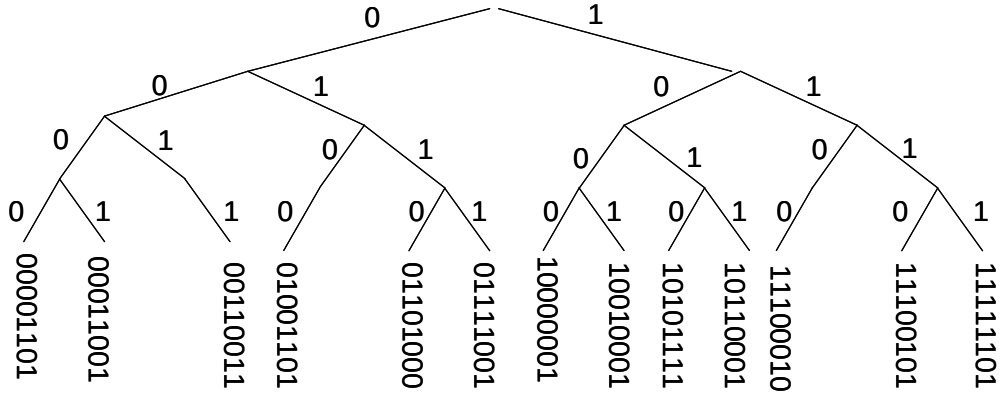
Exercise 3

Exercises Peer-to-Peer-Systems and Security (SS2012)

Thursday 21.6 2012
Hand-in: Thursday 5.7. 2012 in lecture or per mail
Exercise: Monday 9.7. 2012

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

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.

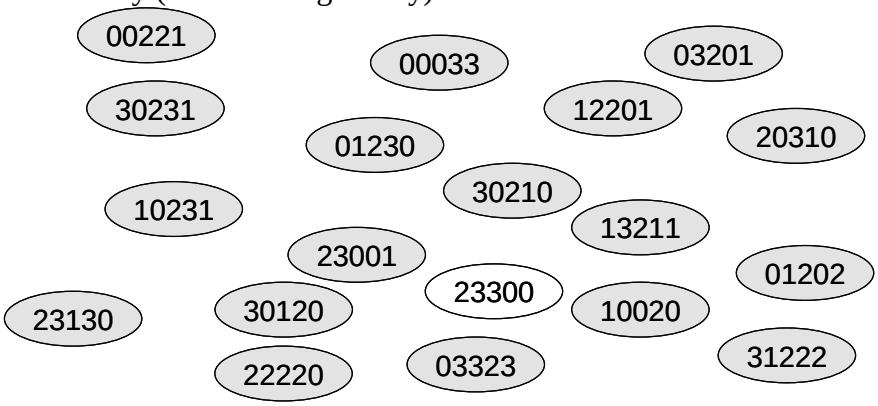
Buckets:
0xxxxxxx : 00110011
10xxxxxx : 10101111
111xxxxx : 11100101, 11111101
1100xxxx : 11000010, -
11011xxx : -
110100xx : -
1101011x : -
11010101 : -

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}

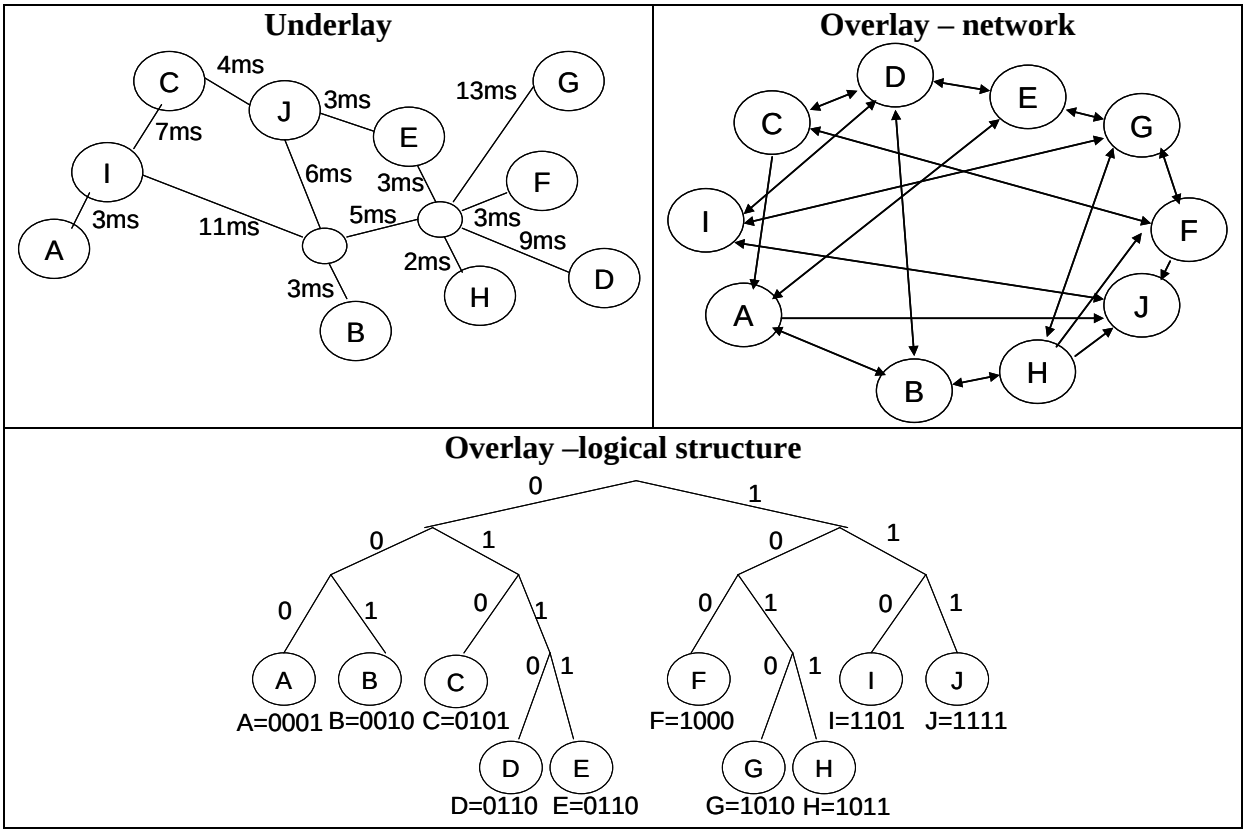
b)

