Bounded Diameter Overlay Construction: A Self Organized Approach

Amos Brocco, Fulvio Frapolli, and Béat Hirsbrunner

Abstract— **This paper describes a distributed algorithm to construct and maintain a peer-to-peer network overlay with bounded diameter. The proposed approach merges a bio-inspired self-organized behavior with a pure peer-to-peer approach, in order to adapt the overlay to underlying changes in the network topology. Ant colonies are used to collect and spread information across all peers, whereas pheromone trails help detecting crashed nodes. Construction of the network favors balanced distribution of links across all peers, so that the resulting topology does not exhibit large hubs. Fault resilience and recovery mechanisms have also been implemented to prevent network partition in the event of node crashes. Validation has been conducted through simulations of different network scenarios.**

I. Introduction

As the computing power and the network bandwidth available to the majority of computers increases, we observe a constant shift from centralized designs to fully decentralized systems and applications. From a user perspective, this evolution is mainly reflected in the growth of peer-to-peer file sharing networks and distributed computing projects. This transition started with networks that are now commonly referred as unstructured *pure* peer-to-peer systems, because no peer hierarchy exists, and each node is considered to have equal capabilities. On one side, those fully decentralized systems help managing resources that are, by nature, distributed (e.g. media files shared by different computers). On the other side, they support the goal of improving robustness and availability, by moving resources outside centralized data centers into several computers distributed across the planet.

A major difference between peer-to-peer systems and more traditional client-server systems is that the exact location of the information or service provider is unknown to the user. Thus, the main challenge in the development of such applications is to provide efficient methods for information retrieval, given that central indexes are generally to be avoided. The problem of locating resources in large-scale, fully decentralized and unstructured peer-to-peer systems can be traced back to the birth of the GNUTELLA [1] network. Although early fully distributed network applications, such as USENET, had been able to solve the issue by simply replicating indexes across all nodes, the large scales of today's networks (in respect of both size of the network, and the size of shared contents) make such solutions impractical.

Amos Brocco, Fulvio Frapolli, and Béat Hirbrunner are with the Department of Informatics, University of Fribourg, Switzerland (email: {amos.brocco, fulvio.frapolli, beat.hirsbrunner}@unifr.ch).

This research is supported by the Swiss Hasler Foundation ("ManCom Initiative", project Nr. 2122).

Resource discovery in decentralized systems is typically performed using flooding protocols or random walks [2]: search queries are forwarded by each node to its neighbors, up to a specified distance. Unfortunately, the explosion of large scale sharing applications has brought to the surface the inherent scalability limits of these methods, which are responsible for a considerable fraction of all the traffic generated by peer-to-peer networks [3].

To solve these issues, development in this field shifted away from the initial and conceptually simpler approaches, and moved to more complex but efficient solutions. One research direction tackled the problem of mapping the information on the network by means of structured topologies [4], [5]: managing large scale networks without large hubs or bottlenecks has been made possible by enforcing strict rules for the construction of the system and for the distribution of information across nodes. Unfortunately, many structured systems, such as distributed hash tables, are only suitable for exact match queries, which limits their usage for generic resource sharing networks.

Another direction taken to improve search efficiency while retaining some of the advantages of unstructured networks introduces the concept of super-peers: nodes with more capabilities and better connectivity (referred as *super-peers* [6]) are exploited to perform additional tasks, such as routing and extensive caching of information. Super-peer networks can be considered as hybrids between unstructured and structured systems, combining the simplicity of the former with the scalability of the latter. The concept of super-peers is currently implemented by many popular file sharing networks, such as KAZAA [7], as well as by recent revisions of the GNUTELLA protocols (with *ultrapeers*). Search efficiency of super-peer networks is improved by the fact that a large number of peers (*leafs*) connects to only a smaller number of super-peers, as in a client-server model. This property ensures that the resulting networks exhibit small diameters, which can be exploited by existing search methods to limit network overhead.

Small diameters can be obtained even in completely unstructured networks through the optimization of connections between peers, without introducing hierarchies or hubs. In particular, the algorithm proposed in this paper, named BLÅTANT-R, focuses on the construction of this kind of optimized overlays where connections are re-arranged dynamically to ensure both short distances between every pair of nodes in the system, and a fair distribution of links. Building, optimization, and maintenance of the network is based on fully distributed swarm intelligence methods, to

ensure adaptability and robustness. Although the algorithm targets unstructured pure peer-to-peer networks, several other usage scenarios can be considered, such as construction of optimized topologies connecting super-peers or nodes in a computational grid. It is important to note that the focus of this paper is solely on the construction and maintenance of the peer-to-peer overlay, leaving the implementation of efficient search methods as a separate research topic. In this paper we will thus avoid further consideration of existing search methods for peer-to-peer networks; the interested reader may find a detailed survey on this topic in [2], [8].

