Fault-Tolerant P2P Networks: How Dependable is Greedy Routing?

Sabina Serbu
University of Neuchâtel
CH-2009 Switzerland
sabina.serbu@unine.ch

Peter Kropf
University of Neuchâtel
CH-2009 Switzerland
peter.kropf@unine.ch

Pascal Felber
University of Neuchâtel
CH-2009 Switzerland
pascal.felber@unine.ch

Abstract

Under churn, the problem of preserving accessibility is addressed by maintaining valid entries in the routing tables towards live nodes. However, if the system fails to replace the entries of dead nodes with entries of live nodes soon enough, requests may fail. In such cases, mechanisms to route around failures are required to increase the tolerance to node failures.

Existing DHTs include extensions to provide fault tolerance when looking up keys, however, these are often insufficient. We analyze the case of greedy routing, which is a routing algorithm that is preferred for its simplicity, however with limited dependability even when extensions are applied.

The main idea is that fault tolerance aspects need to be dealt with already from the design of the overlay. We propose a simple overlay that offers support for alternative paths, and we create a routing strategy which takes advantage of all these paths to route the requests, while keeping maintenance cost low.

1. The Problem

Dependability concerns many properties of a system, such as scalability, security, data integrity, availability, routing or fault tolerance. These properties are generally dealt with according to the system's architecture. In this research we focus on the accessibility of data in structured P2P systems, and more precisely in Chord-like systems.

Data accessibility is provided by both a *fault-tolerant infrastructure* and *fault-tolerant routing*. An overlay infrastructure offers the capacity of recovering from failures by replacing entries of dead nodes with entries of live nodes in the routing tables. To update an entry in the routing table, one must find a node that would fit at that entry. Since it is costly and mostly impossible to keep all routing tables entries always populated with alive nodes, these updates are made periodically: at each time interval, maintenance requests are issued and the routing tables are updated. Subsequently, this still leaves a time window when entries may refer to dead nodes. Because routing table entries become often invalid under churn, the system has to provide fault-tolerant routing by finding alternative routes to forward the requests towards the destination. We say that the overlay routing is dependable if a request reaches its destination.

Existing DHTs essentially focus on preserving stored data and preserving consistency of the network structure, ignoring fault-tolerant access to the data under churn. In Section 2 we present some of the existing solutions for fault-tolerance.

To discuss fault-tolerant routing, we illustrate in Section 3 the case of Chord-like DHTs which use greedy routing, one of the most-known and widely-used routing algorithm. Greedy routing is simple: at each routing step, the request is directed towards a node as close as possible to the destination. This strategy provides fast lookup because the number of hops is minimized. In case of node failure, greedy routing algorithms typically apply a "route around" strategy by using a lower entry from the routing table if the normally chosen entry contains a dead node. Experimental results have confirmed that greedy routing under node failures is by far not a reliable strategy with respect to fault tolerance and routing dependability. Indeed, the advantage of getting as close as possible to the destination at each routing step (i.e., going as far as possible from the source) becomes a disadvantage under node failures, as this strategy exploits only a small part of the possible paths. At each routing step, the number of possible paths towards destination is heavily decreasing, which drastically diminishes the chances of finding a valid path to destination.

Following the analysis of standard greedy routing, we present a simple solution for fault-tolerant routing and then we conclude in Section 5.

2. Providing Fault-Tolerance

In this section we present some of the existing solutions for fault-tolerance regarding the infrastructure and the routing strategies. Note that we do not deal with any security aspects, such as trusted nodes or trusted information (this is well detailed in [5]).

2.1. Fault-Tolerant Infrastructure

The easiest and most widely addopted solution to deal with dead nodes in routing tables is the addition of *backup nodes* (redundant links). The best-known examples are systems like Chord [14] or Pastry [12]. In Chord, each node maintains a list of a fixed number of successors on the ring. When an entry has failed, a lower entry is used. For the lowest entries, the list of successors may be used. Similarly, Pastry maintains a special list of nodes in a leaf set of typical size of 16 or 32. If an entry that should be used has failed, the message is routed to the node from the leaf set which is the closest to the destination. Analyzing the performance of these two methods, Liu *et al.* [10] present a study of the Chord and Koorde [7] systems, running experiments with 4 and 8 successors as backup nodes. Their results show that Chord has better dynamic resilience than Koorde, however they do not focus on the tradeoff between the number of successors and the percentage of request failure.

