Fault-Tolerant Routing for P2P Systems with Unstructured Topology

Leonardo Mariani
Dipartimento di Informatica, Sistemistica e Comunicazione,
Universita degli Studi di Milano Bicocca,
Via Bicocca degli Arcimboldi, 8,
I-20126 - Milano, Italy,
mariani@disco.unimib.it

Abstract

New application scenarios, such as Internet-scale computations, nomadic networks and mobile systems, require decentralized, scalable and open infrastructures. The peer-to-peer (P2P) paradigm has been recently proposed to address the construction of completely decentralized systems for the above mentioned environments, but P2P systems frequently lack of dependability. In this paper, we propose an algorithm for increasing fault-tolerance by dynamically adding redundant links to P2P systems with unstructured topology. The algorithm requires only local interactions, is executed asynchronously by each peer and guarantees that the disappearance of any single peer does not affect the overall performance and routing capabilities of the system.

1 Introduction

Internet-like environments, nomadic networks and mobile systems require the development of decentralized, scalable and open systems that can run even on top of unreliable layers. Recently, the P2P technology has been used to build this kind of systems, e.g., Gnutella [6] provides search functionality over large networks, P-Grid [1] supports cheap searches at the cost of complex joins, and Freenet [5] addresses the construction of P2P systems over mistrusted peers.

Trustability of services implemented by the P2P technology is heavily dependent upon the dependability properties of the underlying system. A lot of work must still be done to develop fully dependable P2P systems, especially in term of scalability, fault-tolerance, anonymity and security [8].

In this paper, we present a new algorithm which increases fault-tolerance by dynamically adding redundant links to P2P systems with unstructured topologies, i.e., P2P systems where peers can join in a spontaneous way by binding to the contacted peer.

P2P systems with unstructured topologies have been used in several contexts, such as the implementation of file-sharing systems in Internet-like environments [6, 17], the implementation of computing platforms for next generation economic transactions [3] and the implementation of distributed file systems [5]. In the above mentioned cases, fault-tolerance is only partially achieved by caching many redundant links so that if a peer fails, it is still possible to route the request through a different path. However, this way of addressing fault-tolerance does not provide any real guarantee. In fact, a single faulty peer can cause either the disconnection of a subsystem or the degradation of the system's performance. The limited improvements that can be achieved by only increasing connectivity without a criterion has been discussed in [16].

We address the problem of providing fault tolerance by allowing each peer to execute an algorithm whose overall effect is to guarantee that no single failing peer can cause neither disconnections nor long routing [1] on the P2P system. Achievement of this property is possible by allowing each peer p to distribute links among neighbors so that if p disappears efficient routing is still possible.

Since the actual topology of a P2P system can be viewed as a graph, the algorithm can be thought to be performing local transformations on that graph in a way that ensures that you can remove any single node and still leave the graph connected. In this context, removal of a node can be due to either a failure or to an unexpected leave of the peer. The algorithm even guarantees that you can remove as many non-neighboring peers as you want while leaving the system connected. Moreover, by choosing among different values for the algorithm's parameters, we can achieve different degrees of both tolerance to faults and routing ef-

[1] the longest routing that can be caused by disappearance of a peer p is equal to p's number of neighbors

ficiency.

The algorithm is very effective in all cases in which the graph resembles the structure of the small-world network model [20], i.e., there are many nodes with local connections and few nodes with long wide-ranging connections. In fact, searches on small-world networks have been demonstrated to be strongly affected from the presence of these long-distance connections that can be used as shortcuts during routing [20]. P2P systems with spontaneous topology tend to organize themselves as in the small-network model [6, 5]; therefore they can strongly take advantage of our algorithm that propagates both local and long-distance connections through peers.

In Section 2 we briefly present the current methods of addressing fault-tolerance in P2P systems; in Section 3 we provide basic definitions that are used during the presentation of our algorithm; in Section 4 we outline the basic ideas underlying the proposed algorithm; in Section 5 we present the algorithm; in Section 6 we discuss related work; and finally, in Section 7 we conclude.

2 Fault-Tolerance in P2P Systems

P2P systems are networks where nodes act as both clients and servers. In particular, each peer provides the same set of functionalities, and the system-level behavior is obtained by the composition of the single behaviors. Therefore, the effectiveness of the whole system is strongly dependent upon the results provided by each single peer. In fact, a single leaving or failing peer can compromise the result of a long interaction through many peers. The common solution consists of providing multiple alternative routing paths for the same communication.

