04, Department of Informatics, University of Fribourg, Fribourg, April 2008. 50

[51] Amos Brocco, Fulvio Frapolli, and Béat Hirsbrunner. Bounded diameter overlay construction: A self organized approach. In *IEEE Swarm Intelligence Symposium*, pages 114–121. SIS 2009, IEEE, April 2009. 50

[52] Amos Brocco, Béat Hirsbrunner, and Michèle Courant. Solenopsis: A framework for the development of ant algorithms. In *Swarm Intelligence Symposium*, pages 316–323. SIS, IEEE, April 2007. 130, 131

[53] Amos Brocco, Apostolos Malatras, and Béat Hirsbrunner. Proactive information caching for efficient resource discovery in a self-structured grid. In *Workshop on Bio-Inspired Algorithms for Distributed Systems*, pages 11–18. ICAC 2009, ACM, June 2009. 88

[54] Amos Brocco, Apostolos Malatras, and Béat Hirsbrunner. Enabling efficient information discovery in a self-structured grid. *Future Generation Computer Systems, Elsevier*, 2010. Accepted. 88

[55] Amos Brocco, Apostolos Malatras, Ye Huang, and Béat Hirsbrunner. Aria: A protocol for dynamic fully distributed grid meta-scheduling. In *Proceedings of The 30th International Conference on Distributed Computing Systems*. ICDCS 2010, IEEE, June 2010. Accepted. 109

[56] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, volume 1, pages 636–646, 2002. 32

[57] N. G. de Bruijn. A combinatorial problem. *PKNAW*, 49:758–764, 1946. 15

[58] John Buford, Heather Yu, and Eng Keong Lua. *P2P Networking and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. 1, 11, 13

[59] Min Cai and Martin Frank. Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 650–657, New York, NY, USA, 2004. ACM. 27

[60] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. Maan: A multi-attribute addressable network for grid information services. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, page 184, Washington, DC, USA, 2003. IEEE Computer Society. 26, 27, 43

[61] Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd. Grid load balancing using intelligent agents. *Future Generation Computer Systems*, 21(1):135–149, 2005. 107, 108

[62] Gianni A. Di Caro, Frederick Ducatelle, and Luca M. Gambardella. Theory and practice of ant colony optimization for routing in dynamic telecommunications networks, 2008. 47

[63] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 1998. 3, 5

[64] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002. 18

[65] Miguel Castro, Manuel Costa, and Antony Rowstron. Should we build gnutella on a structured overlay? *SIGCOMM Comput. Commun. Rev.*, 34(1):131–136, 2004. 40

[66] Miguel Castro, Manuel Costa, and Antony Rowstron. Debunking some myths about structured and unstructured overlays. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 85–98, Berkeley, CA, USA, 2005. USENIX Association. 41

[67] Miguel Castro, Peter Druschel, Anne marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *In SOSP 03*, 2003. 18

[68] Arjav J. Chakravarti and Gerald Baumgartner. The organic grid: Self-organizing computation on a peer-to-peer network. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 96–103, Washington, DC, USA, 2004. IEEE Computer Society. 108

[69] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, New York, NY, USA, 2003. ACM. 33, 35

[70] Hao Chen, Hai Jin, Jianhua Sun, Dafu Deng, and Xiaofei Liao. Analysis of large-scale topological properties for peer-to-peer networks. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:27–34, 2004. 32

[71] Ann Chervenak and Shishir Bharathi. Peer-to-peer approaches to grid resource discovery, 2008. 43, 44

[72] Jonathan Chin, Matthew J. Harvey, Shantenu Jha, and Peter V. Coveney. Scientific grid computing: The first generation. *IEEE Computing in Science and Engineering*, 7(5):24–32, 2005. 2, 106

[73] Vicent Cholvi, Pascal Felber, and Ernst Biersack. Efficient search in unstructured peer-to-peer networks. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 271–272, New York, NY, USA, 2004. ACM. 33, 88

[74] Ian Clarke. Freenet's next generation routing protocol, 2003. http://freenetproject.org/ngrouting.html. 36

[75] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46–??, 2001. 35

[76] Maurice Clerc. *Particle Swarm Optimization*. Wiley, 2006. 50

[77] Edith Cohen, Amos Fiat, and Haim Kaplan. Associative search in peer to peer networks: Harnessing latent semantics, 2003. 33

[78] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):177–190, 2002. 32

[79] R. Cohen and S. Havlin. Scale-Free Networks Are Ultrasmall. *Phys. Rev. Lett.*, 90:058701, 2003. 28

[80] Reuven Cohen, Shlomo Havlin, and Daniel Ben-avraham. Structural properties of scale-free networks. In *In Handbook of Graphs and Networks*, pages 85–110. Wiley, 2002. 28

[81] P V Coveney, R S Saksena, S J Zasada, M McKeown, and S Pickles. The application hosting environment: Lightweight middleware for grid-based computational science. Technical Report physics/0609168, Sep" 2006. 106

[82] Russ Cox, Athicha Muthitacharoen, and Robert Morris. Serving dns using chord. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002. 15

[83] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23, Washington, DC, USA, 2002. IEEE Computer Society. 31, 88

[84] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems, 2002. 33

[85] Paolo Crucitti, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. Efficiency of scale-free networks: error and attack tolerance. *Physica A: Statistical Mechanics and its Applications*, 320:622–642, March 2003. 28

[86] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing, 2001. 42

[87] Luciano da F. Costa, Francisco A. Rodrigues, Gonzalo Travieso, and P. R. Villas Boas. Characterization of complex networks: A survey of measurements. *Advances In Physics*, 56:167, 2007. 27

[88] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001. 15

[89] Anwitaman Datta. Mobigrid: Peer-to-peer overlay and mobile ad-hoc network rendezvous - a data management perspective. In *the 15th Conference On Advanced Information Systems Engineering*, 2003. 10

[90] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range queries in trie-structured overlays. *Peer-to-Peer Computing, IEEE International Conference on*, 0:57–66, 2005. 26

[91] Ze Deng, Dan Feng, Ke Zhou, Zhan Shi, and Chao Luo. Range query using learning-aware rps in dht-based peer-to-peer networks. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 180–187, Washington, DC, USA, 2009. IEEE Computer Society. 26