Hildrum *et al.* [6] also show that by increasing the number of backups for every routing table entry, the chance of reaching the destination improves significantly. Their experiments are done on Pastry and Tapestry [15] systems, with a maximum of 5 backup nodes. The same idea is addopted by Lam *et al.* in [9], where they propose the K-consistent networks. Each node keeps always K nodes at each entry in its routing table. Whenever a node from the routing table fails to respond, a mechanism is started to find a new suitable node for the same entry. In [8], Korzun *et al.* propose to extend the local information (routing tables) with global information. Each node maintains a cache with a set of cycles in the form of a routing path starting at the current node. When a request arrives, and if the node contains a cycle that passes through a node which is close to the destination key, then that particular cycle is followed.

This type of solutions is obviously limited by the number of the backup nodes (or cycles) that are used. The larger the number, the higher the number of alternative paths, and so the higher the probability of success. However, the disadvantage is seen in the additional costs imposed by maintaining more node entries in the routing tables.

As a complementary solution, Castro *et al.* [4] propose techniques to detect node failures and repair routes in MSPastry, an implementation of Pastry. This decreases the number of dead entries in the routing tables, however there is no solution to completely eliminate them.

2.2. Fault-Tolerant Routing

For dependable routing under failures, Aspnes *et al.* [2, 3] propose two extensions to greedy routing. When a node cannot find another node that is closer to the destination than itself, it can use either *random re-route* (random choice of another node to forward the request to), or *backtracking* (sending back the request to the previous node in the request path by keeping track of some visited nodes). These two extensions provide reasonable results, however, they still exploit only a small number of possible paths.

Backtracking is a good technique to enlarge the number of alternative paths, but it is not well exploited when used with greedy routing. A request that gets close to the destination, but is forced to use backtracking, would do small back hops, which means that the gained number of alternative paths remains small.

Another possibility to increase the request success rate is *parallel routing*. In this case, more copies of the same request are sent towards the same destination through different paths. For example, the request is sent through all routing table entries smaller than the suitable entry. In [5], for fault-tolerance, several copies of the same request are sent from the source to a set of its neighbors. This technique is called *neighbor set anycast*, and it causes the messages to follow different paths.

Of course, independently of the choice for the next hops of the paths, when doing parallel routing, more requests are sent in the system, so more processing is required at the nodes. This increases the costs considerably.

In contrast, we aim to improve fault tolerance by allowing at each routing step to consider the maximum number of possible alternative paths, even if no failure has been detected yet. We start by an analysis of greedy routing to show its weaknesses and then we present a simple solution for fault-tolerant routing.

3. On The Dependability of Greedy Routing

Many DHTs use greedy routing to forward the requests because of its simplicity. This strategy gives good results in terms of path length, but it has limitations in the number of paths it can exploit. To get from source to destination, greedy routing adjusts the bits from left to right when the request is forwarded to the next hop in the request path. This strategy generally leads to *path convergence*: the last hops of most requests for a certain destination pass through only a small set of nodes, which are mostly the preceding neighbors. Under a failure-free operation, these nodes are likely to be overloaded if the destination is very popular.

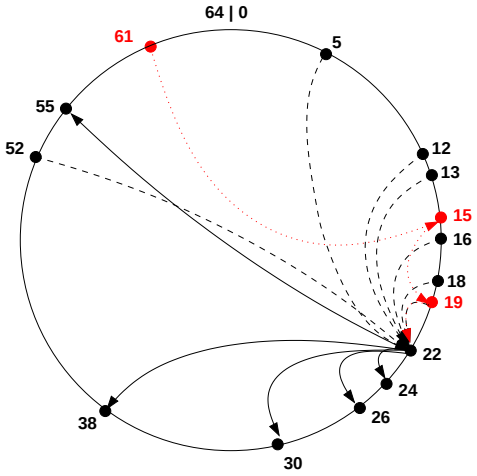

Figure 1. Example of Chord with an identifier space of $2^m = 64$.

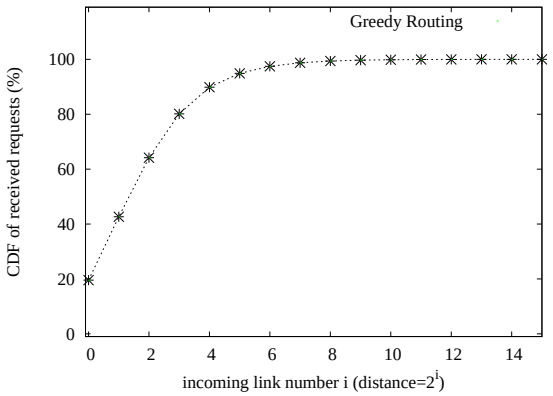

Figure 2. Percentage of received requests per incoming link (2^i) in a Chord system using greedy routing. The quick increase of the Cumulative Distribution Function (CDF) of received requests (i.e, the primary usage of the incoming links from small distances) shows that redundant paths exist, but are not used.

Furthermore, if one of them fails, the traffic will be severely affected.