The rest of this paper is organized as follows: Section II presents some related works in the field of peer-to-peer overlay construction, Section III details the algorithm, Sections IV and V present the evaluation scenarios, respectively the results obtained. Finally, Section VI summarizes the work done and provides some insights on future work. As a peer-to-peer network can also be viewed as a directed graph, for the remaining of this paper the terms *peer* and *node* will be used interchangeably.

II. Related Work

We argue that decentralized search methods can benefit from an optimized overlay topology, with increased efficiency both in terms of bandwidth, and of response time. Thus, before introducing other projects that are conceptually close to the research presented in this paper, we shall define our areas of interest. On one side, our focus is on the construction of generic networks with bounded or minimal diameter using completely distributed mechanisms. On the other side, we are also interested in existing approaches that aim at improving existing search methods by modifying the topology of the underlying overlay network without relying on semantic information about shared content or search queries.

Low diameter networks are very easy to obtain: for instance, power law graphs (also known as *scale free graphs*) have typically low diameters and are very frequent in nature; the Internet itself is a scale free network [9]. An example of peer-to-peer network that explicitly constructs a power law topology in order to keep a low diameter is PHENIX [10]. The protocol focuses on building networks that are highly resilient to malicious attacks, with the identity of major hubs kept hidden to prevent targeted attacks.

Similarly to power law graphs, super-peer networks also rely on a small set of nodes that have a number of connections larger than average, but with the degrees following a bimodal distribution. As a drawback, all these approaches could suffer catastrophic consequences if hubs fail, thus avoiding such topologies may be preferable.

Provided that a global knowledge of the network is available, low diameter non-power law graphs can be easily constructed through random augmentation [11] (i.e. by adding random additional links between nodes). An example of distributed overlay construction that approximates a random graph is introduced by the SUPS project [12], with a protocol that guarantees a low-diameter topology and balanced link distribution. The motivation behind SUPS comes from the observation that research on optimal topologies connecting super-peers is scarce. The genericity of the proposed approach suggests a possible implementation of BLÅTANT-R in similar super-peer scenarios. A similar idea is proposed in [13], with the use of a structured overlay to connect super-peers.

Sharing many features with random graphs, small-world graphs also exhibit short diameters. Some examples of construction of overlay networks with small-world properties are described in [14], while another decentralized algorithm is detailed in [15]. A completely different approach is proposed by NEWSCAST [16], where an overlay with small-world characteristics emerges from random exchanges of neighborhood information between peers using a gossip-based protocol.

Finally, to improve the efficiency of flooding search protocols, the CLUSTELLA [17] system proposes a neighbor selection strategy to avoid redundant links and reduce the number of messages generated by each query.

III. The Algorithm

This section describes the BLÅTANT-R algorithm, which improves our previous algorithm [18] by introducing balanced link distribution, fault resilience, and recovery mechanisms.

A. Optimization Rules

The algorithm optimizes the network according to two simple rules depending on a user defined integer parameter $D > 0$, such that the resulting diameter d is $D \leq d \leq 2D - 1$. The following rules are evaluated to determine whether a new connection is required, or if an existing one is redundant. Both rules refer to a partial view of the network maintained by each node.

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

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

Where $d'_G(x, y)$ is defined as $min(d_G(x, y), d_G(y, x))$.

Whereas the connection rule effectively reduces the diameter of the network, the disconnection rule aims at removing redundant links by breaking small cycles.

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

$$\exists\ n_k \in N_i, k \neq j\ :\ d^*_{G'}(n_j, n_k) + 1 \leq D \tag{2}$$

Where $d^*_G(x, y)$ is defined as $max(d_G(x, y), d_G(y, x))$.

In [19] it was proved that these rules are sufficient

for bounding the diameter according to the value of D while removing unnecessary logical links.

In a centralized scenario it was also proved that the disconnection rule cannot lead to a partitioning of the network. Nonetheless, in order to ensure that it is safe to concurrently use these rules even in a distributed scenario, a restriction on the disconnection rule must be applied. In particular, for each considered cycle, only one node (in the current implementation the one with the greatest identifier according to some ordering known to all nodes in the cycle), called *master*, is allowed to perform a disconnection. We now prove that under this assumption, even a completely decentralized application of the disconnection rule cannot result in a partitioned graph.

Lemma 1. Safeness of the Disconnection Rule. Concurrent application of the disconnection rule, cannot lead to a partitioning of the network.