[92] A. Detsch, L. P. Gaspary, M. P. Barcellos, and G. G. H. Cavalheiro. Towards a flexible security framework for peer-to-peer based grid computing. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 52–56, New York, NY, USA, 2004. ACM. 44

[93] Edsger Wybe Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerical Mathematics*, 1:269–271, 1959. `http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf` – last visited: 27^{th} May 2008. 84

[94] Vassilios V. Dimakopoulos and Evaggelia Pitoura. On the performance of flooding-based resource discovery. *IEEE Transactions on Parallel and Distributed Systems*, 17:1242–1252, 2006. 28, 31

[95] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, 2006. 3

[96] F. Dong, S. G. Akl, and O. Kingston. Scheduling algorithms for grid computing: State of the art and open problems. Technical report, School of Computing, Queen's University, Kingston, Ontario, 2006. 108

[97] Marco Dorigo and Luca Maria Gambardella. Ant colonies for the traveling salesman problem, 1997. 51

[98] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Book, 2004. 3, 50

[99] Kay Dörnemann, Jörg Prenzer, and Bernd Freisleben. A peer-to-peer meta-scheduler for service-oriented grid environments. In *GridNets '07: Proceedings of the first international conference on Networks for grid applications*, pages 1–8, ICST, Brussels, Belgium, 2007. 109

[100] Peter Druschel and Antony Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *In HotOS VIII*, pages 75–80, 2001. 18

[101] Philippe Duchon, Nicolas Hanusse, Emmanuelle Lebhar, and Nicolas Schabanel. Towards small world emergence. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 225–232, New York, NY, USA, 2006. ACM. 28

[102] T.N. Ellahi, B. Hudzia, L. Mcdermott, and T. Kechadi. Security framework for p2p based grid systems. *Parallel and Distributed Computing, International Symposium on*, 0:230–237, 2006. 44

[103] Deger Cenk Erdil and Michael J. Lewis. Grid resource scheduling with gossiping protocols. In *Peer-to-Peer Computing, 2007. P2P 2007. Seventh IEEE International Conference on*, pages 193–202, Sept. 2007. 109

[104] Değer Cenk Erdil and Michael J. Lewis. Supporting self-organization for hybrid grid resource scheduling. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1981–1986, New York, NY, USA, 2008. ACM. 108

[105] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960. 27

[106] G. Exoo and R. Jajcay. Dynamic cage survey. *The Electronic Journal of Combinatorics*, (DS16), 2008. 47

[107] Heylighen F. and C. Gershenson. The meaning of self-organization in computing. *IEEE Intelligent Systems*, 18:72–75, 2003. 1

[108] Muddassar Farooq and Gianni A. Di Caro. Routing protocols for next-generation networks inspired by collective behaviors of insect societies: An overview. In Christian Blum and Daniel Merkle, editors, *Swarm Intelligence*, Natural Computing Series, pages 101–160. Springer, 2008. 33

[109] Marco Fiscato, Paolo Costa, and Guillaume Pierre. On the feasibility of decentralized grid scheduling. In *Proceedings of the Workshop on Decentralized Self-Management for Grids, P2P, and User Communities (Selfman)*, Venice, Italy, October 2008. 108

[110] Martin Flechl and Laurence Field. Grid interoperability: joining grid information systems. In *Proceedings of CHEP07*, 2007. 107

[111] George H. L. Fletcher and Hardik A. Sheth. Unstructured peer-to-peer networks: Topological properties and search performance. In *Third International Joint Conference on Autonomous Agents and MUlti-Agent Systems. W6: Agents and Peerto-Peer Computing*, pages 14–27. Springer, 2004. 32

[112] Alexander Fölling, Christian Grimme, Joachim Lepping, and Alexander Papaspyrou. Decentralized grid scheduling with evolutionary fuzzy systems. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 16–36. Springer Verlag, 2009. Lect. Notes Comput. Sci. vol. 5798. 107, 108

[113] Agostino Forestiero, Carlo Mastroianni, and Michela Meo. Self-chord: A bio-inspired algorithm for structured p2p systems. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 44–51, Washington, DC, USA, 2009. IEEE Computer Society. 15, 33, 51

[114] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. Antares: an ant-inspired p2p information system for a self-structured grid. In *BIONETICS 2007 - 2nd Int. Conf. on Bio-Inspired Models of Network, Information, and Computing Systems*, Budapest, Hungary, December 2007. 15, 33, 88

[115] Open Grid Forum. Job submission description language (jsdl), http://www.ggf.org/documents/gfd.56.pdf. November 2005. 110

[116] Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. pages 118–128. 2003. 42, 106

[117] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996. 129, 131

[118] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997. 106, 108

[119] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edition, December 2003. 1, 105, 106

[120] Ian T. Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, pages 1–4, London, UK, 2001. Springer-Verlag. 106

[121] Fulvio Frapolli, Amos Brocco, Apostolos Malatras, and Béat Hirsbrunner. Flexiblerules: A player oriented board game development framework. In *The Third International Conferences on Advances in Computer-Human Interactions, ACHI, IEEE*, 2010. 135

[122] Charles P. Fry and Michael K. Reiter. Really truly trackerless bittorrent, 2006. 23

[123] A. Ganesh, A. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols, 2003. 10

[124] Luis Garcés-Erice and Ernst W. Biersack. Multi+: A robust and topology-aware peer-to-peer multicast service. *Comput. Commun.*, 29(7):900–910, 2006. 29, 32

[125] Cecile Germain-Renaud, Charles Loomis, Jakub T. Moscicki, and Romain Texier. Scheduling for responsive grids. *Journal of Grid Computing*, 6:15–27, 2008. 106

[126] Carlos Gershenson, Francis Heylighen, and Centrum Leo Apostel. When can we call a system self-organizing. In *In Advances in Artificial Life, 7th European Conference, ECAL 2003 LNAI 2801*, pages 606–614. Springer-Verlag, 2003. 1

[127] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *INFOCOM*, 2004. 30

[128] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *INFOCOM*, pages 1526–1537, 2005. 31

[129] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Perform. Eval.*, 63(3):241–263, 2006. 31