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 Stretch and Proximity Neighbor Selection

The Figure below shows the underlay including the latencies along different paths. The second image is the structure of the overlay (nodes that link to each other) and the image on the bottom is the logical structure if we assume that our network uses a routing table like in Pastry, but without neighbor and leaf set.

In this task, stretch always refers to latency. Always state what path is used when you calculate the stretch.



a) What is the stretch (with respect to latency) for queries from F to A and from B to E?

b) Assume that each node on the way now applies PNS for its routing table. This means that it selected the latency-optimal node in each subtree. What is now the stretch for the queries from a)?

Solution:

a)
F->A:
- direct: 22 ms
- via overlay (via C): 13 + 10 ms
Stretch_FA = 23/22 = 1,045

B->E
- direct: 11 ms
- via overlay (via D): 29 ms
Stretch_BE = 29/11 = 2,63

b)
F->A
PNS from F to subtree 0x: to E in 6ms

PNS from E to subtree 00x: to B in 11 ms
B->A: in 17ms
Stretch_FA_PNS = 34 / 22 = 17 / 11 = 1,54 (got worse due to more hops!)

B->E
PNS from B to subtree 01x: E in 11 ms
Stretch_BE_PNS = 1,0

Please note: in task b) the goal was to change the overlay structure according to PNS using the logical structure. So, for F you have to look what is the best node in 0* and then select this node as F's neighbour in the new overlay structure after PNS. Here, we assumed that the best nodes of all nodes in the corresponding intervals are selected. In reality one may check the distance (e.g. RTT) only to a fixed number of nodes and select them the best among them for the link.

Task 4 Fighting Hotspots in Chord

From the lecture you should know two things. First, Chord proposes to replicate items to the k successors of the item ID for resilience. Second, this successor list as replica set cannot be hit by queries for the item. Thus, this does not help to fight hotspots (popular items are served by multiple nodes).

Modify Chord, so that all nodes in the replica set are utilized to answer queries for an item (Please note: This means that instead of using the successor list, do something different). Argue that your solution reaches this goal.

Solution:

Variant 1:

- Use root node (node holding the interval which fits to the item ID) and k predecessors for this instead
- When root node leaves, item has to be transferred to the successor of the last predecessor node.
- Regularily check if k predecessors hold the item and if you still are among the k closest predecessor
- Since all predecessors can be hit on the path to the root node, many queries will be answered by the predessesors. The root node will usually not be reached by queries since it can only be reached via its direct predecessor

Variant 2:

- Modify Chord so that k nodes share an ID and thus an interval with its items. All of these nodes are interconnected. In the routing table of other nodes, all of these nodes randomly appear, e.g. by always publishing all nodes in an interval in maintenance queries and the node selects one of them randomly for its finger (variant: bucket instead of one node per finger).
- Since the holders of the preceeding interval are always hit, for item balancing they should be updated more frequently (like in Chord Stabilization).
- Such a solution needs a maximum and minimum number of nodes that share an interval. Splitting and merging of intervals will become necessary.
- All nodes are hit by queries for their items if they are known by the predecessor nodes. For queries to other items, all nodes are hit if they are found in some routing tables.

Variant 3:

- Add replica on last node that did not have it in a query. Like in Kademlia.
- This build a tree of replicas of nodes with links towards the root.

  root – predecessor – linksToPredecessor – linksToPredecessorOfPredecessor …
  – linksToPredecessorOfPredecessor …
  – linksToPredecessorOfPredecessor …
  – linksToPredecessor – linksToPredecessorOfPredecessor …
  – linksToPredecessorOfPredecessor …
  …
- Here also, the root node will not be hit, once its predecessor has a replica.

Task 5 Authentication

In this task we specify a cryptographic protocol which is meant to be used for mutual authentication.

a) Specify on what information and when in the protocol do the entities A,B, and S detect a successful authentication run?
b) The protocol is insecure. Find an attack. (The strength of the attacker is that it can read, send, fake, and drop messages in the network, yet it cannot break cryptography. This is a common security model in network security.)

Protocol for Task 2:

Prerequisites:
S is a TTP.
Each participant X has a shared key with S. This key is called k kXS.
Let kab = Nb.
Protocol:

```
A -> B: Na, A
B -> S: {Na, A, B, Nb}kBS
S -> A: B, {Na, Nb}kAS
A -> B: {Nb}kab
```

Solution:

A recognizes B: in step 3, the server sends a message which contains Bob and Na
B recognizes A: in step 4 by receiving Nb from A which was protected with the shared key kab
S does not detect the success of a run. It might recognize B on the knowledge of the kBS encrypted and integrity protected). It does not recognize A, but provides information on Nb that only A should be able to read.

Attack:

```
C is the attacker, C_X means C plays X.
   1. A -> C_B: Na,A // C intercepts message to B
   2. C -> S: {Na,A,C,Nc}kCS
   3. S -> C_A: C, {Na, Nc}kAS // intercepted by C
   4. C_S -> A: B, {Na, Nc}kAS
   5. A -> C_B: {Nc}kab  // intercepted by C
   C can now impersonate B towards A.
```