P2P - Theorie Aufgabe 5

Arne Link (1582381), Lars Fritche (1691285)
Gruppe 14 (S217/103)

January 20, 2014

1 Aufgabe 5.1

1.1 XOR metric

The XOR metric serves as a means to compute the distance between 2 nodes or keys in general, which allows all the knowledge about the group that the XOR arithmetic forms to be extended to certain aspects of the protocol, allowing for closed analysis of the P2P system, in contrast to being restrained to simulations.

Furthermore, this metric and implementation also not only allows to quickly find close nodes, but to use the same algorithm to find close and exact matches for a given key.

1.2 Routing table

The number of entries in the routing table of each node are up to 160 times the replication parameter (k, for example k = 20). For each value of i, with $0 \leq i < 160$, with i representing the XOR distance between two node ids, one k-bucket is allocated. As the k-buckets for low distances between node IDs are generally empty or sparsely populated, a considerable amount of routing entries will normally not be filled.

1.3 Lookup algorithm

For lookup, α nodes, closest to the searched key or node are chosen (via XOR metric). α is a global concurrency parameter, usually 3. Each of these nodes are then sent the FIND_{NODE} message, returning the k closest nodes to the target key. Upon receiving the list of nodes, the α nodes closest to the target that have not yet been queried are recursively sent the FIND_{NODE} message. If no node is closer to the target than the already queried nodes, the FIND_{NODE} message is extended to the k closest nodes not already queried. When all relevant nodes have been queried, the node(s) closest to the target have been identified.

This lookup is an iterative lookup system, as a certain number of nodes (α) are queried iteratively and in parallel, whereas the answer of these queries is usually not the final answer, but a referral to other nodes. Therefore, not one queried node will recursively search for the exact node, but the searching node will iterate on the answers and find nodes close and closed to the targeted node.

1.4 Update Policy

Upon receiving a request, the receiving node pings the least recently seen node in the bucket. If this node fails to respond, the failed node is removed from the bucket and the new node (sender node of the initial request) is inserted at the tail of the bucket. By pinging the least recently seen node and preferring that node instead of the new node if it is still alive, the entries in the bucket are more likely to be alive, given that nodes with an already high uptime can be expected to be alive even longer.

1.5 Kademlia and BitTorrent

Kademlia is used in some (optionally) tracker-less BitTorrent clients as a means of content discovery. One possible downside to this approach is that the actual content can not be determined before downloading and there are no means of marking such content as trustworthy, illegal, incomplete or just wrong by means of administration, comments, etc.

1.6 Sorting Buckets

Nodes in the sorting buckets are sorted from least recently seen node at the head, most recently seen node at the tail. Upon receiving a request, the following will happen:

1. Buckets not full: Add new node to the tail of the list (most recently seen).
2. Buckets full and bucket index not 0: Ping the least recently seen node (LRSN) (head of the list)
   - LRSN answers: Move LRSN to the tail of the bucket
   - LRSN does not answer: Remove LRSN node from the bucket and insert the new node to the tail (most recently seen)
3. Buckets full (otherwise): Split bucket in half, distribute existing nodes onto the two buckets and continue as normal.

By preferring old contacts over new nodes and never evicting living nodes, the percentage of probably life nodes is increased as nodes that are already alive a certain time are likely to stay alive even longer. Furthermore, only least recently seen nodes in the bucket are ever removed, keeping all frequently seen nodes in the bucket. Furthermore, because nodes are chosen very conservatively, basic DOS attacks are automatically mitigated.

2 Aufgabe 5.2

2.1 Kademlia vs. KAD routing tables

Kad uses k-buckets with 10 entries each and for levels of 0 to 4, each bucket can be split if a new node is contacted, not only those with a 0-index. This results in much more detailed routing tables and therefore decreases the average path length. The degree of the tree is therefore in more cases two, compared to plain Kademlia, and whereas in Kademlia every k-bucket with index $\neq 0$ has a degree of zero, in Kad this restriction is lifted for buckets with index $\in [0, 4]$

2.2 Routing table examples

Kademlia Figure 1

KAD Figure 2

2.3 Requests

HELLO_REQ Message to test if a contact node is still alive. Checked approximately every two hours if not checked otherwise (regular messages for example). Used to keep only live nodes in the k-bucket list, to avoid timeouts in later queries.

SEARCH_REQ Once a replica root is found, a SEARCH_REQ message with the search keyword is sent to the replica root. The answer to this message are the content pieces. If more than 300 are found (from one node only or from multiple nodes), no more SEARCH_REQ or KADEMLIA_REQ messages for this keyword are sent anymore. Acts as a means of content discovery.