[130] P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. *SIGCOMM Comput. Commun. Rev.*, 36(4):147–158, 2006. 41

[131] Li Gong. Jxta: A network programming environment. *IEEE Internet Computing*, 5:88–95, 2001. 132

[132] A. González-Beltrán, P. Milligan, and P. Sage. Range queries over skip tree graphs. *Comput. Commun.*, 31(2):358–374, 2008. 26

[133] Matthew Green. Napster opens pandora's box: Examining how file-sharing services threaten the enforcement of copyright on the internet. *Ohio State Law Journal*, 63(799), 2002. 33

[134] Christian Grimme, Joachim Lepping, and Alexander Papaspyrou. Prospects of collaboration between compute providers by means of job interchange. In *Job Scheduling Strategies for Parallel Processing*, pages 132–151, 2007. 107

[135] Jean-Loup Guillaume, Matthieu Latapy, and Clémence Magnien. Comparison of failures and attacks on random and scale-free networks. In *OPODIS*, pages 186–196, 2004. 28

[136] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM. 41

[137] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03*, 2003. 25

[138] Systems A. Gupta, Abhishek Gupta, Divyakant Agrawal, and Amr E. Abbadi. Approximate range selection queries in peer-to-peer. In *In CIDR*, 2002. 26

[139] Junghee Han, David Watson, and Farnam Jahanian. Topology aware overlay networks. In *INFOCOM*, pages 2554–2565, 2005. 32

[140] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, 2003. 23, 26

[141] M. Hauswirth and R. Schmidt. An overlay network for resource discovery in grids. In *Proceedings of Sixteenth Workshop on Database and Expert Systems Applications, 2005*, pages 343–348, August 2005. 43

[142] Manfred Hauswirth and Roman Schmidt. An overlay network for resource discovery in grids. In *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*, pages 343–348, Aug. 2005. 107

[143] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. Insights into pplive: A measurement study of a large-scale p2p iptv system. In *In Proc. of IPTV Workshop, International World Wide Web Conference*, 2006. 10

[144] Felix Heine, Matthias Hovestadt, and Odej Kao. Towards ontology-driven p2p grid resource discovery. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 76–83, Washington, DC, USA, 2004. IEEE Computer Society. 42

[145] Francis Heylighen. The science of self-organization and adaptivity. In *In The Encyclopedia of Life Support Systems (EOLSS), Knowledge Management, Organizational Intelligence and Learning, and Complexity. Developed under the Auspices of the UNESCO, Eolss Publishers*, 2003. 1, 2

[146] Bryan Horling, Victor Lesser, and Regis Vincent. Multi-agent system simulation framework. *16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation*, August 2000. 132

[147] Xin-Mao Huang, Cheng-Yue Chang, and Ming-Syan Chen. Peercluster: A cluster-based peer-to-peer system. *IEEE Trans. Parallel Distrib. Syst.*, 17(10):1110–1123, 2006. 33

[148] Ye Huang, Nik Bessis, Amos Brocco, Pierre Kuonen, and Béat Hirsbrunner. Critical friend model: A vision towards inter-cooperative grid communities. *International Journal On Advances in Intelligent Systems (IJAIS), IARIA*, 2010. Accepted. 130, 131, 140

[149] Ye Huang, Amos Brocco, Michèle Courant, Béat Hirsbrunner, and Pierre Kuonen. Magate simulator: a simulation environment for a decentralized grid scheduler. In *International Conference on Advanced Parallel Processing Technologies*, volume 5737/2009, pages 273–287. APPT'09, Springer LNCS, August 2009. 130

[150] Ye Huang, Amos Brocco, Béat Hirsbrunner, Michèle Courant, and Pierre Kuonen. Smartgrid: A fully decentralized grid scheduling framework supported by swarm intelligence. In *7th International Conference on Grid and Cooperative Computing*, pages 160–168. GCC2008, October 2008. 4

[151] Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi. A peer-to-peer approach to resource location in grid environments. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 419, Washington, DC, USA, 2002. IEEE Computer Society. 2, 43, 87, 88

[152] Adriana Iamnitchi and Ian T. Foster. On fully decentralized resource discovery in grid environments. In *GRID '01: Proceedings of the Second International Workshop on Grid Computing*, pages 51–62, London, UK, 2001. Springer-Verlag. 42

[153] Adriana Iamnitchi, Matei Ripeanu, and Ian T. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 232–241, London, UK, 2002. Springer-Verlag. 32

[154] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, New York, NY, USA, 1998. ACM. 26

[155] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: a decentralized peer-to-peer web cache. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 213–222, New York, NY, USA, 2002. ACM. 18

[156] Svante Janson. Random graphs. Wiley, 2000. 27

[157] Márk Jelasity and Ozalp Babaoglu. T-man: Gossip-based overlay topology management. In *The Fourth International Workshop on Engineering Self-Organizing Applications (ESOA'06)*, Hakodate, Japan, May 2006. 40

[158] Mark Jelasity, Wojtek Kowalczyk, and Maarten Van Steen. Newscast computing. Technical report, Vrije Universiteit, Amsterdam. Internal Report IR-CS-006, 2003. 39, 61, 79

[159] Mark Jelasity and Maarten Van Steen. Large-scale newscast computing on the internet. Technical report, Vrije Universiteit, Amsterdam. Internal Report IR-503, 2002. 39, 79, 92

[160] Jinyang Li Jeremy, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn, 2004. 41

[161] Gian Paolo Jesi, Alberto Montresor, and Ozalp Babaoglu. Proximity-aware superpeer overlay topologies. In *In Proc. of SelfMan'06, LNCS 3996*, pages 43–57. Springer-Verlag, 2006. 32

[162] H. Jiang and S. Jin. Exploiting dynamic querying like flooding techniques in unstructured peer-to-peer networks. In *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, page 10 pp., nov. 2005. 30

[163] Song Jiang, Lei Guo, Xiaodong Zhang, and Haodong Wang. Lightflood: Minimizing redundant messages and maximizing scope of peer-to-peer search. *IEEE Transactions on Parallel and Distributed Systems*, 19:601–614, 2008. 32

