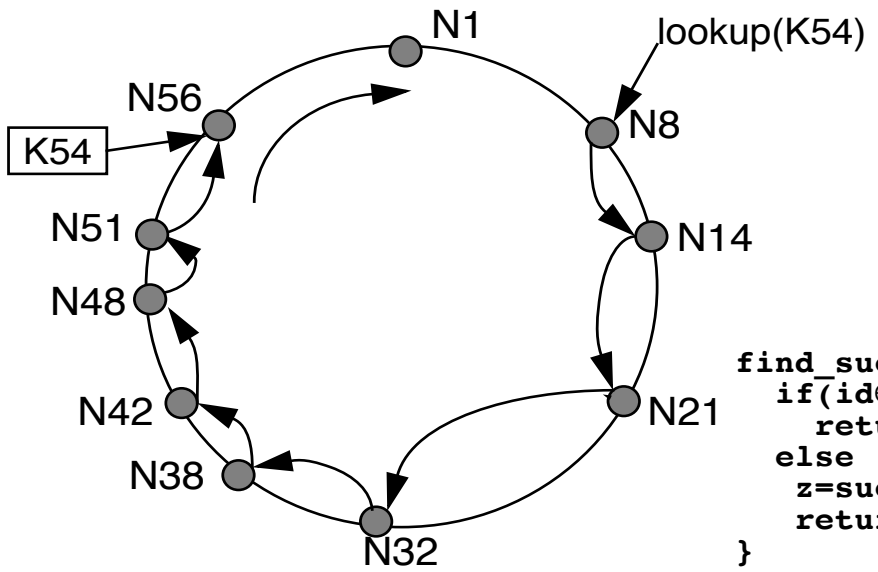
UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Dependable Distributed Systems – 5880V/UE

Part 7b. Peer-to-peer systems – 2021-10-13

Prof. Dr. Hans P. Reiser | WS 2021/22




```
find_succesor(id) {
  if(id∈ (n,succesor])
    return successor
  else
   z=successor.find_successor(id)
   return z
}
```

Overview

1. Overview of peer-to-peer systems
   - Definition
2. Unstructured peer-to-peer systems
   - Napster, Gnutella, ED2K
3. Structured peer-to-peer systems
   - Distributed hash tables (DHT)
   - Chord
   - Kademlia

A possible definition

David Barkei:

- *"Peer-to-Peer computing is a network-based computing model for applications where computers share resources via direct exchange between the participating computers"*

A possible definition

Properties that characterize a distributed system as “peer-to-peer” system:

- There is a shared utilization of resources
  - Each component of the system can act as information provider and information consumer
- Each component has some degree of autonomy
  - No central control or use of services
- It is not necessary for all components to be permanently part of the system
  - Self-organisation of the system
- Components do not need permanent network addresses
  - Addressing of nodes is independent of network layer

A possible definition

(Better?) definition from tutorial at KIVS 2003:

- *“Selbstorganisierendes System gleichberechtigter, autonomer Systeme ohne Nutzung zentraler Dienste auf der Basis eines unzuverlässigen Netzwerks”*

A possible definition

(Better?) definition from tutorial at KIVS 2003:

- *"Selbstorganisierendes System gleichberechtigter, autonomer Systeme ohne Nutzung zentraler Dienste auf der Basis eines unzuverlässigen Netzwerks"*

Characteristics:

- Self-organization
- Equal, autonomous components
- No central services
- Unreliable network

Requirements for peer-to-peer systems

- Global scalability
- Simple dissemination of information to arbitrarily large number of recipients
- Fast and efficient localization of information
- Participants can join and leave the system continuously at arbitrary moments

What is called “peer-to-peer” in practice?

- File sharing systems
  - Napster, Gnutella, eDonkey
- Distributed file systems
  - Freenet, PAST, Oceanstore
- Instant Messaging
  - ICQ, AIM, Jabber
- Distributed computing
  - Grid computing, SETI@home, Folding@home, Popular Power
- Collaboration
  - Collaborative software, groupware
- Content delivery networks
  - BitTorrent, PeerCDN, Swarmify

Classification

Client/server: WWW, SETI@home

- Traditional client/server structure
- No direct interaction between clients

Hybrid P2P systems: Napster, ICQ, AIM

- Joint utilization of shared resources
- Interaction between peers
- Coordination by a central server

Pure P2P systems: Gnutella, Freenet, Chord, . . .