In the case of P2P systems with managed joining, leaving and routing operations, strategic redundant connections can be dynamically added. Chord stores additional links to assure that at least low-performance routing is possible [18]; CAN supports several mechanisms for replicating data and enabling searches across multiple directions [13]; and Oceanstore relies on Tapestry [21] which is a routing infrastructure storing additional links to guarantee fault-tolerance [14].

In the case of P2P systems with unstructured topologies, fault-tolerant routing has been achieved by storing links to dynamically discovered peers [6, 5]. In this way, a peer can choose to send a request through one or more known peers. However, there are no real guarantees on reliability of the final system, even in the case of a single faulty peer. In fact, P2P systems with unstructured topologies arrange their structures as in the Small-World Network Model [20]; therefore, if one peer possessing many wide-range connections fails, the performance of the whole system can be very negatively affected. Samant and Bhattacharyya empirically evaluated the impact on fault-tolerance for random addition of new links [16]. They discovered that increasing the average number of links per node from 5 to 8 provides little benefit to fault-tolerance, but is valuable for increasing tolerance to attacks. Therefore, the early experience from Samant and Bhattacharyya highlights that increasing connectivity without a rationale is not an effective solution to fault-tolerance problems.

Despite this weakness, P2P systems with unstructured topologies have been the *de-facto* organizational model in comparison to those with managed topology, especially in open environments such as the Internet [6, 17]. Making these systems more tolerant to failures, that is the achievement of our work, would allow them to be utilized in construction of a wider range of applications.

Another dimension of fault-tolerance in P2P systems is losing of data that is stored on the failing peers. There are several techniques for replicating data among peers [7, 4, 14, 5]; however we do not discuss this aspect since we focus on robustness of connectivity rather than on data availability.

3 Basic Definitions

In this section, we provide basic definitions that clarify the concepts that will be used in the following sections. In particular, we adapt the definition of Strongly Connected Component to the case of subgraphs of a given diameter to accommodate the local nature of our algorithm.

The structure of a P2P system at a given instant of time can be modeled as a directed graph $G = (P, E)$, where P is a finite set of nodes (or peers) and E is a relation over P (the set of edges). A finite sequence of peers $\langle p_1, \dots p_k \rangle$ is a finite path from u to v if $p_1 = u$, $p_k = v$ and $\forall i : 1 \leq i \leq k-1, (p_i, p_i + 1) \in E$. If exists a path from $u \in P$ to $v \in P$, we say that u reaches v ($u \rightsquigarrow v$). Two nodes u, v are mutually reachable $u \leftrightsquigarrow v$, if $u \rightsquigarrow v$ and $v \rightsquigarrow u$. The distance function $d : P \times P \rightarrow \mathbb{N} \cup \{\infty\}$ associates to a pair (u, v) the minimum length of the path from u to v, if any, otherwise ∞. The set of strongly connected components of G (denoted with SCC_G), is the partition over V induced by the equivalent relation $\leftrightsquigarrow$.

Given a peer $p \in G$, we denote with G_p^a the subgraph of G containing all peers $p_i \in G$, s.t., $0 < d(p, p_i) \leq a$. Thus, G_p^a denotes the subgraph of G that can be accessed from p by following at most paths of length a. In a similar way, given a peer $p \in G$, we denote with $\overline{G}_p^a$ the subgraph of G containing all peers $p_i \in G$, s.t., $0 < d(p_i, p) \leq a$. Thus, $\overline{G}_p^a$ denotes the subgraph of G that can reach p by paths of length at most a.

We denote $SCC^a_G(p)$ as the partition over $G^a_p \cup \overline{G}^a_p$ induced by $\leftrightsquigarrow$. In other words, an element of $SCC^a_G(p)$ is a subgraph

- that does not contain p
- that contains nodes whose distance from p is at most a
- where any node is reachable from any other node
- that is not proper subset of any other set in $G^a_p \cup \overline{G}^a_p$ that satisfies the above properties

A peer p that runs our algorithm must request neighboring information to its direct neighbors (peers in G^1_p). (The reason of this additional interaction is explained in Section 5.) All peers in G^1_p receive messages from peers connected by in-going links. Receivers both respond to the request and store the identity of the requestor; hence, peers create bidirectional links from in-going connections. Therefore we can assume, without loss of generality, that a P2P system with unstructured topology that runs our algorithm uses bidirectional connections.