Systems such as Chord [14] or Pastry [12] suffer from these limitations of greedy routing. A graphical representation of a Chord example is shown in Figure 1. In Chord, each node and object has a m-bit identifier on a 2^m ring, obtained by respectively hashing the IP address and the name. The objects are mapped to their following node on the ring. For routing purposes, each node has a routing table with m entries, each entry i pointing towards the first node on the ring at a distance of minimum 2^i, where $i = 0..m-1$. Conversely, each node is in the routing table of other nodes, so it has incoming links from these nodes. In the example of the figure, 15 (out of 40) nodes are shown on a 2^6 ring. The incoming links of node 22 are shown with dashed dark lines, and its outgoing links are shown with solid dark lines. While each of the outgoing links points to nodes at a distance of close to a power of 2 away, the distance from the incoming link nodes is more variable. Each request is forwarded by greedy routing, following always a clockwise path, as for example the request going from node 61 to node 22 (the dashed grey line).

Figure 2 shows, in percentages, the number of requests that are received per incoming link in a Chord-like system with no node failures. In this experiment, the identifiers are mapped on $m = 15$ bits. The system has 10,000 nodes and 20,000 keys, with 200,000 requests uniformly issued. As can be seen on the left-hand side of the graph, most of the traffic is received from the incoming links with small distances. More than 80% of the traffic is received from the incoming links at 2^i away, where $i \leq 4$. The incoming links of further away nodes ($i > 4$) are merely used. Since the nodes have been uniformly distributed on the identifier space, the distance between two successive nodes is 2^2 on average, which means that the incoming links from the smallest distances (up to a maximum of 2) come from the same node. Thus, the predecessors of a node are critical nodes because they bear the majority of the traffic for that node. If such a node fails, the rate of request success drastically decreases.

In the experiment under failures, whenever a routing table entry that needs to be used contains a failed node, a lower entry is used instead. Figure 3 shows the percentage of failed requests when varying the percentage of failed nodes. The graph shows that this strategy is not dependable: when half of the nodes fail, also half of the requests fail.

The main reason for this poor performance is that alternative paths are not exploited. At each node, the request is sent as close as possible to the destination. This means that the distance between a next hop node n_{nh} and the destination n_d of a request is minimal, and also the number of possible alternative paths is minimal once the request has reached n_{nh}. As an example, a request in Figure 1 goes from node $n_s = 61$ to node $n_d = 22$, by going through nodes 15 and then 19, according to greedy routing. The initial number of incoming links of n_d that can be used is 6: incoming links 5, 12, 13, 16, 18 and 19. The possible alternative paths pass through each of these incoming links. At n_s, greedy routing chooses $n_{nh} = 15$. When this node is reached, the number of incoming links of n_d that can be used decreases to 3, since nodes 5, 12 and 13 are already behind the current position of the request. If nodes 16, 18 and 19 fail, the request will also fail to reach its destination,

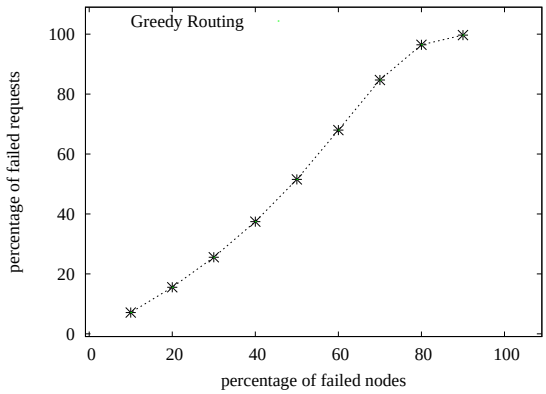

Figure 3. Failure rate of greedy routing. The high percent of failures shows that greedy routing does not exploit the redundant paths.

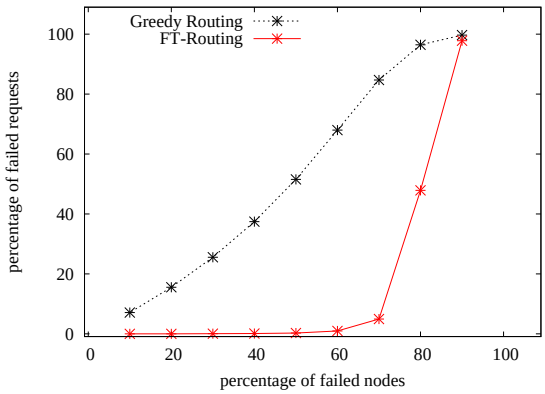

Figure 4. Comparison between the percentage of failures of Greedy Routing and our solution, where up to 60%, the node failures are hardly noticed.

even though alternative paths from n_s through the incoming links of n_d, nodes 5, 12, or 13, would be valid.