- Fully decentralized organization and utilization of resources

Overview: Classification of pure P2P systems

Unstructured peer-to-peer systems

- No predefined organizational structure
- Examples: Gnutella, Freenet

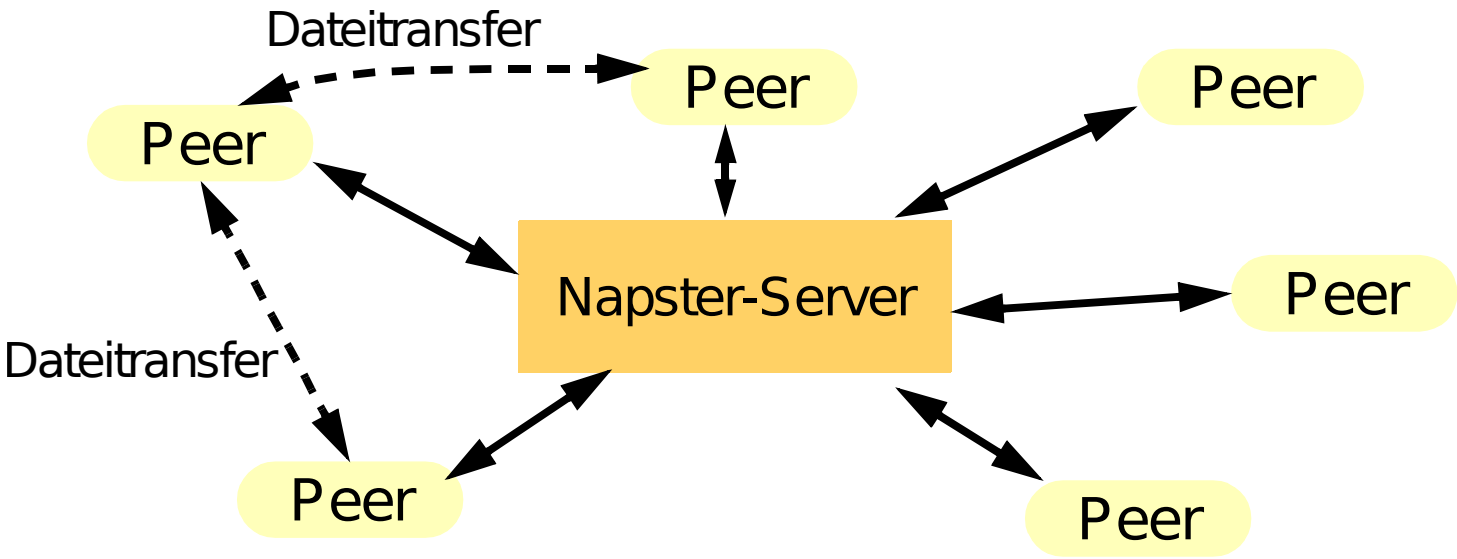
Structured peer-to-peer systems

- Implement a distributed hash table (DHT)
- Each node has a unique identifier
- There is a structural model that defines the topology of the system
- Examples: Chord, Can, Pastry, Kademlia

Overview

Napster: first popular P2P system (1999-2001)

- Purpose: Exchange of music files



- Central directory
  - Manages addresses and file lists of peers

Napster: basic principle

Connection to directory server

- Peers publish their IP address and list of files

Searching for files

- Query to directory server, reply: list of IP addresses

Probing of potential peers

- Direct ping message for checking the connection quality

File transmission

- Direct transfer from peer with best connection

Napster: discussion

Properties:

- Limited scalability due to central directory server
- Directory server is single point of failure
- Hybrid peer-to-peer system

Extensions:

- Multiple interconnected directory servers (OpenNap network)
- Support for arbitrary file formats

Gnutella: decentralized file search

Initially developed by Nullsoft as a kind of “free Napster”

- Literature: The Gnutella Protocol Specification v0.4

Basics

- Joining: need IP address of at least one member
- Communication via TCP
- Usually, each peer has multiple permanent connections to others
- Each participants decides which files to offer to other peers
- Gnutella protocol used for searching, HTTP for file transfer

Gnutella: messages

Search requests

- Flooding of Gnutella network with Query message (limited by TTL)
- Hit message is returned via reverse path

Maintining network structure

- Flooding of network with Ping message
- Reply (Pong message) allows peer to learn about other peers

File transmission

- Directly via HTTP
- Push message to circumvent firewalls

Gnutella: evolution

First generation

- All nodes have identical number of connections
- Nodes with bad connectivity become overloaded, system collapses

Second generation

- Number of connections determined by available network capacity
- Connections to overloaded peers are closed

Third generation

- Introduction of hierarchies: combination of pure and hybrid peer-to-peer systems
- Dynamically determined ultrapeers take over work for peers with bad connectivity
- Similar to FastTrack

FastTrack / Kazaa

Developed with the following goals:

- Avoid utilization of a central server
- Avoid Gnutella's problems

Introduction of Supernodes

- "Simple" peers are connected only to supernodes
- Supernodes know file lists of connected simple peers and act as a proxy
- Supernodes are selected dynamically according to available network connectivity and CPU power

"Pure" peer-to-peer system, which evolves towards hybrid P2P systems

ED2K

Basis: architecture of Napster

- Server for managing file lists and handling searches
- Arbitrary number of servers, but explicitly designated (dedicated server software)

Additional measures to improve performance

- Transmission of file fragments (including transmission from multiple peers)
- Reputation system for participants

Summary of unstructured networks

Development of unstructured networks

- Conceptual approximation of hybrid and pure peer-to-peer systems
- Hybrid systems: Becoming less dependant on central server and increased performance by using many redundant servers
- Pure systems: Performance increase using dynamically established hierarchies

- Goal 1: Integration of each node according to its capacities
- Goal 2: Optimisation of search operations (maximize coverage, minimize costs)

Overview

Distributed hash tables

Basic tasks in peer-to-peer systems:

- Publishing information (where to store it?)
  - Publish(“content”, . . .)
- Locating and retrieving information (where is it stored?)
  - Lookup(”content”)
- Efficient operation (metrics: communication, memory, etc.)
- **Robustness against**
  - **crashes**
  - **joining/leaving nodes (“churn”)**