The existence of bidirectional links implies that $G^a_p = \overline{G}^a_p$. Partitions in $\overline{G}^a_p \cup G^a_p$ and partitions in G^a_p are the same, therefore we can present our algorithm considering only the set G^a_p and ignoring the set $\overline{G}^a_p$. However, in contexts where this simplification is not valid, e.g., when P2P systems hide the identity of the sender, the algorithm can be applied preserving the distinction between the two sets. In this paper, we present the examples referring to the case of bidirectional links.

4 Run-Time Addition of New Links

Our algorithm is based on the idea that

> "each peer can perform actions to assure that its disappearance does not affect the connectivity of the system"

In particular, a peer can communicate with its neighbors to create the additional communication links that are necessary to make its disappearance negligible with respect to the system-level routing capabilities.

The main achievement of our fault-tolerant algorithm, that is "no disappearance of single peers can cause disconnections in the graph", can be stated as enforcing $|SCC^a_G(p)| = 1$ for all $p \in G$ and for a given value of a. In fact, if the neighbors of a peer p form a unique strongly connected component, the navigability is assured even if p disappears.

The algorithm is locally executed by each peer. The result of one execution by a peer p corresponds to achieving $|SCC^a_G(p)| = 1$. Executing the algorithm all peers leads to $|SCC^a_G(p)| = 1$ for all $p \in G$.

Executions are not synchronous, therefore there is not a concrete instant of time where the property holds for all peers of the system. However, we assume that the algorithm is executed frequently enough to efficiently adapt the fault-tolerant capability of the P2P system to the actual topology. In future work, we plan to empirically evaluate the relation between the frequencies of join and leave operations and the frequency of executions of the algorithm to estimate the scalability of the approach in dynamic situations.

A P2P system that addresses fault-tolerance by running our algorithm obtains the following benefits:

- any peer can autonomously decide when executing the algorithm, for instance regularly in time, if is not busy, or when new links are acquired
- the algorithm requires only local interactions; the value of a defines the max diameter of the considered subgraphs, hence it precisely defines the extension of the interactions
- if the algorithm is executed frequently enough, the P2P system self-adapts the support to fault-tolerance to changes in the topology. The frequency of execution can be tuned to different values in different regions of the system

Our algorithm cannot be included in any unstructured P2P system for free. In fact, the algorithm requires additional interactions, i.e., requesting for neighbors' neighboring tables and sending to neighbors the links that must be created to obtain a strongly connected component. Both cases consist of additional interactions that can slow down the system. However, the extra cost can be reduced for P2P systems broadcasting requests [17, 6]. In this case, we can both include the requests for neighboring tables in the messages that are broadcasted when searching for an item and include the neighbor tables in the response messages.

5 The Algorithm

The algorithm is composed of four main phases (p is the peer that executes the algorithm):

1. the peer p acquires neighboring tables from peers distant at most a by using the system routing protocol, e.g., by broadcasting query messages with time-to-live equal to a
2. the peer p computes $SCC^a(p)$[2]
3. if $|SCC^a(p)| > 1$, p computes the links that must be created to have $|SCC^a(p)| = 1$
4. links are pushed to peers

```
// parameters
connectionStrategy = ...; peerSelectionStrategy = ...;

if p.NeighborTable.size = 0 &&
            p.pointedByTable.size == 0 exit();

// acquiring neighbor tables up to a distance a
Tables=acquireNeighborsTables(a, p.NeighborTable,
p.pointedByTable);

// computes the strongly connected component
SCCs=ComputeSCC(Tables)

//compute link that must be sent
links = genLinkForSCCs(SCCs, connectionStrategy,
peerSelectionStrategy);

//send links
sendInfo(links);
```

Figure 1. A sketch of the algorithm for merging a set of disjoined SCCs to a unique SCC

Figure 1 sketches the implementation of the algorithm. The function $acquireNeighborsTables$ requests neighboring tables by using the P2P-specific routing protocol. The parameter a is used to limit the scope of the request. The part of the graph that is discovered includes peers connected by both incoming and outgoing links, which correspond to `NeighborTable` and `pointedByTable` respectively. In the case of bidirectional links, it is enough to consider the `NeighborTable`. The function `ComputeSCC` is used to compute the strongly connected components over the acquired portion of the graph. For this step, it is possible to use any of the well known algorithms performing the computation in a linear amount of time [19, 11]. The function `genLinkForSCCs` is used to generate the set of links that must be added. The behavior of such function is dependent from two parameters: `connectionStrategy` and `peerSelectionStrategy`. The parameter `connectionStrategy` defines the pattern of links that must be created to obtain the unique strongly connected component. The `peerSelectionStrategy` parameter defines the strategy for selecting the peer in the strongly connected component that is used for adding the links. Finally, the function `sendInfo` simply sends information, by using the system routing protocol, to peers selected for gaining the additional links.