Proof: Suppose first that a global knowledge of the network is available. As only the corresponding master can break its cycle by applying the disconnection rule, it is straightforward to see that the graph remains connected. Nonetheless in our completely distributed scenario, information collected by the master about the status of connections may be outdated, e.g. refer to cycles that have already been broken. In such cases, it is necessary that a master does not perform additional disconnections that may effectively partition the network. To prevent this problem each master could maintain a history of detected and broken cycles, but this would require a large amount of memory on each master node, especially in very dynamic networks. A better solution is to only allow a master to remove connections with its own neighbors, thus making it possible to rely only on local and up-to-date information to verify whether the cycle has already been broken. ■

As the optimization algorithm itself cannot break connectivity, fault resilience problems are confined to node or communication breakdowns. The following subsections provide a detailed description of the algorithm; the pseudo-code has been omitted in this paper because of space constraints, but is available online at [20].

B. Network Peers

The proposed approach relies on a *pure* peer-to-peer system: each peer is considered to have equal capabilities (computing power, memory, bandwidth), and there is no concept of *super-peer*. This choice is motivated by the fact that in actual networks it is possible to assume that even the slowest peer has sufficient resources to manage and process the traffic generated by the algorithm. Each peer is in charge of maintaining the network connected, determining if new logical links are necessary according to the connection rule, and removing redundant links according to the disconnection rule.

1) 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 slots are reserved for incoming recovery connections. Except during the connection phase, a node n_i can only communicate with peers in its neighborhood N_i. 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 $n_i \not\leftarrow n_j$. As a node can only communicate with its neighbors, $n_i \leftarrow n_j$ implies $n_i \in N_j$. To avoid the creation of large hubs, the size of the neighbor set is limited.

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 an 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.

2) Peer Logic: At regular intervals c, each node evaluates the information stored in the α table by first constructing a graph, and then by computing the paths to other nodes in this partial view of the network. A connection procedure to most distant node is started if the connection rule applies. In the same way, disconnection procedures are started when cycles satisfying the disconnection rule are found. To avoid unnecessary computations, the frequency of evaluations is directly proportional to the amount of traffic, because the availability of updated information depends on the number of incoming ants.

C. Ant Species

The tasks of collecting information about the status of the network, as well as connecting and disconnecting peers, are performed by different species of ants.

a) Discovery Ants: Discovery ants are in charge of wandering across the network and collecting information about visited nodes. Each ant carries a fixed-size buffer V of length l_V with identifiers of each visited node, along with the identifiers of their neighbors. As discovery ants may get lost because of node crashes, they have a limited lifespan ι (maximum number of wandering steps); conversely, both when a new node connects to the network, and with frequency $1/\iota$, a new individual is generated with probability μ, ensuring the survival of the population.

b) Construction-Link Ants: Construction-Link ants are sent both by nodes wanting to join the network, and during

recovery procedures. A node can either accept the connection, or forward it to one of its neighbors (randomly chosen). Forwarding is required if the node has already reached the maximum number of allowed neighbors. Nodes that have a number of neighbors lower than m will accept the connection with a normal-distributed probability $\mathcal{N}(m/2, 1)$, thus favoring a balanced distribution of links. 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.

c) Optimization-Link Ants: 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. 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.

d) Unlink Ants: Unlink ants remove the links as result of the disconnection rule. 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.

e) Ping Ants: Ping ants are used to keep up to date information between peers when no other traffic does (low traffic situations). An ant traveling from node n_i to n_j, updates the entry corresponding to n_i in α_j.

D. Pheromone trails

On each node n_i we distinguish two types of pheromone trails: outgoing trails γ_i, and incoming trails β_i. These trails are kept alive by ants traveling on the network: $\gamma_i[n_j]$ is reinforced by ants traveling from n_i to n_j, while $\beta_j[n_i]$ is reinforced by ants coming from n_i to n_j. At each migration step, discovery ants preferably choose neighbors with low γ concentration, thus ensuring a uniform network coverage as well as a balanced load across all connections. The concentration of β pheromone is used to detect unresponsive or dead neighbors: when a trail completely evaporates, the corresponding peer is deemed dead, and a recovery operation is started. In order to keep connection alive even in low traffic situations, $ping$ ants are periodically sent between neighbors, increasing the respective pheromone concentration, and updating the respective neighbors information.

Each pheromone trail τ is reinforced according to the formula:

$$\tau \leftarrow \delta_{fill}$$

At regular intervals ω, the evaporation process updates the concentration as follows:

$$\tau \leftarrow \tau * \psi$$

A pheromone trail is considered as completely evaporated when its concentration falls below a certain minimal value. For γ and β, mininimal values are γ_{min}, respectively β_{min}.

E. 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 1a) and 1b) depict an example topology before, respectively after the departure of node n_i.