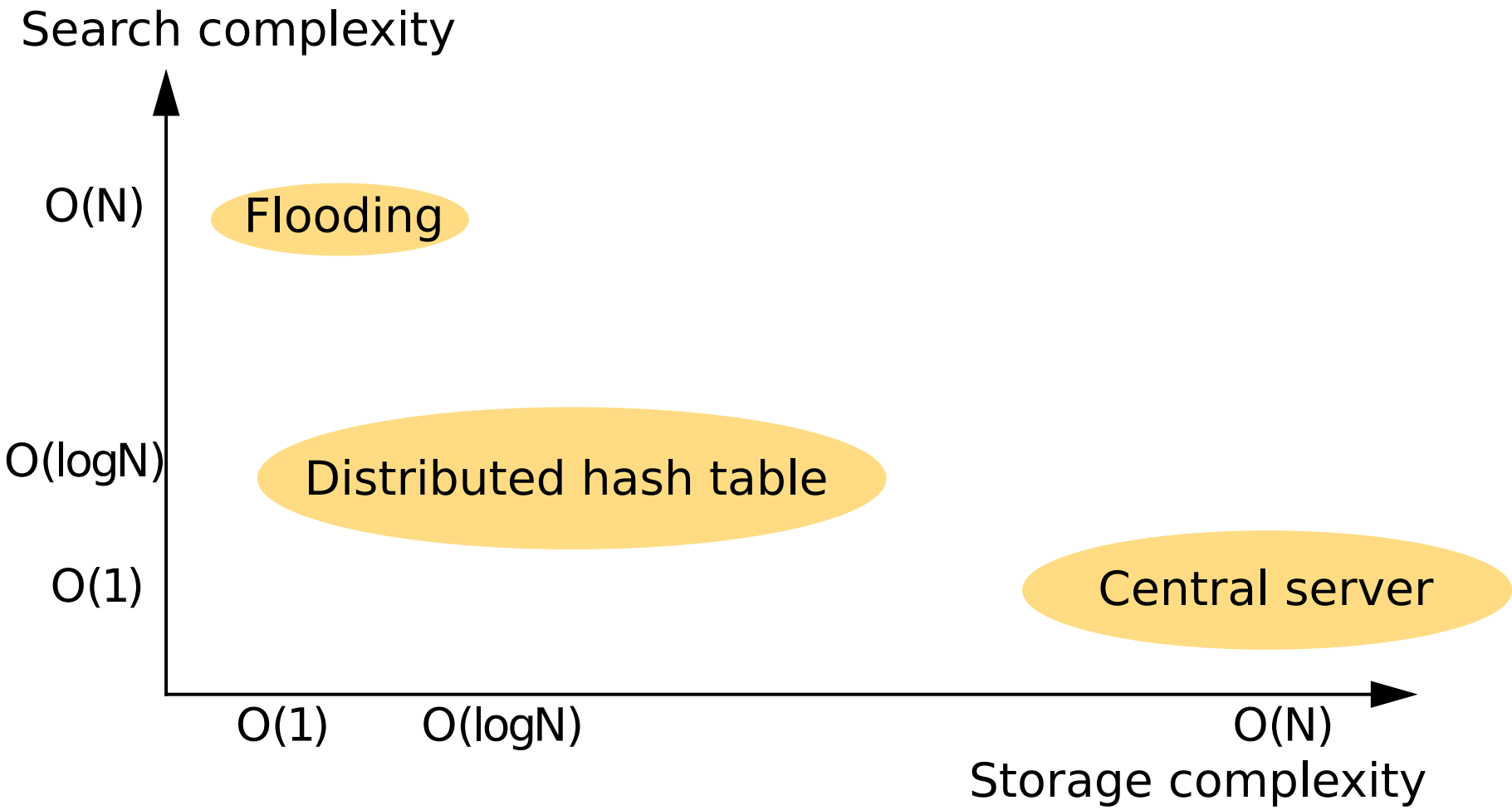
Distributed hash tables

Possibilities how to search for information in a distributed system

- Central server
  - Server stores location information
  - Well-known problems: scalability, outdated information, single point of failure
  - Usually the best approach for simple applications
- Search by flooding (such as in Gnutella)
  - Fully decentralized information
  - Bad scalability, high communication costs
- Distributed hash table
  - Goal: some compromise in-between the extremes
  - Efficiency: usually scales with $O(\log(N))$
  - Highly resistant against changes (crashes, attacks, churn)

Distributed hash tables

Search complexity vs. storage complexity



Distributed hash tables

Basic idea of distributed hash tables

- Systematically distributed data over all nodes
- Request data from node responsible for that data

Goals

- Even distribution of data on all nodes
- Continuous adjustment if nodes crash, join, or leave
  - Assignment of responsibility for some data to newly joining nodes
  - Redistribution of responsibility if nodes leave or crash

Distributed hash tables

Basic mode of operation

- Mapping of data to a linear key space
  - Usually $0 \ldots 2^{m-1} \gg$ number of objects stored in the system
  - Mapping of content to key space using a hash function
- Distribution of key space over all nodes

Overview

Chord: one of the first DHT protocols

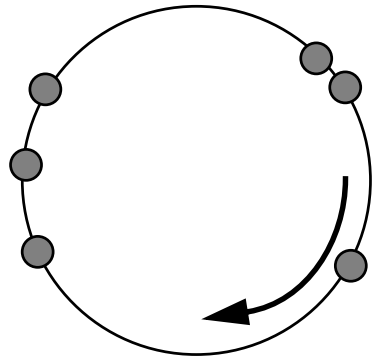
Literature

- URL `https://github.com/sit/dht/wiki`
- Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan: *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications*. IEEE Transactions on Networking, 2003
  (and prior publication on SIGCOMM 2001)

Chord

Basic structure: Chord ring

- Each node has a unique 160 bit ID on a ring
  (obtained by hashing the IP address)
- All nodes are mapped to a point on a ring, ordered by their ID