We have defined three connection strategies:

sequential the disjoined strongly connected components are sequentially connected

onetoall one strongly connected component is connected with all other strongly connected components

alltoall every strongly connected component is connected to all other strongly connected components

The *sequential* strategy binds the number of additional links (if n is the number of identified strongly connected components, up to $n-1$ links can be generated), but in the worst case the delivery of a message can require traversing all strongly connected components; the *onetoall* strategy is used to achieve fast routing with a limited amount of additional links (if n is the number of identified strongly connected components, up to $n-1$ links can be generated), however the peer that owns all new connections can be a bottleneck for the system; finally, the *alltoall* strategy is used to achieve high routing capabilities at the cost of a high number of created edges and many generated messages (if n is the number of identified strongly connected components, up to $(n-1)!$ links can be generated).

When a link among two strongly connected components must be created, we have to select (at least) two peers that will be connected by the new link from the corresponding strongly connected components. For this purpose, we have identified three peer selection strategies:

mostconnectedpeer the peer with most connections is selected

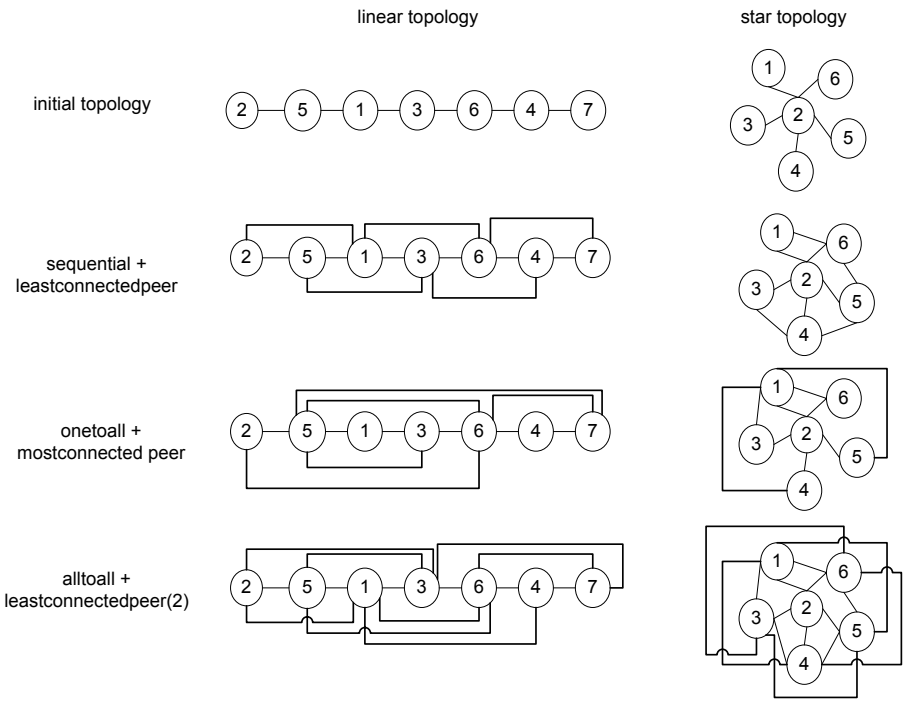
leastconnectedpeer the peer with the smallest set of connections is selected

leastconnectedpeers(n) the n peers (if any) with the smallest set of connections are selected

The *mostconnectedpeer* strategy is used to concentrate all links on few peers that will work as gates for the SCCs they belong to. This approach can guarantee a fast way to move through the SCCs of the system, but create also a set of most important peers. The *leastconnectedpeer* strategy is used to homogeneously distribute links over peers, but routing can be slow since a message can even pass all members of a strongly connected component before exiting. The *leastconnectedpeers(n)* strategy is used to select the n peers with the smallest amount of connections to increase redundancy of links. For optimal values of n, the last strategy can be a good tradeoff between the number of additional links and the connectivity of the resulting strongly connected component.