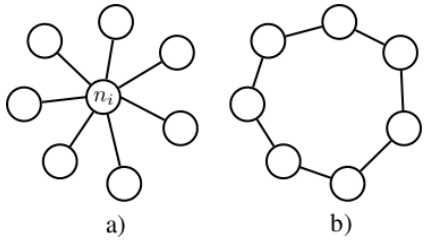

Fig. 1. Leaving procedure

F. 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 γ pheromone trail, it may start a recovery procedure. The exact behavior of the node depends on whether $n_j \not\leftarrow n_i$ or $n_j \leftarrow n_i$.

1) 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.

2) 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 n_i as well as to all known neighbors of n_i. Figure 2a) shows an example situation, and Figure 2b) the result of a recovery procedure started after the crash of node n_i.

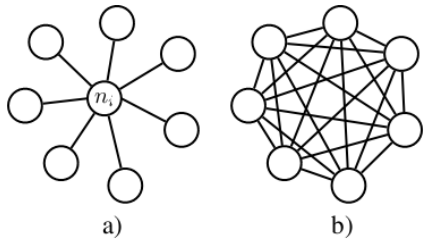

Fig. 2. Recovery after node crash

IV. Evaluation

This section presents and discusses results for both the convergence of the algorithm to an optimal topology, and the total communication cost. In particular, we evaluated the behavior of the algorithm on different scenarios of deployment ranging from static and stable networks, to dynamic and unreliable ones. For each scenario 20 different runs were repeated. Table I shows the actual algorithm parameters used during all simulation runs. The time is expressed in *iterations*: an iteration corresponds to a migration of the whole ant population (except delayed or loss ants in the unreliable network scenarios). Each simulation run consists of 10000 iterations.

Optimization parameter	D	5
α size limit	$\alpha_{maxsize}$	28
$max(\|N_i\|)\ \forall i$	m	8
Allowed optimization links	mo	6
Discovery Ant lifespan	ι	25
Discovery Ant birth probability	μ	0.05
Discovery Ant vector length	l_V	15
Pheromone update	δ_{fill}	1
Pheromone decay	ψ	0.9
Minimum γ pheromone concentration	γ_{min}	0.25
Minimum β pheromone concentration	βmin	0.005
Pheromone evaporation interval	ω	1
Evaluation interval	c	1
Construction-Link Ant lifespan	$clant_{ttl}$	20

TABLE I
Simulation Parameters

A. Simulation Scenarios

In all scenarios, communication between nodes is assumed to be asynchronous: when an ant migrates between two peers, the sender receives no feedback or acknowledgement from the receiving peer. Whereas in reliable scenarios an acknowledgement is implicit (because reception is guaranteed), in unreliable ones no assumption can be made about the reception of the information sent. All scenarios share the same initial topology of a typical local area network consisting of 1281 nodes[1]. It should be noted that this topology is not exactly reproduced during simulations, because some peers would forward incoming connection requests when reaching the limit of $m = 8$ neighbors (the maximum allowed in this simulation).

1) Static topology (S): This scenario simulates just the construction of the network, without further changes.

2) Dynamic topology with proper node disconnection (DP): In order to evaluate the resiliency of the algorithm we choose to simulate the addition and removal of nodes as a homogeneous Poisson process $\{N(t) : t \geq 0\}$ with intensity $\lambda = 0.05$ described by Equation 3. The process of adding and removing nodes is started after iteration 100, to avoid influence on network bootstrap, and stopped at iteration 5000, to determine if the network becomes stable afterwards.

[1]http://diuf.unifr.ch/pai/blatant/lan1281

$$P[N(t+\tau) - N(t) = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!} \tag{3}$$

This results in an average frequency of one node addition and one removal every 20 iterations, and an average total number of 245 nodes added/removed during simulations. New peers try to connect to a random existing node by sending a Construction-Link ant. When a node is removed, it executes the leaving procedure. Ants running on a node that disconnects are killed.

3) Dynamic topology with improper node disconnection (DI): As in the previous scenario, we simulate the addition and departure of nodes as an homogeneous Poisson process. Instead of performing a proper disconnection, departing peers just leave the network and stop communicating, resulting in the execution of recovery procedures. It should be noted that this scenario is the most pessimistic one, because it supposes that none of the peers leaves the network properly.

4) Reliable network (R): In this scenario no information can be lost during transmission, and each ant takes exactly an iteration to migrate from one node to another.

5) Unreliable network (U): To evaluate the fault tolerance of the algorithm, this scenario simulates packet delivery delays and packet loss as follows:

- at each migration, each ant has a probability of 10% to be delayed until the next iteration;
- at each migration, each ant has a probability of 1% to be lost.