[164] Hai Jin. Challenges of Grid Computing. In *WAIM*, pages 25–31, 2005. 106

[165] Hai Jin, Yi Pan, Nong Xiao, and Jianhua Sun, editors. *Grid and Cooperative Computing - GCC 2004: Third International Conference, Wuhan, China, October 21-24, 2004. Proceedings*, volume 3251 of *Lecture Notes in Computer Science*. Springer, 2004. 106

[166] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *IPTPS*, pages 98–107, 2003. 15

[167] V. Kalogeraki, D. Gunopulos, and Zeinalipour D. Yazti. A local search mechanism for peer-to-peer networks. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 300–307. ACM Press, 2002. 29, 31

[168] Ibrahim Kamel, Zaher Al Aghbari, and Ahmed Mustafa. Efficient range queries in spatial databases over peer-to-peer networks. *Int. J. Internet Protoc. Technol.*, 4(2):79–90, 2009. 26

[169] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM. 14

[170] I. Kassabalidis, El M. A. Sharkawi, R. J. Marks, P. Arabshahi, and A. A. Gray. Swarm intelligence for routing in communication networks, November 2001. 51

[171] Stuart A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, USA, 1 edition, June 1993. 1

[172] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003. 3

[173] Anne-Marie Kermarrec and Maarten van Steen. Gossiping in distributed systems. *SIGOPS Oper. Syst. Rev.*, 41(5):2–7, 2007. 40

[174] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *in Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000. 25, 28

[175] Jon Kleinberg. Complex networks and decentralized search algorithms. In *International Congress of Mathematicians (ICM)*, 2006. 28

[176] John R. Koza. Introduction to genetic programming. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 2, pages 21–42. MIT Press, Cambridge, MA, USA, 1994. 3

[177] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *FOSE '07: 2007 Future of Software Engineering*, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society. 3

[178] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience*, 32:135–164, 2002. 42

[179] Barbara Kreaseck, Larry Carter, Henri Casanova, and Jeanne Ferrante. Autonomous protocols for bandwidth-centric scheduling of independent-task applications. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 26.1, Washington, DC, USA, 2003. IEEE Computer Society. 108

[180] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGARCH Comput. Archit. News*, 28(5):190–201, 2000. 19

[181] Minseok Kwon and Sonia Fahmy. Topology-aware overlay networks for group communication. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 127–136, New York, NY, USA, 2002. ACM. 29, 32

[182] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the kazaa network. In *WIAPP '03: Proceedings of the The Third IEEE Workshop on Internet Applications*, page 112, Washington, DC, USA, 2003. IEEE Computer Society. 37

[183] Derek Leonard, Zhongmei Yao, Vivek Rai, and Dmitri Loguinov. On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. *IEEE/ACM Trans. Netw.*, 15(3):644–656, 2007. 41

[184] Elias Leontiadis, Vassilios V. Dimakopoulos, and Evaggelia Pitoura. Cache updates in a peer-to-peer network of mobile agents. In *4th International Conference on Peer-to-Peer Computing (P2P'04)*, pages 10–17, 2004. 31

[185] Lun Li, David Alderson, Reiko Tanaka, John C. Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications (extended version). Oct 2005. 28

[186] Mei Li, Wang-chien Lee, and Anand 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. 26, 43

[187] Xing Li and Jiguo Yu. A novel p2p overlay network based on cycloid and folded hypercube. In *GCC '08: Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing*, pages 374–379, Washington, DC, USA, 2008. IEEE Computer Society. 25

[188] Xiuqi Li and Jie Wu. A hybrid searching scheme in unstructured p2p networks. *Int. J. Parallel Emerg. Distrib. Syst.*, 22(1):15–38, 2007. 33, 139

[189] Jian Liang, Rakesh Kumar, and Keith W. Ross. The fasttrack overlay: a measurement study. *Comput. Netw.*, 50(6):842–858, 2006. 37

[190] Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M. Ni, and Dafu Deng. Anysee: Peer-to-peer live streaming. In *Proc. of INFOCOM*, April 2006. 10

[191] Prakash Linga, Indranil Gupta, and Ken Birman. Kache: Peer-to-peer web caching using kelips. In *In In submission*, 2004. 25

[192] Bin Liu, Wang-Chien Lee, and Dik Lun Lee. Supporting complex multi-dimensional queries in p2p systems. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 155–164, Washington, DC, USA, 2005. IEEE Computer Society. 26

[193] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005. 13

[194] D.J. Lutz, P. Mandic, S. Neinert, R. del Campo, and J. Jaehnert. Harmonizing service and network provisioning for federative access in a mobile environment. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 843–846, April 2008. 106

[195] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 258–259, New York, NY, USA, 2002. ACM. 29, 30, 31, 32, 88

[196] Jens Mache, Melanie Gilbert, Jason Guchereau, Jeff Lesh, Felix Ramli, and Matthew Wilkinson. Request algorithms in freenet-style peer-to-peer systems. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 90, Washington, DC, USA, 2002. IEEE Computer Society. 29

[197] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM. 19

[198] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: distributed hashing in a small world. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association. 25

[199] M. Marzolla, M. Mordacchini, and S. Orlando. Resource discovery in a dynamic grid environment. In *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*, pages 356–360, Aug. 2005. 30

[200] Carlo Mastroianni, Domenico Talia, and Oreste Verta. A super-peer model for resource discovery services in large-scale grids. *Future Gener. Comput. Syst.*, 21(8):1235–1248, 2005. 43

[201] Carlo Mastroianni, Domenico Talia, and Oreste Verta. Evaluating resource discovery protocols for hierarchical and super-peer grid information systems. In *PDP '07: Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 147–154, Washington, DC, USA, 2007. IEEE Computer Society. 43

[202] Carlo Mastroianni, Domenico Talia, and Oreste Verta. Designing an information system for grids: Comparing hierarchical, decentralized p2p and super-peer models. *Parallel Comput.*, 34(10):593–611, 2008. 43

[203] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65, Berlin, Heidelberg, October 2002. Springer Berlin Heidelberg. 22

[204] Elke Michlmayr. Self-organization for search in peer-to-peer networks: the exploitation-exploration dilemma. In *BIONETICS '06: Proceedings of the 1st international conference on Bio inspired models of network, information and computing systems*, page 29, New York, NY, USA, 2006. ACM. 33, 51