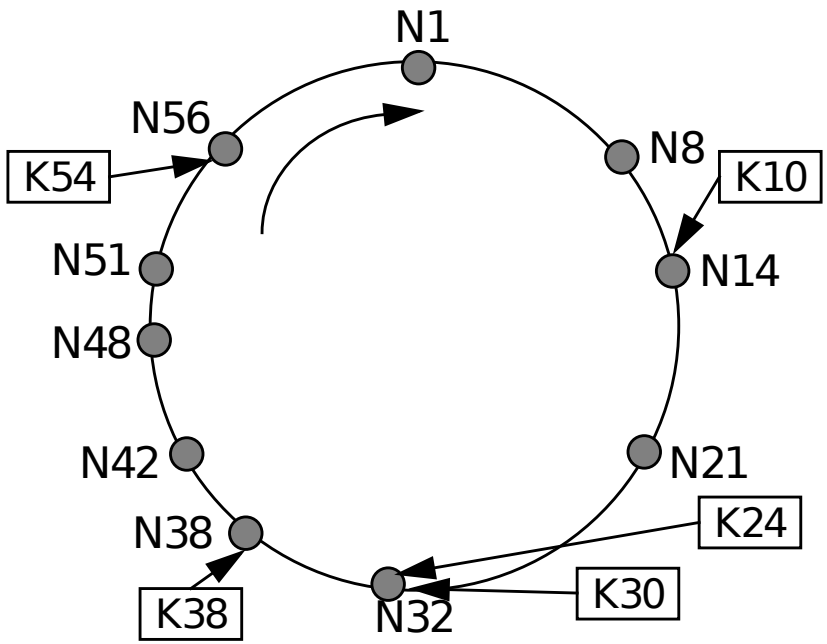


Chord

Mapping data to nodes on the ring

- For each data stored in Chord, an ID is created
- Data with ID i is stored on node with smallest ID $\geq i$

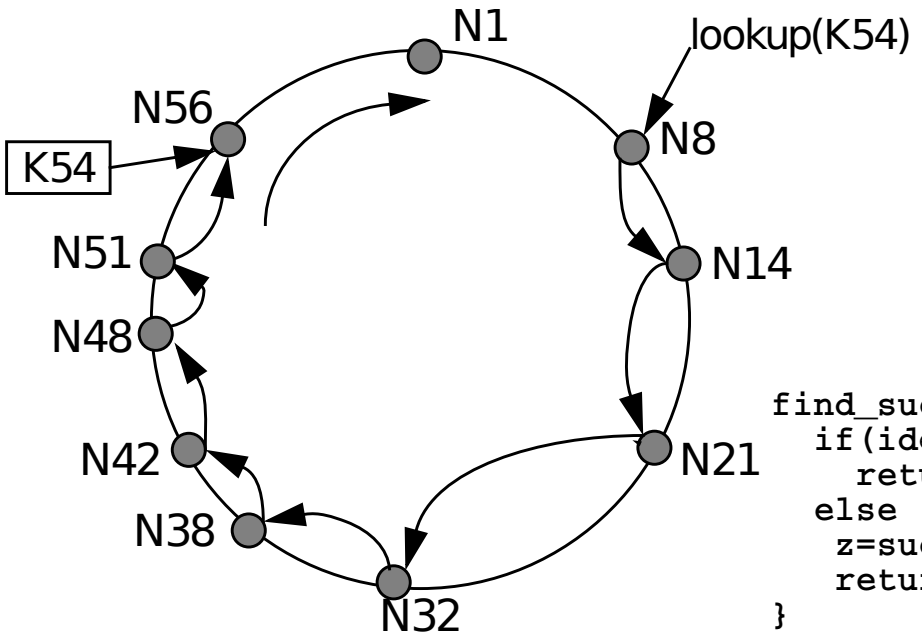
Example: Key size $m = 6$ bit, 10 nodes, 5 data items



Chord

Naive search operation on Chord ring:

- Each node knows the direct successor on the ring
- Each node recursively forwards request on ring until node with node ID > search ID > predecessor node ID is found
- Resulting node is responsible for the search ID




```
find_succesor(id) {
  if(id∈ (n,succesor])
    return successor
  else
   z=successor.find_successor(id)
   return z
}
```

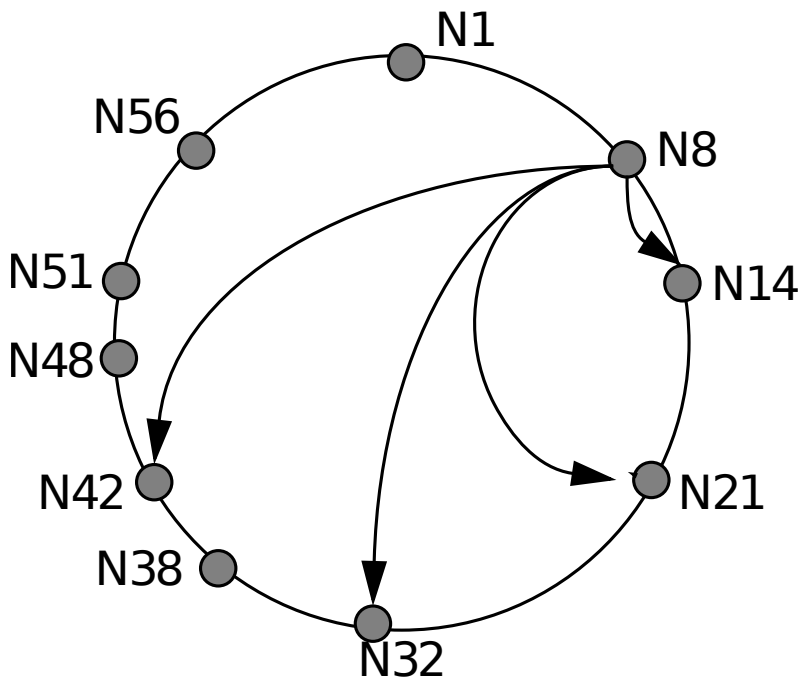
Chord

For efficient search on Chord, a node with ID n maintains the following routing information:

- The ID and IP address of the immediate successor on the ring
- The ID and IP address of the immediate predecessor on the ring
- A *finger table*, which contains "finger" pointers to nodes responsible for ID $(n + 2^{k-1}) \bmod 2^m$, for $1 \leq k \leq m$.
  The finger table contains node ID and IP address

Chord: Finger table

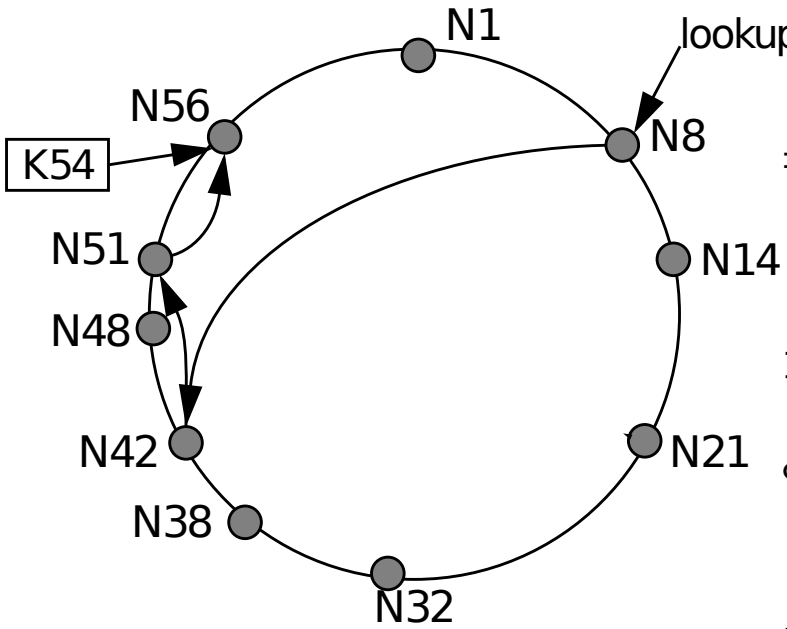
Example: finger table of node ID 8 ($m = 6$)



ID	Responsible Node
8+1	14
8+2	14
8+4	14
8+8	21
8+16	32
8+32	42

Chord: Finger table

Searching for ID using finger table:




```
find_succesor(id) {
  if(id ∈ (n,succesor])
    return successor
  else
    n'=closest_preceding_node(id)
    return n'.find_successor(id)
}

closest_preceding_node(id){
  for i = m downto 1
    if(finger[i] ∈ (n,id))
      return finger[i]
  return n
}
```

- Search complexity: $O(\log n)$
- Storage complexity: $O(\log n)$

Chord: maintaining routing information

- Create new Chord ring (node ID is *n*):

```
create(){
    predecessor = nil
    successor = n
}
```

- New node *n* joins Chord ring (*n* knows a ring member *n'*)

```
join(n'){
    predecessor = nil
    successor = n'.find_successor(n)
}
```

Chord: maintaining routing information

Routing data structures are updated by periodically executing a stabilize() method

```
stabilize() {
   x = successor.predecessor()
   if( x ∈ (n,successor))
      successor = x
   successor.notify(n)
}

notify(n') {
   if (predecessor is nil or n' ∈ (predecessor,n))
   predecessor = n'
}

check_predecessor() {
   if(predecessor has failed)
      predecessor = nil
}
```

Chord: maintaining routing information

Similarly, each nodes periodically updates its finger table

```
fix_finger() {
    next = next + 1
    if (next > m){
        next = ⌊ log(successor - n) ⌋ + 1
    }
    finger[next] = find_successor(n + 2^(next-1))
}
```

Chord: further optimizations

Each node stores the r next successors (instead of just one)

- Ability to tolerate the crash of up to $r - 1$ successor nodes
- Extended `stabilize` function updates successor lists with information obtained from successors
- If timeouts occurs while contacting node in `find_successor`, advance to next suitable node in successor list

Optimizing network communication

- For each finger table entry, maintain a list of nodes
- Select node according to best network connectivity

