Exercise 3

Exercises Peer-to-Peer-Systems and Security (SS2011)

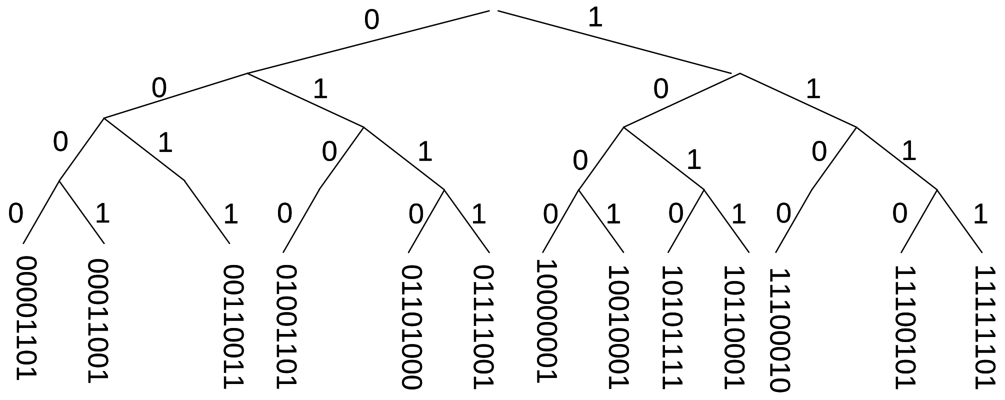
Monday 6.6 2011
Hand-in: Thursday 16.6. 2011 in lecture or per mail
Exercise: Monday 20.6. 2011

Dr. Heiko Niedermayer
Lehrstuhl für Netzarchitekturen und Netzdienste
Technische Universität München

Rules: There will be five exercise sheets. You have to hand-in 70 % of the assignments, attend atleast 3 exercise courses and present a solution in the exercise course to get the 0.3 bonus..

Task 1 Kademlia

Now, we simulate the operation of Kademlia. Bucket size is k=2. Alpha is 2. The IDs are 8 bit long.



a) You want to store item 01000001. Calculate the distance (in bits and decimal notation) according to the XOR metric to the nodes 00110011, 01001101 and 10101111. Which node is responsible for the item?

b) Node 11010101 joins the network. Node 00110011 is the rendezvous peer that node 11010101 uses to join. Describe the operations of the join operations over time including the filling of the buckets.

Solution:

a)

```
Distance to     00110011
                01000001
                ------------
XOR             01110010 = 2+16+32+64 = 114

Distance to     01001101
                01000001
                ------------
XOR             00001100 = 4+8 = 12

Distance to     10101111
                01000001
                ------------
XOR             11101110 = 2+4+8+32+64+128 = 238
```

The Item is assigned to noden 01001101.

b)
Node 11010101
Buckets: 11010101, 1101011x, 110100xx, 11011xxx, 1100xxxx, 111xxxxx, 10xxxxxx, 0xxxxxxx

1. Rendezvous peer 00110011
-> add to Bucket 0xxxxxxx
-> 2 nodes for the bucket 1xxxxxxx, e.g. 11111101 and 10101111

2. Continue towards ID
-> 11111101 replies with his 2-Bucket 110xxxxx, which is 11000010.

- Add 11111101 to bucket 111xxxxx .

-> 10101111 replies with 2 nodes from his 2-bucket 11xxxxxx, e.g. 11000010 und 11100101

- Add 10101111 to Bucket 10xxxxxx

3. Continue towards ID
-> 11000010 just returns neighbors that are further away, also 11100101 und 10110001.

- 11000010 to Bucket 1100xxxx.

-> 11100101 returns 11000010 and another close node 10101111.

- 11100101 to Bucket 111xxxxx.

We did not find new nodes, so we reached the destination.

Buckets:

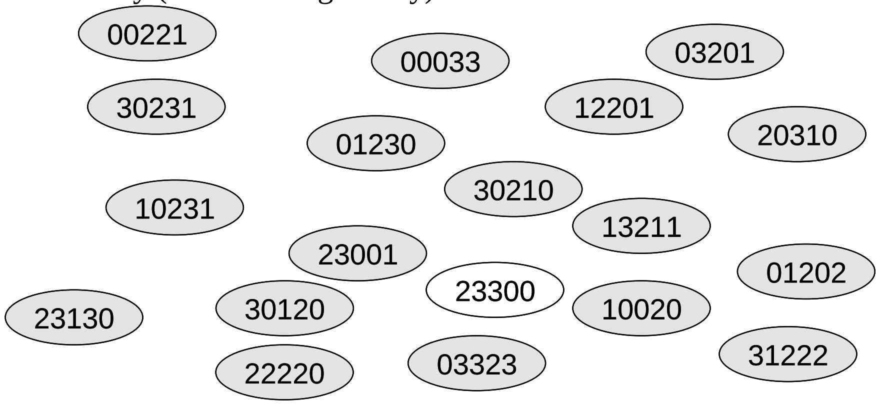
0xxxxxxx	: 00110011
10xxxxxx	: 10101111
111xxxxx	: 11100101, 11111101
1100xxxx	: 11000010, -
11011xxx	: -
110100xx	: -
1101011x	: -
11010101	: -