[205] Milena Mihail, Amin Saberi, and Prasad Tetali. Random walks with lookahead in power law random graphs, available at http://www.cc.gatech.edu/fac/milena.mihail/lookahead.pdf. *Internet Mathematics*, 3:2007, 2004. 31

[206] Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967. 25, 28

[207] M. Miller and J. Siran. Moore graphs and beyond: A survey of the degree/diameter problem. *Electronic Journal of Combinatorics*, (DS14), 2005. 47

[208] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. May 2004. 28

[209] Nelson Minar, Rogert Burkhart, Chris Langton, and Manor Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. Working Papers 96-06-042, Santa Fe Institute, June 1996. 132

[210] Alberto Montresor and Hein Meling. Messor: Load-balancing through a swarm of autonomous agents. In *In Proceedings of 1st Workshop on Agent and Peer-to-Peer Systems*, pages 125–137, 2002. 3, 5, 51, 132

[211] Richard Mortier and Emre Kiciman. Autonomic network management: some pragmatic considerations. In *INM '06: Proceedings of the 2006 SIGCOMM workshop on Internet network management*, pages 89–93, New York, NY, USA, 2006. ACM. 3

[212] Rossana Motta, Wickus Nienaber, and Jon Jenkins. Gnutella: integrating performance and security in fully decentralized p2p models. In *ACM-SE 46: Proceedings of the 46th Annual Southeast Regional Conference on XX*, pages 272–277, New York, NY, USA, 2008. ACM. 35

[213] M. Newman. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, December 1999. 28

[214] M. E. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys Rev E Stat Nonlin Soft Matter Phys*, 64(2 Pt 2), August 2001. 28

[215] M. E. J. Newman. Models of the small world. *Journal of Statistical Physics*, 101(3):819–841, November 2000. 28

[216] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(Suppl 1):2566–2572, February 2002. 28

[217] Graham R. Nudd, Darren J. Kerbyson, Efstathios Papaefstathiou, Stewart C. Perry, John S. Harper, and Daniel V. Wilcox. Pace–a toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000. 108, 111

[218] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable wide-area resource discovery, 2004. 26, 43

[219] Francis Otto and Song Ouyang. Improving search in unstructured p2p systems: Intelligent walks (i-walks). In Emilio Corchado, Hujun Yin, Vicente J. Botti, and Colin Fyfe, editors, *Proceedings of the 7^{th} International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'06)*, volume 4224 of *Lecture Notes in Computer Science*, pages 1312–1319. Springer, september 2006. 31

[220] Esther Pacitti, Patrick Valduriez, and Marta Mattoso. Grid data management: Open problems and new issues. *J. Grid Comput.*, 5(3):273–281, 2007. 107

[221] M. Parashar and C. Lee. Grid computing - introduction and overview. *Proceedings of the IEEE: Special Issue on Grid Computing*, 93(3), March 2005. 2

[222] Charles E. Perkins. Ad hoc networking: an introduction. pages 1–28, 2001. 1

[223] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA, 1997. ACM. 16

[224] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system: Research articles. *Concurr. Comput. : Pract. Exper.*, 20(2):127–138, 2008. 32

[225] Franco P. Preparata and Jean Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Commun. ACM*, 24(5):300–309, 1981. 25

[226] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990. 23

[227] Diego Puppin, Stefano Moncelli, Ranieri Baraglia, Nicola Tonellotto, and Fabrizio Silvestri. A grid information service based on peer-to-peer. In José C. Cunha and Pedro D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 2005. 88

[228] I.Gruber R. Schollmeier and F. Niethammer. Protocol for peer-to-peer networking in mobile environments. In *Proc. of the 12th Int'l Conf. on Computer Communications and Networks*, pages 121–127, 2003. 10

[229] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association. 32

[230] M. Rambadt and Ph. Wieder. Unicore - globus interoperability: Getting the best of both worlds. In *Proceedings of 11th International Symposium on High Performance Distributed Computing (HPDC)*, page 422, Edinburgh, Scotland, 2002. IEEE Computer Society Press. 107

[231] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. A study on peer-to-peer based discovery of grid resource information. Technical report, 2006. 43

[232] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. Peer-to-peer-based resource discovery in global grids: a tutorial. *Communications Surveys & Tutorials, IEEE*, 10(2):6–33, 2008. 108

[233] Weixiong Rao, Lei Chen, Ada Wai-Chee Fu, and YingYi Bu. Optimal proactive caching in peer-to-peer network: analysis and application. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 663–672, New York, NY, USA, 2007. ACM. 32

[234] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM. 21

[235] Sylvia Ratnasamy, Sylvia Ratnasamy, Joseph M. Hellerstein, Joseph M. Hellerstein, Scott Shenker, and Scott Shenker. Range queries over dhts, 2003. 26

[236] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association. 41

[237] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, New York, NY, USA, 2005. ACM. 19, 26

[238] M. Ripeanu. [15] peer-to-peer architecture case study: Gnutella network. *Peer-to-Peer Computing, IEEE International Conference on*, 0:0099, 2001. 33, 35

[239] M. Ripeanu, A. Iamnitchi, I. Foster, and A. Rogers. In search of simplicity: A self-organizing group communication overlay. In *Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on*, pages 371–374, July 2007. 30, 38

[240] Matei Ripeanu, Ian Foster, Adriana Iamnitchi, and Anne Rogers. Umm: A dynamically adaptive, unstructured multicast overlay. 38

[241] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: search methods. *Computer Networks*, 50(17):3485–3521, 2006. 29

[242] Luis Rodero-Merino, Antonio F. Anta, Luis López, and Vicent Cholvi. Performance of random walks in one-hop replication networks. *Computer Networks*, October 2009. 32

[243] Mathilde Romberg. The unicore architecture: seamless access to distributed resources. In *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pages 287–293, 1999. 106, 108

[244] Habib Rostami, Jafar Habibi, and Ali Rahnama. Semantic hypercup. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 223.1, Washington, DC, USA, 2006. IEEE Computer Society. 25

[245] A. Rotem-Gal-Oz. Fallacies of distributed computing explained. http://www.rgoarchitects.com/files/fallacies.pdf, 2006. 1, 2