Simulation scenarios include a combination of both the static and the dynamic topology on both a reliable and an unreliable network. For remaining rest of this paper, simulation scenarios will be referred as:

1) S-R, for static topology on a reliable network;
2) DP-R, for dynamic topology with proper disconnection on a reliable network;
3) DI-R, for dynamic topology with improper disconnection on a reliable network;
4) S-U, for static topology on an unreliable network;
5) DP-U, for dynamic topology with proper disconnection on an unreliable network;
6) DI-U, for dynamic topology with improper disconnection on an unreliable network.

V. Results

This section presents simulation results for the algorithm in the previously described scenarios. In Figures 3-7 vertical lines mark the 100^{th} and 5000^{th} iteration, corresponding to the beginning, respectively the end, of addition/removal of nodes in dynamic scenarios. More detailed statistical results for selected measurements are available in Table II for the diameter, Table III for the average path length, and Table IV for the size of the largest connected component. Information regarding iteration k are noted as θ_k and σ_k, for the average value, respectively the standard deviation. The maximum standard deviation across all iterations is noted as σ_{max}, whereas the mean standard deviation is noted as σ_{mean}.

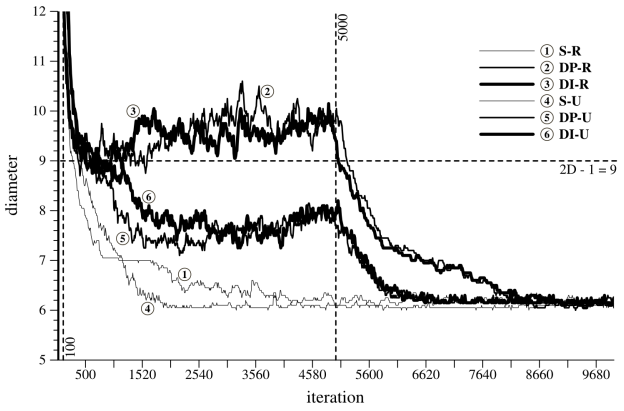

Fig. 3. Diameter

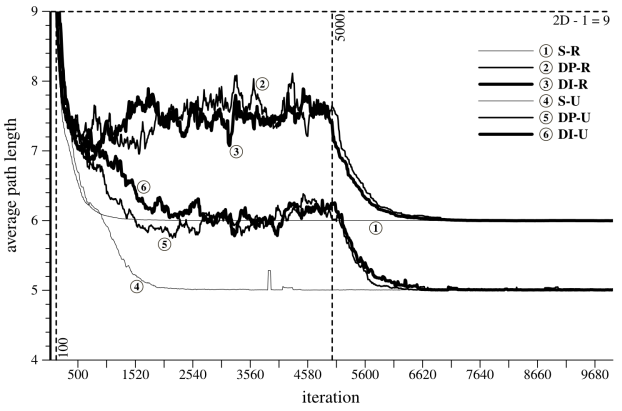

Fig. 4. Average Path Length

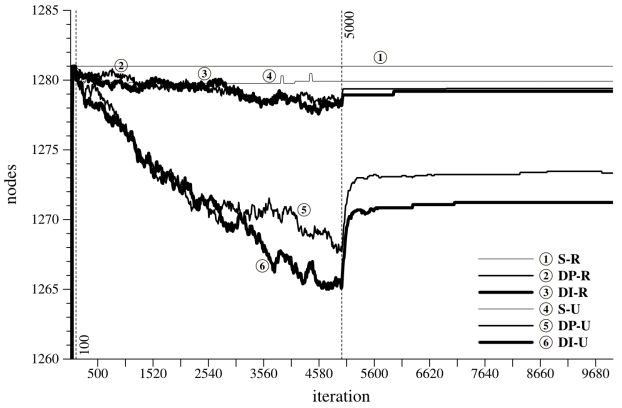

Fig. 5. Size of the Largest Connected Component

Finally, $|G_k|$ represents the average size of the network at iteration k. The small average standard deviation of obtained results confirms the stability of the algorithm.

A. Diameter and Average Path Length

The goal of the algorithm is to create and maintain a network with bounded diameter. Figure 3 shows the evolution of the diameter length during the simulation. In each considered scenario, the algorithm is able to bound the original diameter pretty fast during the construction phase. In S-R and all unreliable scenarios, the diameter always remains under the $2D - 1 = 9$ limit. Although surprising, the fact that DP-U and DI-U perform better than DP-R and DI-R can be explained by a side-effect of the additional links created by the algorithm in unreliable scenarios.

In all but the S-R and DP-R scenarios, the diameter slowly grows as new nodes start to connect and existing ones disconnect. The reason behind this behavior is the very limited number of neighbors allowed for each peer, the high frequency of addition/removal of nodes, and the relatively high packet delivery delay and packet loss probabilities. A detailed analysis of the simulation results shows that recovery procedures are executed even in the S-U scenario: this is caused by missed pings that result in wrongful detections of crashed neighbors.