4. Refresh of the buckets = ask for a random ID from the bucket.

Buckets:

0xxxxxxx	: 00110011, 00001101
10xxxxxx	: 10101111, 10000001
111xxxxx	: 11100101, 11111101
1100xxxx	: 11000010, n/a
11011xxx	: n/a
110100xx	: n/a
1101011x	: n/a
11010101	: n/a

Task 2 Pastry
This task is about Pastry. The figure is a snapshot with the current nodes (dark) and the new node (white) and how they are positioned in the underlay. If they are close in the figure they are close in the underlay (short message delay). Assume that the size of L and M is 2.



a) The white node 23300 wants to join the network via the node 10020. Describe the join process step-by-step and state the corresponding routing information seen and stored by the white node. What is the routing table of 23300 at the end? (Hint: There is no need to calculate each routing table, but simply make reasonable assumptions.)
b) Now, send a message from 23300 to ID 03023. For the first step use the routing table from a) and then make reasonable assumptions.

Solution:
a) N|=|L|=2

Step 1: Interaction with 10020

Set own Neighbor Set to the Neighbor Set of 10020
N_23300 = {10020,13211}

First row of routing table can be taken from 10020:
- ⇨ to 0* via 00221, to 1* via 10020, nto 2* n/a (own prefix), to 3* via e.g. 30120
- ⇨ 10020 could have 23130 in 2*

Step 2: Interaction with 23130
Second row of routing table can be taken from 23130:
- ⇨ to 20* via -, to 21* via -, to 22* via 22220, to 23* n/a (own prefix),
- ⇨ 23130 also fits to next step

Third row of routing table can be taken from 23130:
- ⇨ to 230* via 23001, to 231* via 23130, to 232*-, to 233* n/a (own prefix),
- ⇨ Done

Leaf set L_23300 = {23130,30120}

b)
1. step: to 0*
   Send packet to 00221
2. step: to destination
   00221 sends packet to destination via its leaf set 03201 (based on numeric distance)
3. Destination replays to 23300

Task 3 Key-based-Routing-API

Please, briefly describe your idea first, before you write pseudocode. Use a pseudocode that avoids unnecessary details. Assume that you have a structure Peer-to-Peer system that implements the Key-based Routing-API. All messages are processed recursively (e.g. no iterative lookup):

a) *Ping and Pong:* a node A pings (sends "Ping") a node B via the KBR network by sending message to its ID. Node B replies to the ID of node A with a "Pong". Give the code for the send and receive operation.

b) *Counter:* implement a counter that counts the number of messages that the node forwards.

Solution:

a)

```
void ping(Key x){
        route(x,"ping "+myKey,null);
        }
In  the event routine
void deliver(Key x, Message m)
        {
        if (m is a ping)
                {
                route(m.getID(),"pong"+myKey,null);
                }
        if (m is a pong)
                {
                deliverPong(x,m); // notify application or maintenance if of interest
                }
        }
```

b)

```
Initialize counter in the class
int messagecounter = 0;

In the forwarding event routing
void forward(Key x, Message m, NodeHandle nextHopNode)
        {
        messagecounter++;
        }
```

Task 4 CoolSpots – specific structure
Propose a structured network that is not a DHT for the CoolSpots network. It should be possible to

- efficiently look-up / store a GPS coordinate
- efficiently support range queries like all spots close to the coordinates on the way from Garching Forschungszentrum to Munich Marienplatz. It should be efficient in the sense that the cost is dominated by either the look-up (O(logn)) or the number of responsible nodes for the area (all nodes holding relevant data for the range query).
- the state for the routing should be O(logn)
- the required state for replication of a spot or similar measures should be below a small constant k per spot (e.g. k = 10).

Describe

a) the structure
b) the look-up operation and the storage operation

Solution:

a)
Optimal would be to use a map-like solution, like a 2d-CAN. This can be used for basic connectivity and maintenance. Long-distance links in exponential distance (divide ID space into 4 squares, link to the 3 squares you are not in, repeat the process for the s quare you are in (~ Pastry)) would be helpful to reach the O(logn) goal (one could also follow ideas from Skip Graphs).
For robustness multiple peers per interval should be used.

b)
standard query with the CAN links and long-distance links to closest node of ID. Responsibilities are assigned CAN-alike (rectangles in the ID space)
Storage: store item (all data) on closest node in ID space
Lookup: route to closest node in ID space and retrieve item

Task 5 CoolSpots – specific structure
On the basis of your proposal from task 4, describe

a) a maintenance operation for the structure
b) an example range query for GPS coordinates from Garching to Munich Marienplatz

Solution:
a)
Join: like CAN, if ID in wrong interval, use parallel peers for same interval. Split interval if enough peers in both intervals after split would exist, share data

Maintainance (after leave): regularily check peers in zone and direct CAN links, from time to time check distance links / lookup fitting distant node, if not enough nodes in zone, merge with neighbour zone (which was split to create the two zones)

b) If the areas are too small, then one needs to scan the neighbors orthogonal to the direction of the path, here also check nodes west and east of the path. A scan like this with g being Garching, m being Marienplatz and x being a scanned node, * being an ignored node.

```
***************
**** xgx *******
**** xxx *******
**** xxx *******
**** xxx *******
**** xxx *******
**** xmx *******
****************
```