[246] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001. 16

[247] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994. 26

[248] Supriya Krishnamurthy Sameh, Sameh El-ansary, Erik Aurell, and Seif Haridi. A statistical theory of chord under churn. In *In Proceedings of the 4th IPTPS*, pages 93–103, 2005. 41

[249] Oskar Sandberg. Decentralized search with random costs. *CoRR*, abs/0804.0577, 2008. 28, 35, 36

[250] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. January 2002. 41

[251] Mario T. Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. Hypercup - hypercubes, ontologies, and efficient search on peer-to-peer networks. In *AP2PC*, pages 112–124, 2002. 25

[252] Stefan Schmid and Roger Wattenhofer. Structuring unstructured peer-to-peer networks. In *High Performance Computing HiPC 2007*, 2007. 40

[253] Jennifer M. Schopf. Ten actions when grid scheduling: the user as a grid scheduler. pages 15–23, 2004. 106

[254] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. A structured overlay for multi-dimensional range queries. pages 503–513. 2007. 26

[255] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. Range queries on structured overlay networks. *Comput. Commun.*, 31(2):280–291, 2008. 26

[256] Ruchir Shah, Bhardwaj Veeravalli, and Manoj Misra. On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments. *IEEE Transactions on Parallel and Distributed Systems*, 18(12):1675–1686, 2007. 108

[257] Anju Sharma and Seema Bawa. Comparative analysis of resource discovery approaches in grid computing. *JCP*, 3(5):60–64, 2008. 42

[258] Haiying Shen, Amy Apon, and null Cheng-Zhong Xu. Lorm: Supporting low-overhead p2p-based range-query and multi-attribute resource management in grids. *Parallel and Distributed Systems, International Conference on*, 1:1–8, 2007. 43

[259] Haiying Shen, Ze Li, Ting Li, and Yingwu Zhu. Pird: P2p-based intelligent resource discovery in internet-based distributed systems. In *Distributed Computing Systems, 2008. ICDCS '08. The 28th International Conference on*, pages 858–865, June 2008. 107

[260] Haiying Shen and Cheng-Zhong Xu. Performance analysis of dht algorithms for range-query and multi-attribute resource discovery in grids. *Parallel Processing, International Conference on*, 0:246–253, 2009. 26, 43

[261] Haiying Shen, Cheng-Zhong Xu, and Guihai Chen. Cycloid: a constant-degree and lookup-efficient p2p overlay network. *Perform. Eval.*, 63(3):195–216, 2006. 25

[262] Kai Shen. Structure management for scalable overlay service construction. In *In NSDI*, pages 281–294, 2004. 37

[263] Kai Shen and Yuan Sun. Distributed hashtable on pre-structured overlay networks. Technical report, in 9th International Workshop on Web Caching and Content Distribution, 2004. 37

[264] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using sip. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 63–68, New York, NY, USA, 2005. ACM. 10

[265] Ray Solomonoff and Anatol Rapoport. Connectivity of random nets. *Bulletin of Mathematical Biology*, 13(2):107–117, June 1951. 27

[266] David Spence and Tim Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 216, Washington, DC, USA, 2003. IEEE Computer Society. 43

[267] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *IEEE Infocom*, San Francisco, CA, March 2003. 89

[268] Mudhakar Srivatsa, Bugra Gedik, and Ling Liu. Large scaling unstructured peer-to-peer networks with heterogeneity-aware topology and routing. *IEEE Transactions on Parallel and Distributed Systems*, 17:1277–1293, 2006. 32

[269] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 149–160, New York, USA, October 2001. ACM Press. 14

[270] Daniel Stutzbach and Reza Rejaie. Capturing accurate snapshots of the gnutella network. In *INFOCOM*, 2006. 35

[271] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM. 41

[272] Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan, and P. Sadayappan. Distributed job scheduling on computational grids using multiple simultaneous requests. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, USA, 2002. 107, 108

[273] Hailong Sun, Jinpeng Huai, Yunhao Liu, and Rajkumar Buyya. Rct: A distributed tree for supporting efficient range and multi-attribute queries in grid computing. *Future Gener. Comput. Syst.*, 24(7):631–643, 2008. 43

[274] Xiaoping Sun. Scan: a small-world structured p2p overlay for multi-dimensional queries. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1191–1192, New York, NY, USA, 2007. ACM. 21

[275] Domenico Talia, Paolo Trunfio, Jingdi Zeng, and Mikael Högqvist. A dht-based peer-to-peer framework for resource discovery in grids. Technical Report TR-0048, Institute on System Architecture, CoreGRID - Network of Excellence, June 2006. 3, 42, 43

[276] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice Hall, 2 edition, October 2006. 1

[277] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. psearch: information retrieval in structured overlays. *SIGCOMM Comput. Commun. Rev.*, 33(1):89–94, 2003. 26

[278] Saurabh Tewari and Leonard Kleinrock. Analysis of search and replication in unstructured peer-to-peer networks. *SIGMETRICS Perform. Eval. Rev.*, 33(1):404–405, 2005. 32

[279] Saurabh Tewari and Leonard Kleinrock. Proportional replication in peer-to-peer networks. In *INFOCOM*, 2006. 32

[280] Sabu M. Thampi and Chandra K. Sekaran. Review of replication schemes for unstructured p2p networks. Mar 2009. 32

[281] D. A. Tran and T. Nguyen. Hierarchical multidimensional search in peer-to-peer networks. *Comput. Commun.*, 31(2):346–357, 2008. 43

[282] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi. Peer-to-peer resource discovery in grids: Models and systems. *Future Gener. Comput. Syst.*, 23(7):864–878, 2007. 42, 43

[283] D. Tsoumakos and N. Roussopoulos. A comparison of peer-to-peer search methods, 2003. 29

[284] Dimitrios Tsoumakos and Nick Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 102, Washington, DC, USA, 2003. IEEE Computer Society. 32

[285] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005. 10, 40

[286] Spyros Voulgaris and Maarten Van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In *In Proceedings of the 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, (DSOM 2003), number 2867 in Lecture Notes in Computer Science*, pages 41–54. Springer, 2003. 62