Figure 4 shows the average path length. In all scenarios this value conforms to results shown in Figure 3, with the average path length consistently lower than the diameter and well withing the bounds determined by $D = 5$. In dynamic scenarios, as soon as the addition and removal of nodes is finished (after iteration 5000), the average path length converges close to optimum values around 5 and 6.

B. Connectivity and Edges count

In order to optimize the network, the algorithm constantly modifies the topology. During this process, it is important to ensure that the network does not become partitioned. To analyze this aspect, we monitored the size of the largest connected component as shown in Figure 5. In Section III it was proved that if the topology is static (i.e. no new node joins the network, and no existing node disconnects), the optimization process is safe. This property has been validated by simulation runs, where the size of the largest component in the S-R scenario always corresponds to the total number of nodes. Also shown in the same figure is the impact of unreliable communication between nodes. In all reliable scenarios, as well as in the static unreliable one, the size of the largest connected component remains very close to the total number of nodes. In unreliable dynamic scenarios the network becomes slightly partitioned, but partially recovers after iteration 5000.

Finally, Figure 6 shows that the number of edges is close to the limit in all scenarios ($\approx 1281 * m = 10248$, $\approx 1281 * mo = 7686$). We also see that in all unreliable scenarios, the algorithm makes use of the additional two free slots ($m - mo$) available on each node dedicated to recovery procedures.

C. Communication cost

The total communication cost is proportional to the size of ant populations. To take into account the fact that each species carries a different amount of information, and to abstract from the different ways in which this information can be represented, we introduce the notion of *token*. Each token stores the unique identifier of a peer and a numeric

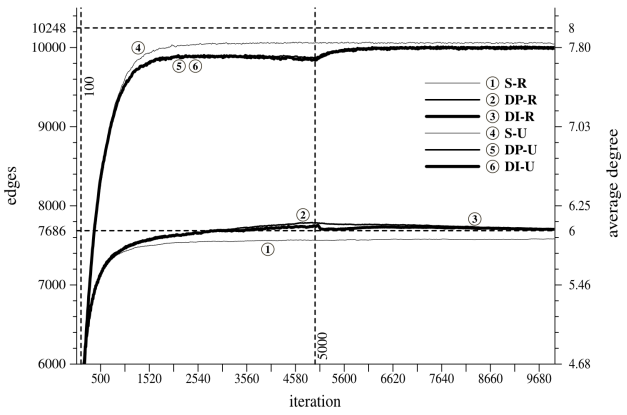

Fig. 6. Edges count and average degree

S-R	θ_{100}	=	11.85	σ_{10^4}	=	0.31
	θ_{5000}	=	6.1	σ_{max}	=	3.06
	θ_{10^4}	=	6.1	σ_{mean}	=	0.41
DP-R	θ_{100}	=	12.15	σ_{10^4}	=	0.22
	θ_{5000}	=	9.9	σ_{max}	=	4.92
	θ_{10^4}	=	6.05	σ_{mean}	=	0.84
DI-R	θ_{100}	=	12.25	σ_{10^4}	=	0.44
	θ_{5000}	=	9.4	σ_{max}	=	3.89
	θ_{10^4}	=	6.25	σ_{mean}	=	0.83
S-U	θ_{100}	=	13.9	σ_{10^4}	=	0.31
	θ_{5000}	=	6.1	σ_{max}	=	3.48
	θ_{10^4}	=	6.1	σ_{mean}	=	0.38
DP-U	θ_{100}	=	14.25	σ_{10^4}	=	0.31
	θ_{5000}	=	7.7	σ_{max}	=	4.2
	θ_{10^4}	=	6.1	σ_{mean}	=	0.65
DI-U	θ_{100}	=	13.3	σ_{10^4}	=	0.41
	θ_{5000}	=	7.95	σ_{max}	=	3.13
	θ_{10^4}	=	6.2	σ_{mean}	=	0.75

TABLE II
DIAMETER θ

S-R	θ_{100}	=	9.31	σ_{10^4}	=	0
	θ_{5000}	=	6	σ_{max}	=	2.19
	θ_{10^4}	=	6	σ_{mean}	=	0.03
DP-R	θ_{100}	=	9.31	σ_{10^4}	=	0
	θ_{5000}	=	7.57	σ_{max}	=	2.47
	θ_{10^4}	=	6	σ_{mean}	=	0.47
DI-R	θ_{100}	=	9.54	σ_{10^4}	=	0
	θ_{5000}	=	7.41	σ_{max}	=	2.42
	θ_{10^4}	=	6	σ_{mean}	=	0.45
S-U	θ_{100}	=	10.40	σ_{10^4}	=	0.01
	θ_{5000}	=	5	σ_{max}	=	2.41
	θ_{10^4}	=	5	σ_{mean}	=	0.09
DP-U	θ_{100}	=	10.51	σ_{10^4}	=	0.01
	θ_{5000}	=	6.04	σ_{max}	=	2.73
	θ_{10^4}	=	5.01	σ_{mean}	=	0.29
DI-U	θ_{100}	=	9.95	σ_{10^4}	=	0.01
	θ_{5000}	=	6.25	σ_{max}	=	2.56
	θ_{10^4}	=	5.01	σ_{mean}	=	0.4

