Recursive or iterative routing? Hybrid!

Gerald Kunzmann
gerald.kunzmann@tum.de
Institute of Communication Networks at the Technische Universität München (TUM), Germany

Abstract: In our current work with the Chord protocol [SMK$^+$01] we had to decide whether to use iterative or recursive routing. Iterative routing provides the initiating node with a lot of information and influence on the routing path. Recursive routing on the other hand is faster and results in less overhead, most notably in stable networks. We present a hybrid routing solution that inherits the advantages of both approaches without increasing the overhead or slowing down the lookup procedure.

1 Introduction

Peer-to-Peer (P2P) networks and their applications gain increasing importance in today's Internet, as already today the majority of IP traffic is caused by P2P applications. Since the upcoming of Napster a lot of research has been done in this area. Still, growing demands like less data rate consumption, faster and more reliable search responses and the development of new applications engage many researchers worldwide. A fundamental problem that confronts peer-to-peer applications is to efficiently locate the node that stores a particular data item. Currently a lot of research concentrates on structured P2P networks, where neighbors are not chosen by a random process, but form a deterministic network structure [ESZK04]. Each peer in the network has a specific identifier (node ID) that is e. g. created by hashing the nodes IP address and port number. The peers are then arranged in the ID space according to their identifiers. The Chord protocol for example arranges its nodes in a 1-dimensional ring-shaped ID space. Content identifiers (content ID) are generated with the same hash function and content is stored at nodes that's node ID is close to the content ID. To provide efficient and scalable routing, each Chord node stores a list of neighbours and fingers, i. e. "‘shortcuts"’ through the network.

As the placement of nodes and content is predetermined, lookups benefit from knowing the location where the queried content should be located in the network. So the query messages can be forwarded on an almost direct path without the need of flooding the message through the network. Arriving at its deterministic destination, a lookup can be resolved at any rate: If the content or a link to it is available, it is returned to the initiator of the query, otherwise, a "content not available in the network" message is sent back.

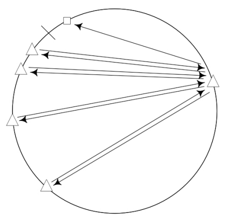

Figure 1: **Iterative routing**: Nodes communicate only with the originator. The tick mark denotes the position of the queried key in the ID space. The square shows the key's successor node. The triangles and arrows show a lookup path. The last node before the tick mark is the key's predecessor.

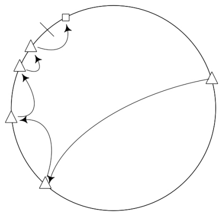

Figure 2: **Recursive routing**: Each node forwards the query to the next node. During the lookup process no information is send back to the originator, resulting in less packet overhead.

2 Iterative routing

The basic Chord [SMK+01] and Kademlia [MM02] algorithms use iterative routing (see figure 1). When looking up a certain key, the originator sends a request to its finger that is the closest predecessor to the key ID. This node returns its closest finger preceding the key. After receiving this information, the originator sends the request to this new node that is already closer to the desired key, waits for the answer and so on. This procedure is continued until the node that precedes the key ID is reached. The predecessor finally returns the key's successor that in turn is responsible for the key. This procedure is called iterative routing as the requesting node handles the complete routing traffic. The initiator iteratively asks each node on the logical path to the requested key for the according routing information for the next hop. The initiator also monitors the whole routing process by using a timer for each hop to determine if the queried node is absent or the packet has been lost and a new packet has to be sent.

Iterative routing has two main advantages: First, the originator can easily keep track of the lookup route and can react to problems, e. g. nodes that send wrong routing information, very fast. Second, when the lookup fails due to an absent node, the originator is soon aware of the failure and probably can continue the lookup somewhere next to the absent node, skipping all previous hops.

3 Recursive routing

If recursive routing (see figure 2) is employed each node forthright forwards the query to the next node in the routing path, without sending information back to the initiator, until the request reaches the key's predecessor. This node finally returns the key's successor to the originator. Theoretically this might halve the delay of each hop, because nodes do not have to wait for any acknowledgments. Simulations prove that in an error-free environment recursive lookups require only 60% of the lookup time compared to iterative routing [DLS+04].