Overview

Kademlia

Basic properties

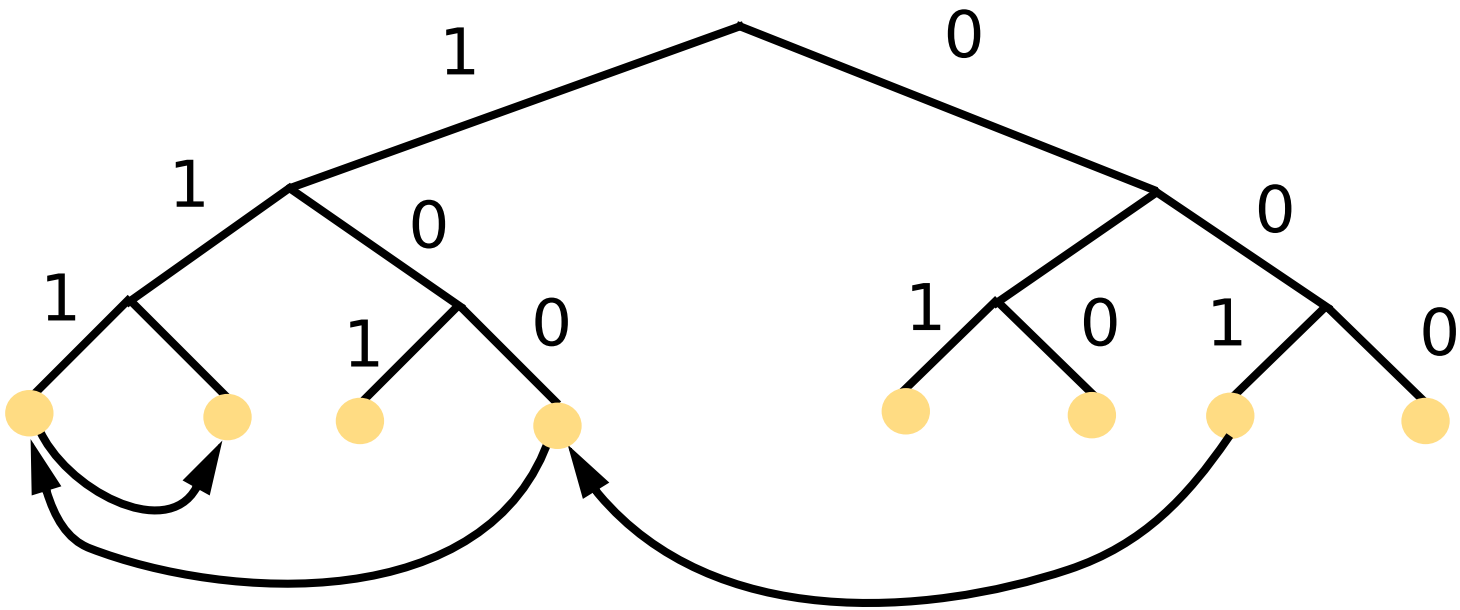
- Topology defined by XOR distance metric
  (vs. distance on ring in Chord)
- Avoids additional message for maintaining routing data structures
- Parallel execution of request
- Probabilistic routing

Literatur:

- P. Mayamounkov, D. Mazieres: Kademlia: A Peer-to-peer Information System Based on the XOR Metric (IPTPS 2002)

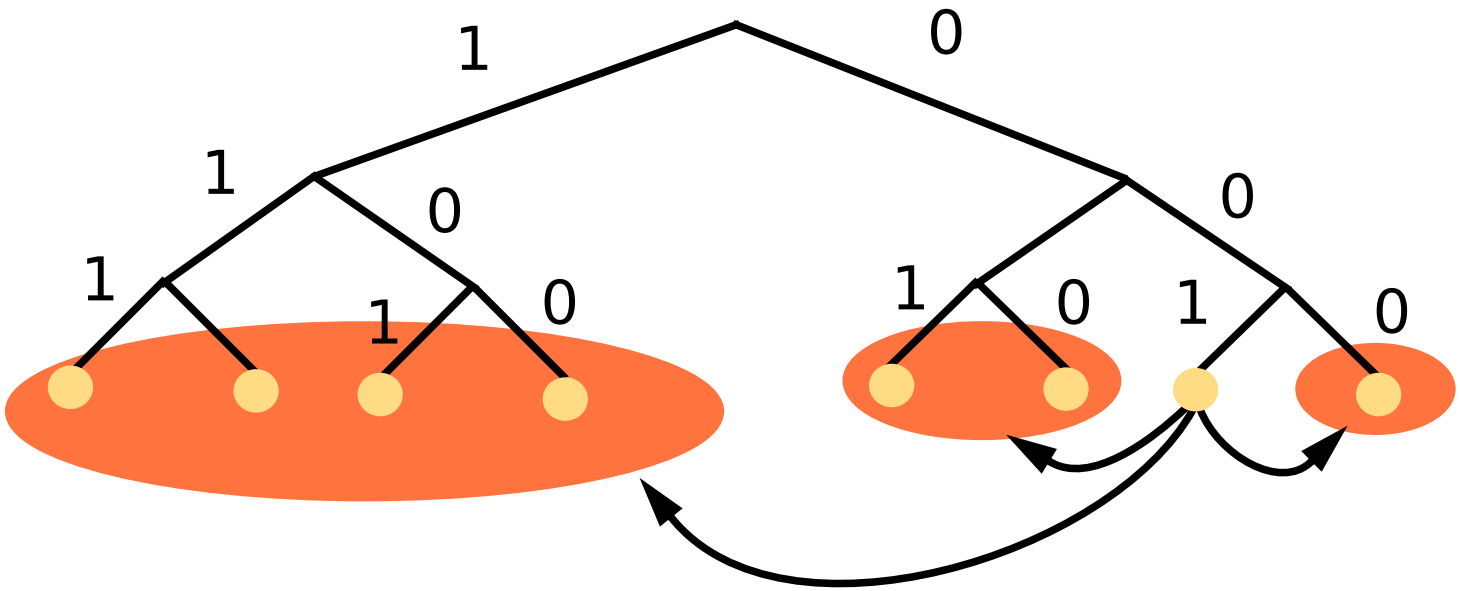
Kademlia: basic idea

- Digits of numeric node ID define a tree structure (image below: for binary representation)
- Iterated search for ID, in which for each iteration the XOR distance is reduced by at least one digit
  Example: node 001 searches ID 110 by going $\rightarrow 1xx \rightarrow 11x \rightarrow 110$



Kademlia: basic idea

- Each node knows at least one node in each (non-empty) subtree
- Example: Node with ID 001 knows nodes in subtrees with prefixes 1xx, 01x, 000