TABLE III
AVERAGE PATH LENGTH θ

timestamp; if we make the token correspond to an IP address (4 bytes) and an integer timestamp (2 bytes), it is possible to get an estimation of the actual traffic generated by the algorithm. Based on the parameters used during simulations,

S-R	θ_{100}	=	1281	σ_{10^4}	=	0
	θ_{5000}	=	1281	σ_{max}	=	169.78
	θ_{10^4}	=	1281	σ_{mean}	=	0.22
	$\lvert G_{10^4}\rvert$	=	1281			
DP-R	θ_{100}	=	1280	σ_{10^4}	=	4.52
	θ_{5000}	=	1278.3	σ_{max}	=	160.43
	θ_{10^4}	=	1279.15	σ_{mean}	=	4.1
	$\lvert G_{10^4}\rvert$	=	1279.15			
DI-R	θ_{100}	=	1280	σ_{10^4}	=	4.31
	θ_{5000}	=	1278.19	σ_{max}	=	161.72
	θ_{10^4}	=	1279.2	σ_{mean}	=	3.94
	$\lvert G_{10^4}\rvert$	=	1277.7			
S-U	θ_{100}	=	1281	σ_{10^4}	=	1.89
	θ_{5000}	=	1280.3	σ_{max}	=	109.21
	θ_{10^4}	=	1280.3	σ_{mean}	=	2.08
	$\lvert G_{10^4}\rvert$	=	1281			
DP-U	θ_{100}	=	1280	σ_{10^4}	=	5.04
	θ_{5000}	=	1268.65	σ_{max}	=	150.03
	θ_{10^4}	=	1274.05	σ_{mean}	=	4.47
	$\lvert G_{10^4}\rvert$	=	1277.45			
DI-U	θ_{100}	=	1280	σ_{10^4}	=	4.25
	θ_{5000}	=	1266	σ_{max}	=	162.1
	θ_{10^4}	=	1272.2	σ_{mean}	=	4.02
	$\lvert G_{10^4}\rvert$	=	1276.75			

TABLE IV
SIZE OF LARGEST CONNECTED COMPONENT θ

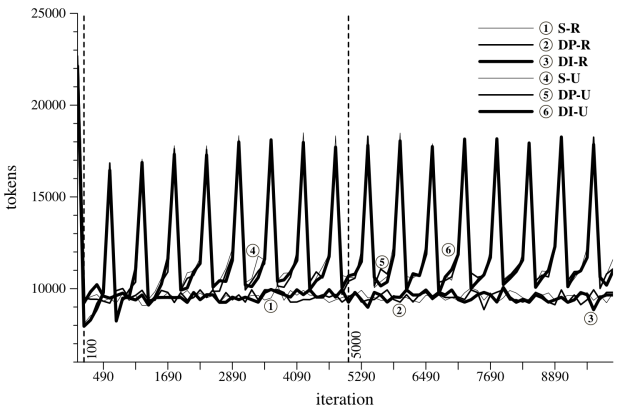

Fig. 7. Network Traffic

we can evaluate the size of each species of ant as:

- Discovery Ant: at most 15 entries in the vector, with at most 8 tokens per entry, totaling 135 tokens;
- Optimization-Link Ant and Construction-Link Ant: 2 tokens (source and target nodes);
- Unlink Ant: 2 tokens (source and target nodes);
- Ping Ant: 1 token for the source.

According to the value of $\mu = 0.05$, it is possible to estimate the average size of the Discovery Ant population at around 64 ($=\approx 1281 * 0.05$) individuals during all simulations. At the beginning, the size of the population is doubled because connections of nodes to the network (which could generate new ants) corresponds to the beginning of intervals ι. The size of other colonies heavily depends on the behavior of the algorithm. As discovery ants make up for the largest part of the traffic, there are no significant differences between dynamic and static scenarios.