Another aspect of this kind of overlays is that the outgoing links of a node are not exactly at 2^i away, so the request does not necessarily follow 2^i jumps. This fact prevents from applying a deterministic routing strategy to exploit other valid paths.

All these observations uncover the mismatch between the goal of providing fault tolerance and the means used for lookup with greedy routing. The extensions for fault-tolerance may give good results, however, if better support for exploiting alternative paths is already considered at overlay design time, even better results may be obtained as our approach presented in the next section demonstrates.

4. Towards More Dependable Routing

To provide a high level of fault tolerance, we take into account both, the overlay and the routing strategy to offer such support for alternative paths.

Research studies have shown that the hypercube is a structure that provides the possibility to rely on a high number of alternative paths [11, 13, 1], thus providing a good fault tolerance, however at a high maintenance cost. To benefit from the advantages but still trying to keep costs at a low level, we use an innovative way of creating the overlay by getting as close as possible to a hypercube structure. This allows the usage of power of 2 links to easily forward requests towards destination.

The routing strategy must provide a large number of alternative paths even when the request is close to the destination (*close* means a small number of digits that are different in the source and destination identification sequences). As a consequence, in our solution the request follows small steps in the beginning of the routing path (where the low-order digits are treated) and then longer steps (treating higher-order digits). It is clear that when no failures occur, the strategy gives similar results to greedy routing, because the number of hops to route a request is of the order $O(logN)$, assuming that we can treat the bits in any order. However, when failures do occur, this strategy exploits a much higher number of possible paths, as the requests are routed around failures with a higher probability.

In short, we are applying simple modifications to Chord-like systems. Chord cannot easily exploit redundant paths because of its non-determinism in node placement that does not permit treating digits in any order, so we are fixing this by adding some **determinism in the placement of the nodes**. This means that we may control the position of the nodes on the ring, which is obviously advantageous for the routing strategy. In contrast to the common method of using a hash function to map the nodes on the ring, we approximate a hypercube structure by trying to maintain an even inter-node distance equal to a power of 2 despite churn. Then, we are also modifying the **routing protocol to exploit alternative paths** by taking into account all possible incoming links of the destination.

We have compared the percentages of failures presented in Figure 3 for greedy routing with the percentages obtained with our approach (FT-routing). The two graphs are shown in Figure 4. The increase in node failures starts reflecting in request failures at an only high percentage of 70% failed nodes. This low percentage of failed requests for even high rates of failures demonstrates the high dependability of this strategy.

5. Conclusions

The analysis on the fault-tolerant infrastructures and routing strategies shows that the support for fault-tolerance cannot be an afterthought when willing to provide a high fault tolerance at low costs. Greedy routing is simple, however for fault-tolerance, it lacks of dependability.

Consequently, we designed both the infrastructure and the routing strategy of our overlay with the goal to offer the support for fault tolerance. The rate of request success is much higher, and the maintenance cost remains low, since no additional structures are required.

References

[1] J. I. Alvarez-Hamelin, A. C. Viana, and M. D. Amorim. DHT-based functionalities using hypercubes. In *Proceedings of 8th IFIP/IEEE International Conference on Mobile and Wireless Communications Networks*, 2006.

[2] J. Aspnes, Z. Diamadi, and G. Shah. Fault-tolerant routing in peer-to-peer systems. In *Proc. 21st ACM Symp. on Principles of Distributed Computing*, 2002.

[3] J. Aspnes, Z. Diamadi, and G. Shah. Greedy routing in peer-to-peer systems. *extended version of Fault-tolerant routing in peer-to-peer systems*, 2006.

[4] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proc. 2004 International Conference on Dependable Systems and Networks*, page 9, 2004.

[5] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. 5th Symposium on Operating Systems Design and Implementation*, pages 299–314, 2002.

[6] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Proc. 17th International Symposium on Distributed Computing*, pages 321–336, 2003.

[7] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, pages 323–336, 2003.

[8] D. Korzun and A. Gurtov. Cyclic routing in peer-to-peer systems. *Technical Report*, 2007.

[9] S. S. Lam and H. Liu. Failure recovery for structured p2p networks: Protocol design and performance under churn. In *SIGMETRICS - Performace 2004*, 2004.

[10] Z. Liu, G. Chen, C. Yuan, S. Lu, and C.-Z. Xu. Fault resilience of structured p2p systems. In *WISE*, pages 736–741, 2004.

[11] T. Locher, S. Schmid, and R. Watternhofer. eQuus: A provably robust and locality-aware peer-to-peer system. In *Proceedings of the 6th International Conference on Peer-to-Peer Computing*, 2006.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[13] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup – hypercubes, ontologies and efficient search on p2p networks. In *Workshop on Agents and P2P Computing*, 2002.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, 2001.

[15] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.