The parameter a can be tuned to define the scope of the algorithm. Considering distant peers in the computation of the strongly connected components requires intensive communications for discovering the part of the subgraph that is distant a from the peer running the algorithm, requires additional computation to identify the strongly connected components, and creates large strongly connected components that can require long routing to be traversed. Considering only close peers requires less communication, less computation, and creates smaller strongly connected components

[2] in the description of the algorithm we left the G symbol implicit



Legend
The first row contains the initial topology, while the last three rows contain the final configuration obtained with the parameters `sequential+leastconnectedpeer`, `onetoall+mostconnectedpeer`, and `alltoall+leastconnectedpeer(2)` respectively. Edges are bidirectional and node numbering represents the order of execution of the algorithm among nodes.

Figure 2. The output of the algorithm for the linear topology and the star topologies.

than including also distant peers. Moreover, small values of a tend to provide a structure that is less stable, but more cheap to run and more adaptable to the contingent situations; large values of a are more suitable for stable environments. For this reason, we now both consider examples and discuss the algorithm for $a = 1$.

Examples To provide evidences of the effectiveness of our algorithm, we discuss the final topology that is obtained by applying our algorithm to two simple and one complex target topologies. We initially consider the linear and star topologies where it is possible to easily trace the behavior of the algorithm for the different values of the parameters. Then, we apply the algorithm to a complex topology that represents a possible configuration for a P2P system. We remark that for all examples edges between nodes represent bidirectional links and the numbering of the nodes correspond to the order that is assumed for running the algorithm (in this way the reader can check the result). For example, the first node running the algorithm is node 1, then node 2, and so on until the last node executes the algorithm. If one pass is not enough to reach the final configuration, the algorithm is assumed to be executed again by node 1, then node 2, and so on until the final configuration is obtained.

Let us start with the linear and star topologies shown in Figure 2. We computed the three possible outputs that can be obtained by three different choices for the parameters. We do not represent all possible combinations of parameters because some combinations produce similar results (and space limits prevent the representation of all cases).

If we look at the resulting topologies, we recognize that, as expected, no single failing peer can cause the disconnection of the graph. Moreover, we notice three clear different styles to obtain that. In the first case, `sequential+leastconnectedpeer`, several short-distance links are added to provide a way to move around any single peer. This style adds a limited amount of links, but seldom produces long-distance links thus overall navigability is not effectively improved. This effect is particularly evident in the star topology where disappearance of node 2 provokes routing among all its neighbors. In the second case, `onetoall+mostconnectedpeer`, the system contains several new paths also with long-distance

links. The overall navigability is increased, but some nodes own many links, thus they can be a bottleneck for the system if many requests are received. For instance, look at nodes number 5 and 6 for the linear topology, and node number 1 for the star topology. Finally, the third case, `alltoall+leastconnectedpeer(2)`, clearly increases overall navigability without introducing any bottleneck in the system, but many links are added among all nodes. This way to propagate links can be expensive in term of both communications and memory consumption.

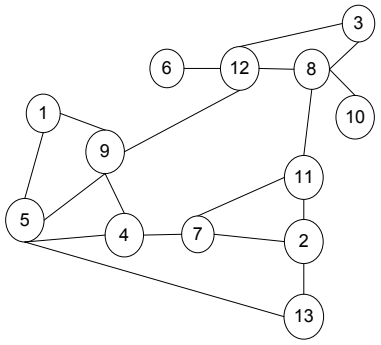


Figure 3. A complex topology

We can conclude that the style `sequential+leastconnectedpeer` can be used when failures are unlike to take place at the same time on two connected peers, and we are not interested in highly increasing search performances of the system. The style `onetoall+mostconnectedpeer` can be used when we are interested in both fault tolerance and search performances, but we estimate that there are not too much concurrent requests taking place at the same time, otherwise most connected peers will be flooded of requests. Finally, the style `alltoall+leastconnectedpeer(n)` can be used when both fault-tolerance and search performances are important and we estimate that both intensive communications among peers and additional memory consumption are rewarded by the increasing in search performance.

To demonstrate that the algorithm both propagates long-distance links and is effective in complex topologies, we provide two additional examples, both of them referring to the topology shown in Figure 3. The first example computes the new topology by using the parameters `sequential+leastconnectedpeer` and the second example computes the new topology by using the parameters `onetoall+mostconnectedpeer`. Results are presented in Figure 4 and Figure 5, respectively.