[287] Spyros Voulgaris and Maarten van Steen. Epidemic-style management of semantic overlays for content-based searching. pages 1143–1152. 2005. 40

[288] Marcel Waldvogel and Roberto Rinaldi. Efficient topology-aware overlay network. *SIGCOMM Comput. Commun. Rev.*, 33(1):101–106, 2003. 32

[289] Dan S. Wallach. A survey of peer-to-peer security issues. In *International Symposium on Software Security*, pages 42–57, 2002. 2, 3

[290] Chen Wang and Li Xiao. An effective p2p search scheme to exploit file sharing heterogeneity. *IEEE Trans. Parallel Distrib. Syst.*, 18(2):145–157, 2007. 31

[291] Shihui Wang, Yan Zhang, and Wei Wang. Reliable self-clustering p2p overlay networks. *Computer Software and Applications Conference, Annual International*, 1:105–108, 2007. 33

[292] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998. 28

[293] Duncan J. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton & Company, February 2003. 28

[294] Douglas B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, August 2000. 11

[295] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator, technical report cse-tr-456-02. Technical report, University of Michigan, 2002. 60

[296] Chi-Jen Wu, Kai-Hsiang Yang, and Jan-Ming Ho. Antsearch: An ant search algorithm in unstructured peer-to-peer networks. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 429–434, Washington, DC, USA, 2006. IEEE Computer Society. 33

[297] Lijuan Xiao, Yanmin Zhu, Lionel M. Ni, and Zhiwei Xu. Gridis: An incentive-based grid scheduling. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2005. IEEE Computer Society. 109

[298] Yan Xu, XiaoJun Ma, and Charles Wang. Selective walk searching algorithm for gnutella network. In *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, pages 746 –750, jan. 2007. 32

[299] Zhichen Xu and Zheng Zhang. Building low-maintenance expressways for p2p systems. Technical report, 2002. 21

[300] Beverly Yang and Hector Garcia-Molina. Comparing hybrid peer-to-peer systems. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 561–570, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 13

[301] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *22nd Int. Conf. on Distributed Computing Systems (ICDCS'02)*, pages 5–13. IEEE, 2002. 88

[302] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. *Data Engineering, International Conference on*, 0:49, 2003. 29, 32

[303] Beverly Yang, Patrick Vinograd, and Hector Garcia-Molina. Evaluating guess and non-forwarding peer-to-peer search. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 209–218, Washington, DC, USA, 2004. IEEE Computer Society. 33, 139

[304] Zhongmei Yao, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. Modeling heterogeneous user churn and local resilience of unstructured p2p networks. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 32–41, Washington, DC, USA, 2006. IEEE Computer Society. 41

[305] Demetrios Zeinalipour-yazti. Exploiting the security weaknesses of the gnutella protocol. Technical report, University of California - Riverside, Computer Science, 2002. 35

[306] Demetrios Zeinalipour-Yazti, Vana Kalogeraki, and Dimitrios Gunopulos. Information retrieval techniques for peer-to-peer networks. *Computing in Science and Engg.*, 6(4):20–26, 2004. 29

[307] Bo Zhang, T. S. Eugene Ng, Animesh Nandi, Rudolf Riedi, Peter Druschel, and Guohui Wang. Measurement based analysis, modeling, and synthesis of the internet delay space. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 85–98, New York, NY, USA, 2006. ACM. 2

[308] Chi Zhang, Arvind Krishnamurthy, and Randolph Y. Wang. Skipindex: Towards a scalable peer-to-peer index service for high dimensional data. Technical report, Department of Computer Science, Princeton University, May 2004. 26, 35

[309] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004. 18

[310] Ming Zhong, Kai Shen, and Joel Seiferas. The convergence-guaranteed random walk and its applications in peer-to-peer networks. *IEEE Transactions on Computers*, 57:619–633, 2008. 31

[311] Yingwu Zhu, Xiaoyu Yang, and Yiming Hu. Making search efficient on gnutella-like p2p systems. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 56.1, Washington, DC, USA, 2005. IEEE Computer Society. 40

[312] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20, New York, NY, USA, 2001. ACM. 19

[313] Zhenyun Zhuang, Yunhao Liu, Li Xiao, and Lionel M. Ni. Hybrid periodical flooding in unstructured peer-to-peer networks. *Parallel Processing, International Conference on*, 0:171, 2003. 32

Curriculum Vitae

Amos Brocco

Personal

Born on October 7, 1981.

Swiss Citizen.

Languages: Italian (mother tongue), French, German, and English.

Academic Qualifications

October 2010 : PhD in Computer Science, University of Fribourg, Switzerland. Thesis title: "Exploiting self-organization for the autonomic management of distributed systems"; supervisor: Prof. Béat Hirsbrunner.

October 2005 : BsC and MsC in Computer Science, University of Fribourg, Switzerland. Thesis title: "RoXanne Framework: x.core and x.click components"; supervisor: Prof. Andreas Meier.

Education

2005 - 2010 : University of Fribourg, Switzerland. Ph.D Student at the "Pervasive and Artificial Intelligence research group", within the SmartGRID project. Research supported by the Swiss Halser Foundation in the framework of the "ManCom Initiative", project nr. 2122.

2000 - 2005 : University of Fribourg Switzerland. BsC and MsC studies in Computer Science (major) and Mathematics (minor).

1996 - 2000 : Scuola Cantonale di Commercio, Bellinzona, Switzerland. "Maturità Commerciale Cantonale".

Professional Experience

2008 - 2010 : Developer for the "Arbeitspsychologie und kognitive Ergonomie Forschungsgruppe at the Department of Psychology, University of Fribourg. Development and customization of the AutoCAMS platform.

2005 - 2010 : Teaching assistant at the Department of Informatics, University of Fribourg. Courses: Operating Systems, Distributed Systems, and Computer Architecture.

2006 : System Administrator (25%) at the Department of Informatics, University of Fribourg. Tasks: Workstation configuration, development.

2004 : Student assistant for the "Document, Image and Voice Analysis research group", at the Department of Psychology, University of Fribourg. Tasks: development.

2001, 2002 : Summer internship at Crédit Suisse Financial Services, Bellinzona, Switzerland. Tasks: Computer systems support, development of an intranet information portal.