Kademlia: routing information

Node IDs

- Node IDs are 160 bit numbers (created by hash function)
- For each contact, Kademlia stores tuple (node ID, IP address, UDP port)

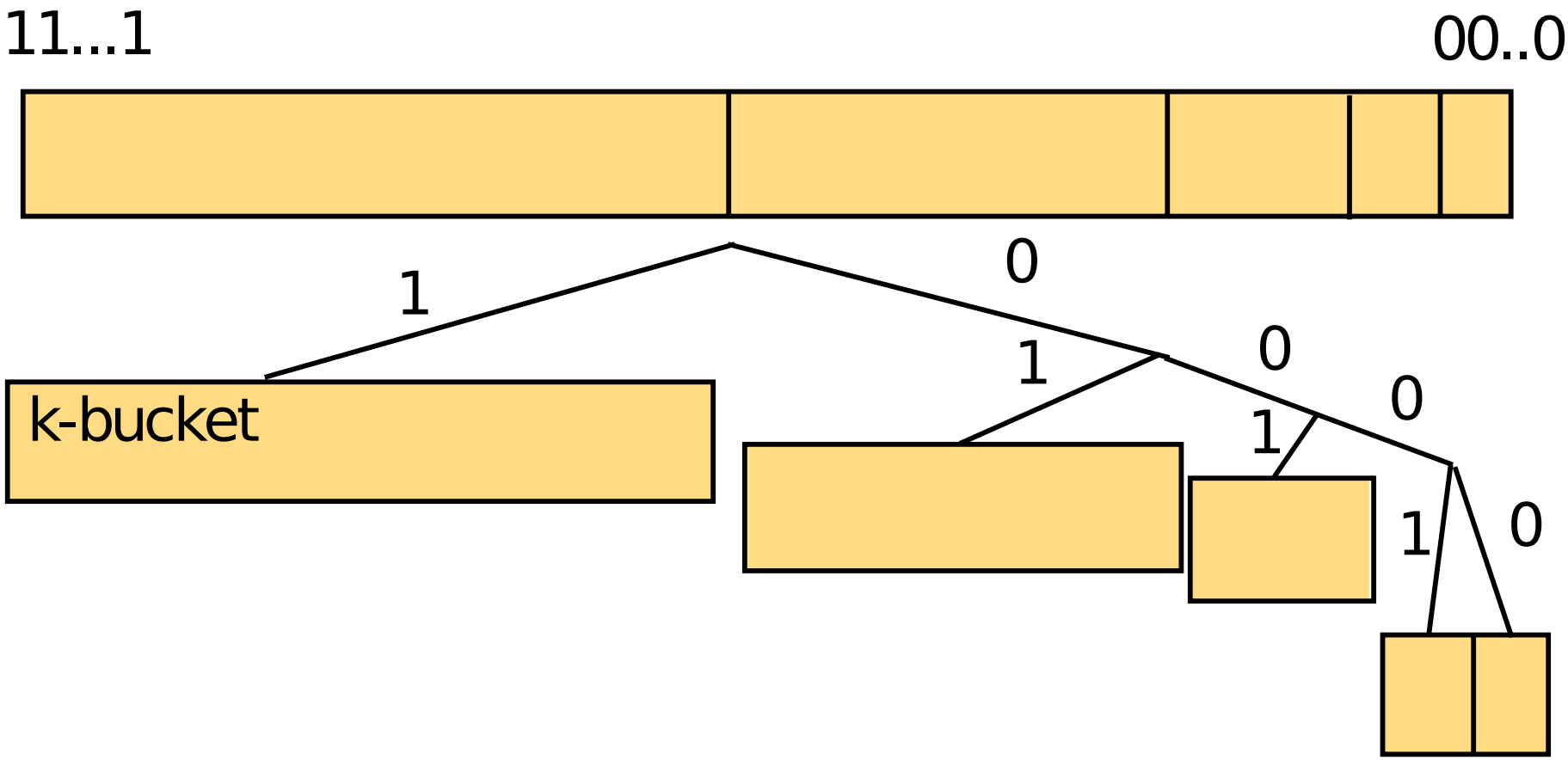
k-Buckets

- A k-bucket is a container for some subtree
- Contains at most k contacts (default: $k = 20$)

Routing table

- Stores the set of all k-bucket containers

Kademlia: routing information



Kademlia: k-bucket

Structure of a k-bucket

- List of nodes with distance $d \in [2^i, 2^{i+1})$ for some i, $0 < i \leq 160$
- List element: (node ID, IP address, UDP port)
- Maximum number of elements in list: k (usually, $k = 20$)
- List entries sorted by “last-recently seen”
- Correct nodes are *never* removed from a list