In the case of `sequential+leastconnectedpeer` parameters, we can notice that several paths turning around peers have been added and some long-distance links have been propagated. For instance, two short-connections

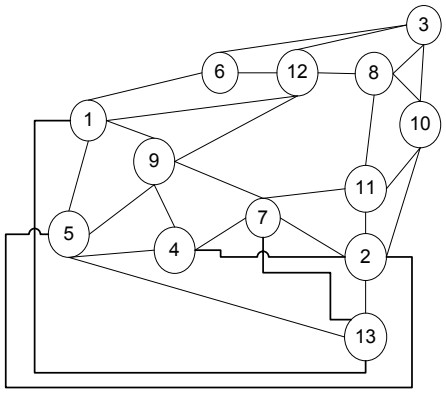


Figure 4. The final configuration for the P2P system in Figure 3 when the algorithm is executed with the parameters `sequential` and `leastconnectedpeer`

between 1 and both 6 and 12 have been added. These connections are used to provide fault-tolerance for failures in 9. The sequential approach can introduce turn-around routing. For example, if 9 disappears and 4 must communicate with 12, the available path is $5, 1, 12$. Moreover, the sequential approach can propagate some of the long-distance links, in fact new links between 1 and 13, and between 5 and 2 are created.

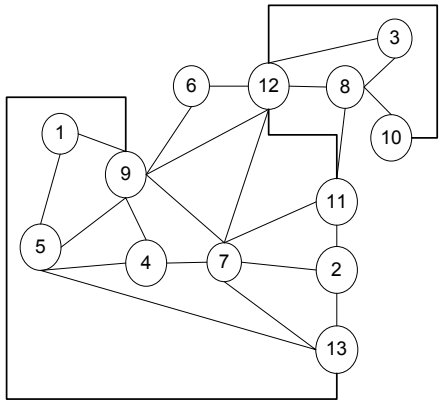


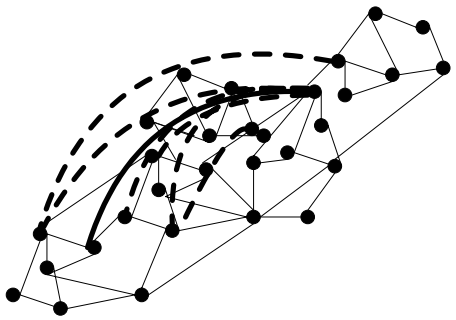
Figure 5. The final configuration for the P2P system in Figure 3 when the algorithm is executed with the parameters `onetoall` and `mostconnectedpeer`

In the case of the `onetoall+mostconnectedpeer` parameters, we can notice that links are concentrated in few strategic nodes that manage the communication across long

distances. In fact, node 9 owns 7 connections (2 connections more than the previous case), node 12 owns 8 connections (3 connections more than the previous case), and node 7 owns 6 connections (1 connection more than the previous case). Even if nodes are concentrated, the algorithm guarantees that we can remove any of these nodes and the graph is still connected. In this case, long distance links are concentrated on few peers; in fact peers 9, 7 and 12 manage direct connections to the whole systems.

The different styles underlying the two approaches are also graphically evident. The topology in Figure 4 is obtained from topology in Figure 3 by mainly adding peripheral links and few internal links, while the topology in Figure 5 is obtained from the topology in Figure 3 by mainly adding internal links between most important nodes and few peripheral links.

If we move these results to larger systems resembling the small-world network models, we obtain that peers managing long-distance connections push to their neighbors these connections increasing search performances. Figure 6 highlights this situation.



Legend
The bold line represents one long-distance connection. The dotted bold lines are some of the new long-distance connections that would be established for connecting distant unconnected components with any combination of parameters.

Figure 6. An example of the long-distance links that are added by our algorithm considering only one already existing long-distance link.

6 Related Work

The importance of evaluating the robustness of organizational networks such as P2P networks with unstructured topology has been recently shown by Dodds, Watts and Sabel in [9]. Their evaluation criterion matches the basic ideas of our algorithm; in fact they evaluate connectivity robustness by taking into consideration the size of the greater strongly connected component.

In the context of VLSI design, several algorithms for identifying min-cut balanced bipartitions have been proposed, i.e., partitioning a netlist of a circuit into two disjoint components with equal weight such that number of connections between components is minimized [15, 10, 12]. In principle, these algorithms can identify subnets connected by few connections also for P2P systems. Therefore, it would be possible to enforce new connections to increase fault-tolerance. However fully decentralized versions of these algorithms for P2P networks are not available.