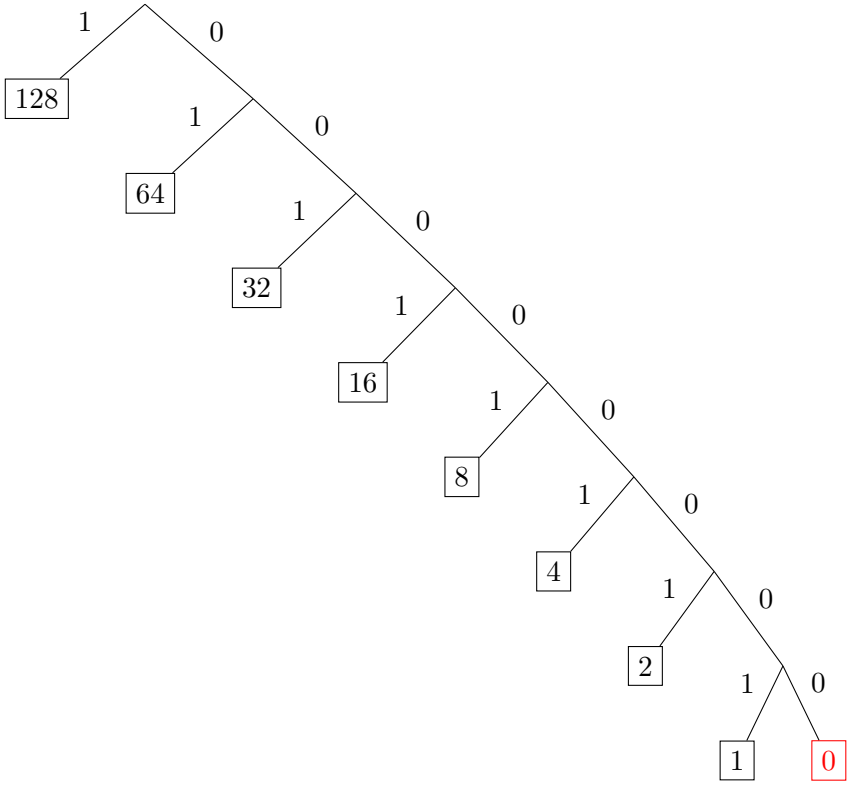

Figure 1: Kademlia routing table

KADEMLIA_REQ Used for discovery of nodes that have certain content. Contains MD4 hash of the keyword that is looked for. The response are more contacts that are closed to being replica roots for that keyword. A node is considered a match if its distance to the keyword is less than the search tolerance (a system wide constant).

2.4 Attacks

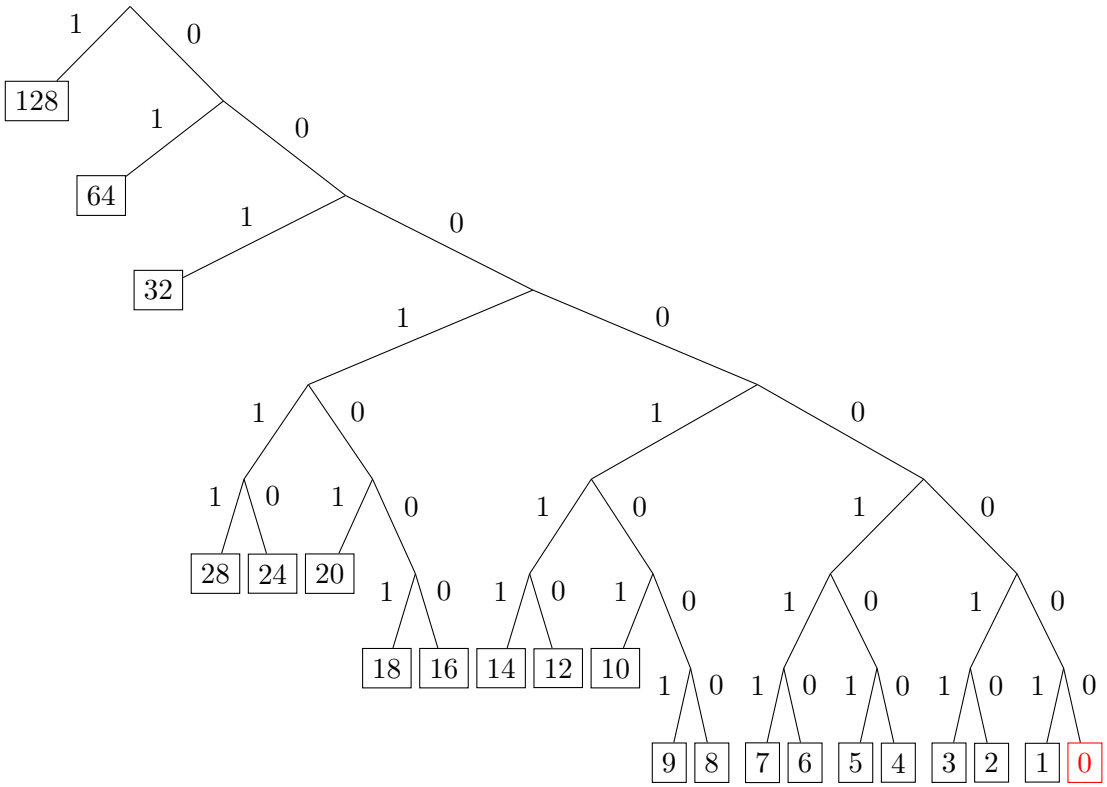

Figure 2: KAD routing table