Kademlia: k-bucket

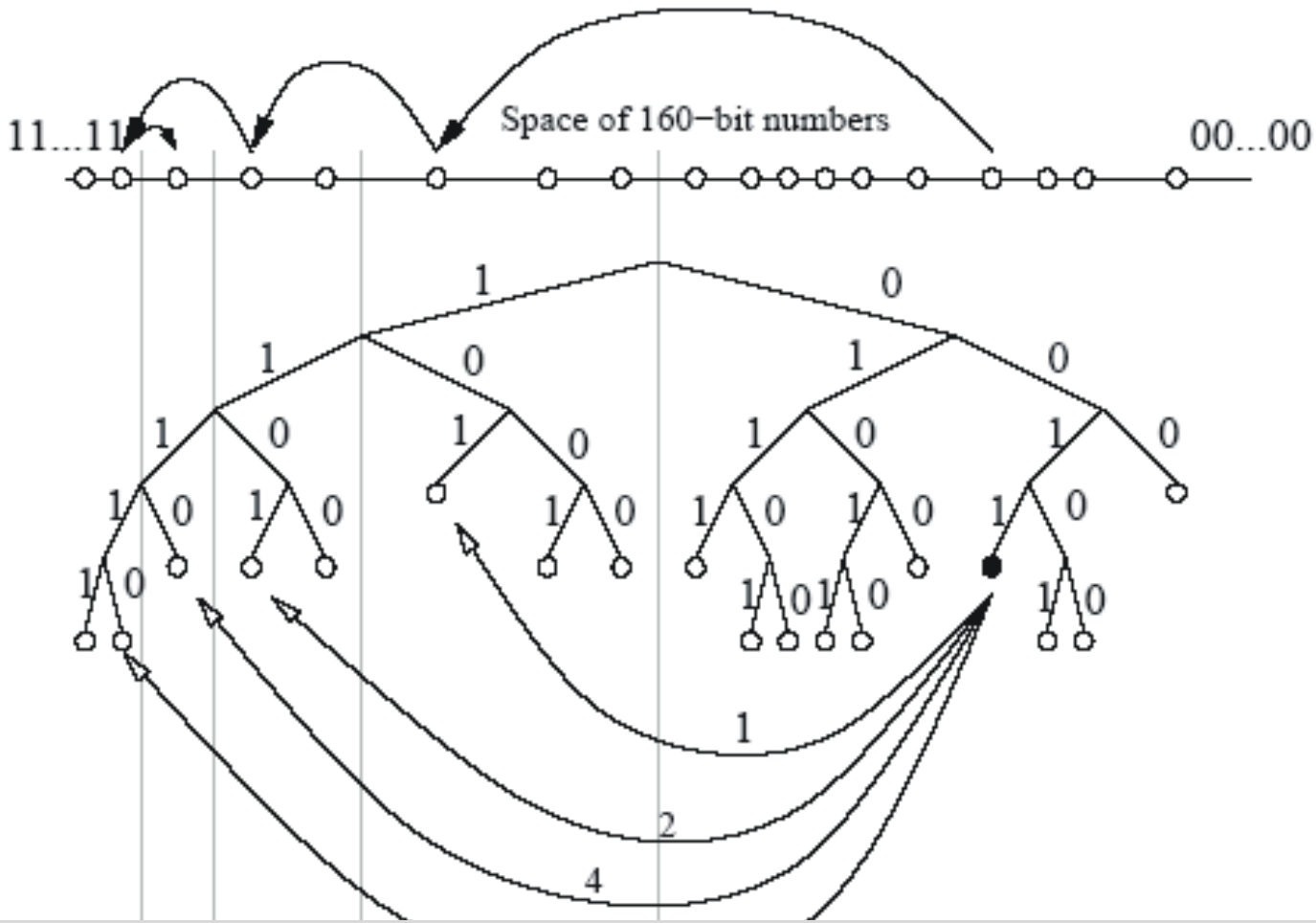
Updating contacts in a k-bucket

- Recently contacted nodes are at the end of the list, old contacts are at the beginning of the list
- If some message is received from some node:
  - If node is contained in some k-bucket: move it to the end of the list
  - If node is unknown and corresponding k-bucket has less than k elements: append it to the end of the list
  - If node is unknown and k-bucket is full, check if first node on list is alive. If not, remove dead node and append new node to the end of the list

Kademlia: searching

Iterated search

- Example: node with ID 0011 searches ID 1110



Kademlia: RPC-based protocol

Kademlia implements 4 RPCs:

- `FIND_NODE`: returns *k* closest contacts for some ID
- `FIND_VALUE`: same as FIND_NODE, except if node is responsible for ID, then it returns the stored data
- `STORE`: stores some key/value pair
- `PING`: checks if target node is alive

Kademlia: finding responsible nodes

For finding a set of *k* nodes responsible for some ID, Kademlia executes the following steps:

1. Initiator sends `FIND_NODE` to a set of α nodes that are close to the destination according to the XOR metric (e.g., $\alpha = 3$)
2. Recipient of `FIND_NODE` returns a set of *k* closest nodes
3. Initiator collects the replies, determines the best *k* contacts, and again sends `FIND_NODE` to a set of α new nodes. If some contact does not reply, a different node (from the set of *k* contacts) is contacted
4. The previous step is repeated until no new nodes are learned and all *k* closest nodes have been contacted

Kademlia: finding responsible nodes

Nodes:

- Initiator implicitly learns about new contacts (and can use them to update the routing date structures)
- The selection of α contact nodes can be made based on recorded round-trip delays

Kademlia: storing and retrieving data

Storing data (id, value):

- Finding *k* responsible nodes for *id*
- Sending `STORE` message for (id,value) to these *k* nodes

Retrieving data for id:

- Interaction identical to finding responsible nodes
- If node that stores data for id is contacted, it returns the data instead of a list of *k* contact nodes
- Nodes may cache data

Summary: Kademlia vs Chord

Kademlia vs. Chord

- Tree structure vs ring structure
- Explicit vs. implicit management of routing data structures
- Iterative and parallel vs. recursive operation
- Both have mechanisms for optimizing network communication
- Both scale well in search time and routing data structure size ($O(\log n)$)

Summary

1. Overview of peer-to-peer systems
   - Definition
2. Unstructured peer-to-peer systems
   - Napster, Gnutella, ED2K
3. Structured peer-to-peer systems
   - Distributed hash tables (DHT)
   - Chord
   - Kademlia