In the context of unstructured P2P systems, two main approaches have been followed: caching and replica management. Caching consists on peers dynamically acquiring new links from both forwarded and received messages. Caching increases the number of existing connections, but does not provide real guarantee for fault-tolerance. Caching has been used in several systems, such as Farsite [2] and Freenet [5]. Our algorithm does not replace caching mechanisms, but can be integrated with them to provide P2P systems able to both increase connectivity of individual peers and provide fault-tolerant connectivity.

Replica management consists of peers explicitly sending information about either stored data or existing links, to other peers. Replica management for stored data has been implemented by several P2P systems, such as Farsite [2] and P-Grid [1]. On the other hand, our approach can be considered the first attempt to implement replica management of links for unstructured P2P systems. Replica management for both data and links can be combined to obtain a system where peers proactively address fault-tolerance with respect their own failures.

7 Conclusions and Future Work

We propose an algorithm increasing fault-tolerance on P2P systems with unstructured topologies. The algorithm guarantees that no single disconnection can cause either disconnection of a subgraph or an increase of routing steps greater than the number of neighbors of the disappeared peer. The algorithm is executed asynchronously from each peer and requires only local interactions. The connection strategy can be configured in different ways for both linking the strongly connected components and selecting the specific peers to connect to. In the case of failures, some configurations of the parameters can provide even stronger guarantees on the routing capabilities than routing along all neighbors.

Results obtained by simulating the algorithm in different modalities provide confidence of its effectiveness. In future work, we plan to experimentally investigate the behavior of

the algorithm when it is concurrently executed on different peers, and to evaluate the relation between frequency of execution of the algorithm and number of joining and leaving peers. Moreover, quantitative inspection of the degree of fault-tolerance that is provided by our algorithm is necessary to confirm the suitability of the approach for large-scale P2P systems.

We foresee the possibility to develop a variant of the Gnutella protocol [6] that uses our algorithm both during joins operations and during a peer's life time to push fault-tolerance into the system. In this way, it is possible to build a system with a high degree of freedom in interactions, with a high degree of fault-tolerance and with a customizable topology, i.e., forcing different topology styles by choosing different values of the two parameters of the algorithm.

References

[1] K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt. Improving data access in P2P systems. *IEEE Internet Computing*, 6(1):58–67, January/February 2002.

[2] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[3] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *The Journal of Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, November/December 2002. Special Issue on Grid Computing Environments.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 1–14, 2002.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval systems. In *Proceedings of the ICSI Workshop on Design Issuer in Anonymity and Unobservability*, number 2009 in Lecture Notes in Computer Science. Springer-Verlag, Berkley, California, 2001.

[6] Clip2. The gnutella protocol specification v0.4. Specification, 2001.

[7] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of ACM Special Interest Group on Data Communication (SIGCOMM) 2002*, August 2002.

[8] F. DePaoli and L. Mariani. Dependability in peer-to-peer systems. *IEEE Internet Computing*, 8(4):54–61, July/August 2004.

[9] P. S. Dodds, D. J. Watts, and C. F. Sabel. Information exchange and the robustness of organizational networks. *Proceedings of the National Academy of Sciences of the United States of America*, 21(100):12516–12521, October 2003.

[10] C. Fiduccia and R. Mattheyses. A linear time heurisic for improving network partition. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[11] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Baltimore, Maryland, 2003. Society for Industrial and Applied Mathematics.

[12] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM) 2001*, San Diego, California, 2001.

[14] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, September/October 2001.

[15] B. Riess, K. Doll, and M. Frank. Partitioning very large circuits using analytical placement techniques. In *Proceedings of the third ACM/IEEE Design Auotmation Conference*, pages 646–651, 1994.

[16] K. Samant and S. Bhattacharyya. Topology, search, and fault tolerance in unstructured P2P networks. In *Proceedings of the 27th Hawaii Conference on System Sciences*, pages 289–294. IEEE, January 2004.

[17] Sharman Networks. KaZaA. http://www.kazaa.com, Visited in 2004.

[18] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrisham. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[19] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[20] D. Watts and S. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 390:440–442, 1998.

[21] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide area location and routing. technical report UCB/CSD-01-1141, University of California at Berkeley, April 2001.