Figure 7 shows the evolution of traffic during the simulation. As expected dynamic networks require more bandwidth due to connection requests sent by incoming nodes, as well as by recovery procedures executed when nodes left. Traffic in all reliable scenarios is very similar, with DP-R having slightly lower bandwidth requirements than DI-R, because the leaving procedure generates less traffic than the recovery procedure. It is also interesting to note that unreliable networks produce significantly more traffic, mostly because of retransmission of lost data, and display peaks at regular intervals. These peaks are caused by delays in packet delivery which result in longer relative lifespans of discovery ants. Although not visible in the figure, an important part of the bandwidth is consumed by ping ants, with an average of 1000 individuals in static scenarios, and 1300 in dynamic scenarios. In real situations these ants may not be necessary as that information could be piggybacked on application traffic (e.g. resource discovery and monitoring queries).

D. Complexity

Although a complete analysis of the complexity of BLÅTANT-R is not yet available, it is possible to provide a simple evaluation based on the complexity of the basic algorithms involved, which gives an upper-bound for the overall complexity. Each peer has to execute shortest-path algorithms on a partial graph of the network based on the information found in the α table. Although the complexity of these procedures can be as high as $O(n^2)$, the size of n is in fact very small and limited by both the size of α, and the maximum number of neighbors for each peer. For the results presented in this paper the maximum size of the partial graph is $28 * 8 = 228$ nodes. To reduce the load on each peer n_i many computations can be avoided when evaluating new connections, by just ignoring nodes $n_j \in N_i$, and neighbors-of-neighbors (i.e. $n_k \mid \exists\, n_j \in N_i \wedge n_k \in N_j$). On the other side, while evaluating disconnections, only the distances between pairs of active neighbors need to be computed. Simulations runs have confirmed that the overall time taken by evaluation procedures is negligible.

VI. CONCLUSIONS

This paper presented the BLÅTANT-R algorithm for constructing and maintaining a bounded diameter network overlay. The topology is optimized according to a user defined parameter by adding necessary logical links between nodes and removing redundant connections. The algorithm uses ant inspired methods to monitor the network and collect information used by the optimization process; this results in an adaptive behavior suitable for dynamic networks. Simulations showed that BLÅTANT-R is able to construct and maintain a topology with bounded diameters in both static and dynamic scenarios. Results also proved that the algorithm is fault tolerant in respect to underlying network problems such as delivery delays and packet loss. Recovery mechanisms ensure that the network remains connected even in the event of node crashing. Nonetheless, simulations also underlined the fact that proper disconnections procedures and reliable communication result in significantly lower communication costs.

Future work will concentrate on the evaluation of efficient search methods exploiting the topology maintained by BLÅTANT-R. In particular, this algorithm, along with other swarm inspired methods, will be used to support resource discovery in a novel grid middleware [21].

REFERENCES

[1] Gnutella 0.6 protocol draft. `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`.

[2] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[3] Jordan Ritter. Why gnutella can't scale. no, really. 2001. `http://www.darkridge.com/~jpr5/doc/gnutella.html`.

[4] Sylvia Ratnasamy, Paul Francis, Scott Shenker, and Mark Handley. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[5] Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM 2001*, San Diego, CA, September 2001.

[6] B. Beverly Yang and H. Garcia-Molina. Designing a super-peer network. *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60, March 2003.

[7] The kazaa website. `http://www.kazaa.com`.

[8] J. Mischke and B. Stiller. A methodology for the design of distributed search in p2p middleware. *Network, IEEE*, 18(1):30–37, Jan/Feb 2004.

[9] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[10] Rita H. Wouhaybi and Andrew T. Campbell. Building resilient low-diameter peer-to-peer topologies. *Comput. Netw.*, 52(5):1019–1039, 2008.

[11] Paul Erdős and Alfréd Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[12] Young June Pyun and Douglas S. Reeves. Constructing a balanced, (log(n)/loglog(n))-diameter super-peer topology for scalable p2p systems. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pages 210–218, Washington, DC, USA, 2004. IEEE Computer Society.

[13] Miguel Castro, Manuel Costa, and Antony Rowstron. Peer-to-peer overlays: structured, unstructured, or both. Technical Report MSR-TR-2004-73, Microsoft Research, Cambridge, 2004.

[14] Jon Kleinberg. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians (ICM)*, 2006.

[15] 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.

[16] Márk Jelasity and Maarten van Steen. Large-scale newscast computing on the internet, October 2002.

[17] Stefan Schmid and Roger Wattenhofer. Structuring unstructured peer-to-peer networks. In *Proceedings of HiPC 2007*, pages 432–442. Springer, 2007.

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

[19] Amos Brocco, Fulvio Frapolli, and Béat Hirsbrunner. Shrinking the network: The blåtant algorithm. Technical Report 08-04, Department of Informatics, University of Fribourg, Fribourg, Switzerland, April 2008, http://diuf.unifr.ch/pai.

[20] Amos Brocco. Blåtant-r pseudo code. January 2009. `http://diuf.unifr.ch/pai/blatant`.

[21] 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*. GCC2008, October 2008.