Publication List

Journals

2010

Ye Huang, Nik Bessis, Amos Brocco, Pierre Kuonen and Béat Hirsbrunner, Critical Friend Model: A Vision Towards Inter-cooperative Grid Communities, *International Journal On Advances in Intelligent Systems (IJAIS)*, IARIA, 2010.

Ye Huang, Amos Brocco, Pierre Kuonen and Béat Hirsbrunner, MaGate: an interoperable, decentralized and modular high-level grid scheduler, *International Journal of Distributed Systems and Technologies (IJDST)*, IGI Global, 2010.

Fulvio Frapolli, Amos Brocco, Apostolos Malatras and Béat Hirsbrunner, Decoupling Aspects in Board Games Modeling, *International Journal of Gaming and Computer-Mediated Simulations (IJGCMS)*, volume II, number 2, 2010.

Amos Brocco, Apostolos Malatras and Béat Hirsbrunner, Enabling Efficient Information Discovery in a Self-Structured Grid, *Future Generation Computer Systems*, Pages 838-846, Elsevier, ISSN 0167-739X, 2010.

Amos Brocco and Béat Hirsbrunner, Service Provisioning For a Next-Generation Adaptive Grid, *The International Journal of Parallel, Emergent and Distributed Systems*, Taylor and Francis, ISSN 1744-5760, 2010.

Proceedings

2010

Amos Brocco, Apostolos Malatras, Ye Huang and Béat Hirsbrunner, ARiA: A Protocol for Dynamic Fully Distributed Grid Meta-Scheduling, *Proceedings of The 30th International Conference on Distributed Computing Systems*, ICDCS 2010, IEEE, Genoa, Italy, June, 2010.

Ye Huang, Amos Brocco, Nik Bessis, Pierre Kuonen and Béat Hirsbrunner, Community-Aware Scheduling Protocol for Grids, *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 334-341, AINA 2010, IEEE, April, 2010.

Fulvio Frapolli, Amos Brocco, Apostolos Malatras and Béat Hirsbrunner, FLEXIBLERULES: A Player Oriented Board Game Development Framework, *The Third International Conferences on Advances in Computer-Human Interactions*, pages 113-118, ACHI 2010, IEEE, 2010.

2009

Ye Huang, Nik Bessis, Amos Brocco, Stelios Sotiriadis, Michèle Courant , Pierre Kuonen and Béat Hirsbrunner, Towards an integrated vision across inter-cooperative grid virtual organization, *International Conference on Grid and Distributed Computing*, pages 120-128, Springer LNCS, GDC 2009, Jeju Island, Korea, December, 2009.

Ye Huang, Nik Bessis, Amos Brocco, Pierre Kuonen, Michèle Courant and Béat Hirsbrunner, Using Metadata Snapshots for Extending Ant-based Resource Discovery Service in Inter-cooperative Grid Communities, *International Conference on Evolving Internet*, pages 89-94, INTERNET 2009, IEEE, Cannes/La Bocca, French Riviera, France, August, 2009.

Ye Huang, Amos Brocco, Michèle Courant , Béat Hirsbrunner and Pierre Kuonen, MaGate Simulator: a simulation environment for a decentralized grid scheduler, *International Conference on Advanced Parallel Processing Technologies*, pages 273-287, Springer LNCS, APPT'09, Rapperswil, Switzerland, August, 2009.

Amos Brocco and Béat Hirsbrunner, Service Provisioning Framework for a Self-Organized Grid, *Workshop on Grid and P2P Systems and Applications (GridPeer 2009)*, pages 1-6, ICCCN 09, IEEE, San Francisco, CA USA, August, 2009.

Amos Brocco, Apostolos Malatras and Béat Hirsbrunner, Proactive Information Caching for Efficient Resource Discovery in a Self-Structured Grid, *Workshop on Bio-Inspired Algorithms for Distributed Systems*, pages 11-18, ACM, ICAC 2009, Barcelona, Spain, June, 2009.

Amos Brocco, Fulvio Frapolli and Béat Hirsbrunner, Bounded Diameter Overlay Construction: A Self Organized Approach, *IEEE Swarm Intelligence Symposium*, pages 114-121, IEEE, SIS 2009, Nashville, Tennessee, USA, April, 2009.

2008

Ye Huang, Amos Brocco, Béat Hirsbrunner, Michèle Courant and Pierre Kuonen, SmartGRID: A fully decentralized Grid Scheduling Framework supported by Swarm Intelligence, *7th International Conference on Grid and Cooperative Computing*, pages 160-168, GCC2008, Shenzhen, China, October, 2008.

Amos Brocco, Fulvio Frapolli and Béat Hirsbrunner, BlatAnt: Bounding Networks' Diameter with a Collaborative Distributed Algorithm, *Sixth International Conference on Ant Colony Optimization and Swarm Intelligence*, pages 275-282, Springer, ANTS, Bruxelles, September, 2008.

2007

Amos Brocco, Béat Hirsbrunner and Michèle Courant , Solenopsis: A Framework for the Development of Ant Algorithms, *Swarm Intelligence Symposium*, pages 316-323, IEEE, SIS, Honolulu, Hawaii, April, 2007.

Amos Brocco, Béat Hirsbrunner and Michèle Courant , A Modular Middleware for High-level Dynamic Network Management, *1st Workshop on Middleware-Application Interaction*, pages 21-24, ACM, MAI, EuroSys, Lisbon, Portugal, March, 2007.

2006

Vincenzo Pallotta, Amos Brocco, Dominique Guinard, Pascal Bruegger and Pedro De Almeida, RoamBlog: Outdoor and Indoor Geoblogging Enhanced with Contextual Service Provisioning for Mobile Internet Users, *Proceedings of the 1st International workshop on Distributed Agent-based Retrieval Tools*, pages 103-121, Polimetrica International Scientific Publisher, DART, ISBN 88-7699-043-7, June, 2006.

Technical Reports

2008

Amos Brocco, Fulvio Frapolli and Béat Hirsbrunner, Shrinking the Network: The BlatAnt Algorithm, *Technical Report, number 08-04, Department of Informatics, University of Fribourg*, April, 2008.