Wrong finger table entries are the main problem with recursive routing. Assuming a peer-to-peer network where nodes join and leave frequently, fingers also change very often. As finger entries are not updated immediately, the probability of using a finger that is no longer participating in the network is significant. This results in the loss of query packets as they can not be delivered to the absent node. Again, the initiator monitors the lookup request with a timer. Whereas each hop can be monitored separately when using iterative routing, a node gets no information about the routing progress when recursive routing is employed. This is why the timer has to be chosen noticeably longer. In addition, the probability for contacting an absent node rises with increasing churn activity. Lookups therefore fail very often, but are noticed only after the global timer did expire.

Recursive routing can be improved by sending acknowledgments for each received packet (e. g. by using TCP/IP instead of UDP). Each node that forwards an query waits a certain time for receiving an acknowledgment that the packet has successfully reached its destination. If this acknowledgment is not received a local timer expires and the node assumes that the finger is no longer available and tries to send the packet to the next best finger. If transmission just took slightly longer and the first packet actually did arrive at its destination instead of being lost, the request is duplicated, resulting in a higher traffic. Therefore, the local timer must not be chosen to short to avoid a common duplication of the requests.

Worse are failures of a node during forwarding a query or waiting for the acknowledgment for a packet that is lost. Then, in all probability the recursive lookup process is stopped and has to be resumed from the beginning. As recursive routing requires a noticeable long timer compared to iterative routing, retries are started significantly later. Increasing the number of nodes participating in the ring and with it increasing the mean hop count would even require a much higher timeout. This would further extend the time until the initiator takes notice of a failure.

Additionally, each node can verify used fingers when it receives acknowledgements from them. Otherwise it can immediately start the fix finger procedure and repair the finger entry instead of waiting for the periodic fix finger invocation. This leads to a higher availability of finger entries and therefore reduces the probability of routing errors.

4 Hybrid routing

We suggest a hybrid routing protocol that combines the advantages from both routing procedures. It possesses the low delay of recursive routing in error-free circumstances and provides the faster failure recovery of iterative routing.

In our approach we present a recursive routing with a fundamental addition: at every hop a node sends an additional acknowledgment back to the originator of the initial request (see figure 3). Therefore, the originator constantly receives up-to-date information about the lookup path and where and when the routing failure occurs. This information for example could be used to skip the first hops of the logical lookup path and restart the routing from somewhere ahead. Every received acknowledgment restarts the timer used in the initiator,

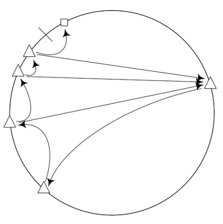

Figure 3: **Hybrid solution**: Each node forwards the query to the next node and sends an acknowledgement to the originator.

so the originator does not have to wait for the longer recursive timeout to expire when failures occur, but can work with the shorter timeout values used in iterative routing.

On the other hand, hybrid routing benefits from the lower delay of recursive lookup when routing is free of errors, and from the fact, that congestion control may be easier as shown in [DLS+04]. Furthermore, each node benefits from receiving direct information about the correctness of its fingers as explained in the previous section. Obviously, for N nodes in the lookup path we need $(2N-1)$ messages in iterative routing, N messages in recursive routing and $(2N-2)$ messages in the hybrid routing scheme.

5 Conclusions

Our hybrid routing solution combines the benefits from recursive and iterative routing. Except for a larger number of messages compared to recursive routing, significant advantages can be gained using this approach, such as lower routing delay due to a faster failure recovery or a complete overlook of the routing path by the originator.

References

[DLS+04] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, March 2004.

[ESZK04] J. Eberspächer, R. Schollmeier, S. Zöls, and G. Kunzmann. Structured P2P Networks in Mobile and Fixed Environments. In *Proceedings of the HET-NETs'04 International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks*, July 2004.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. Technical report, NYU's Computer Science Department, 2002.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan†. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Technical report, MIT Laboratory for Computer Science, 2001.