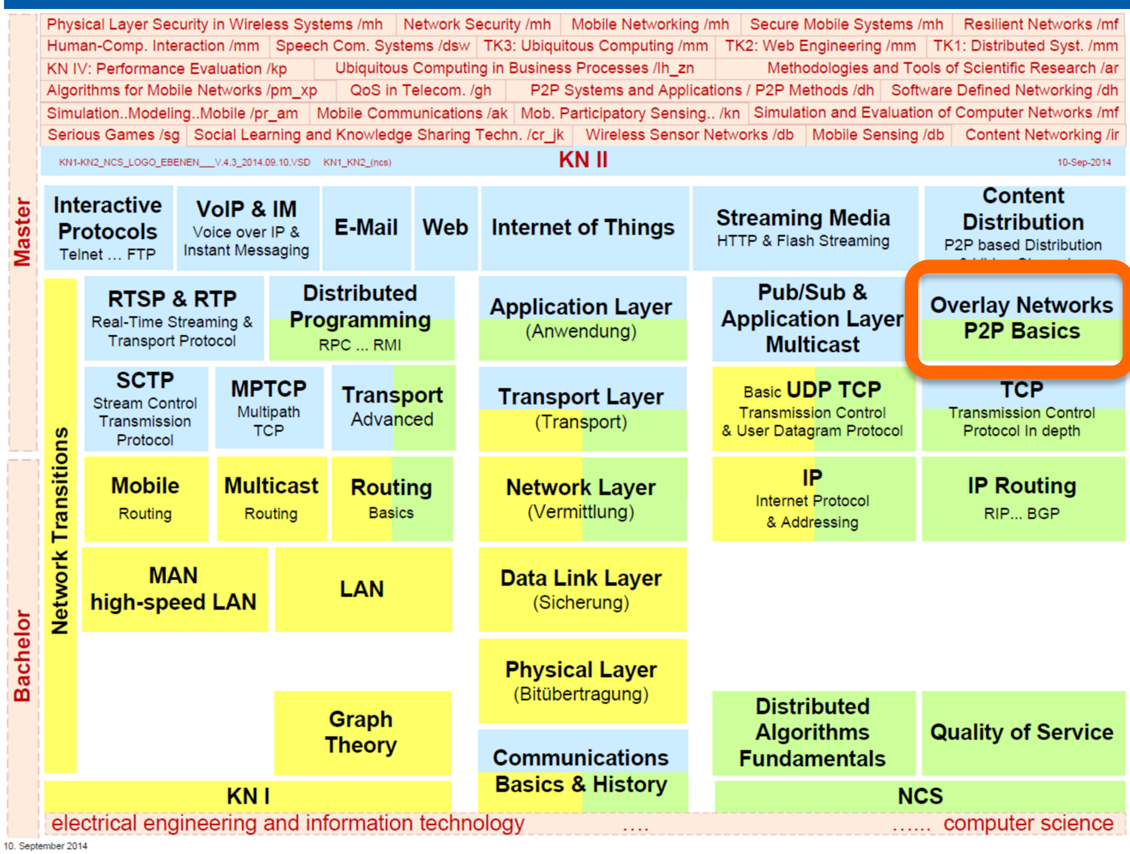
Communication Networks 2

Overlay Networks



Prof. Dr. Thorsten Strufe
Prof. Dr. Max Mühlhäuser

Dr.-Ing. **Doreen Böhnstedt**
Prof. Dr.-Ing. **Ralf Steinmetz**
KOM - Multimedia Communications Lab

Overview

TECHNISCHE UNIVERSITÄT DARMSTADT

A huge number of nodes participating in the network

- Have resources to share
- Have demands towards the use of resources which may not be satisfied easily and by single nodes

Main questions:

- Which of the nodes provides the resources
- Which instance of a resource shall be provided (exactly)?

Solution: Peer-to-Peer (P2P)

- Offer mechanisms to find / look up resources in a distributed manner
- Build overlay network(s) for direct interaction between peers

Modes of operation

- First, locate the node providing the desired service
- Second, interact directly with that node

Definition of Peer-to-Peer

TECHNISCHE UNIVERSITÄT DARMSTADT

P2P (Peer-to-Peer):

- A distributed system and
- A communications paradigm

Definition of Distributed Systems (very general)

"A collection of individual computing devices that can communicate & interact directly with each other"

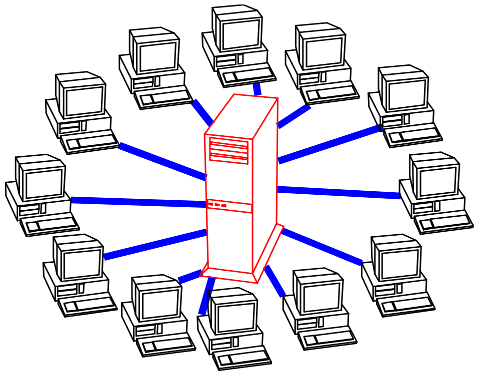
Main principles of a P2P system:

- Systems with loosely coupled (no fixed relationship) autonomous devices
- Devices have their own "semi-independent" agenda
  - Comply to some general rules
  - But local policies define their behavior
- (At least) limited coordination and cooperation needed
- Strategies to find peer providing desired content

Peer-to-Peer: 9 Features

1. **Relevant resources located at nodes (“peers”) at the edges of a network**
2. **Peers share their resources**
3. **Resource locations**
   - Widely distributed
   - Most often largely replicated
4. **Variable connectivity is the norm**
5. **Combined Client and Server functionality (“Servent”)**
6. **Direct interaction (provision of services, e.g. file transfer) between peers (= “peer-to-peer”)**
7. **Peers have significant autonomy and mostly similar rights**
8. **No central control or centralized usage/provisioning of a service**
9. **Self-organizing system**

Client / Server Model vs. P2P Technology



Main features of a Client / Server model:

- In principle, 1 server and N clients
- Clear distinction between server and client functionalities
- Simple search mechanism

Issue:

- Which server has the required information or file?

Solution:

- Look it up on one server after another
  (or on Google, which does this for you)

Advantages:

- Reliable, well known behavior

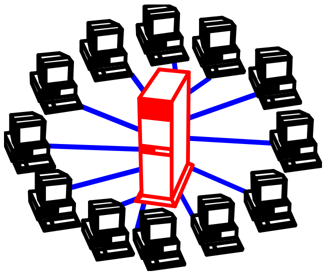
Drawbacks:

- Server needs to provide (almost) all resources

Client / Server model is not P2P:

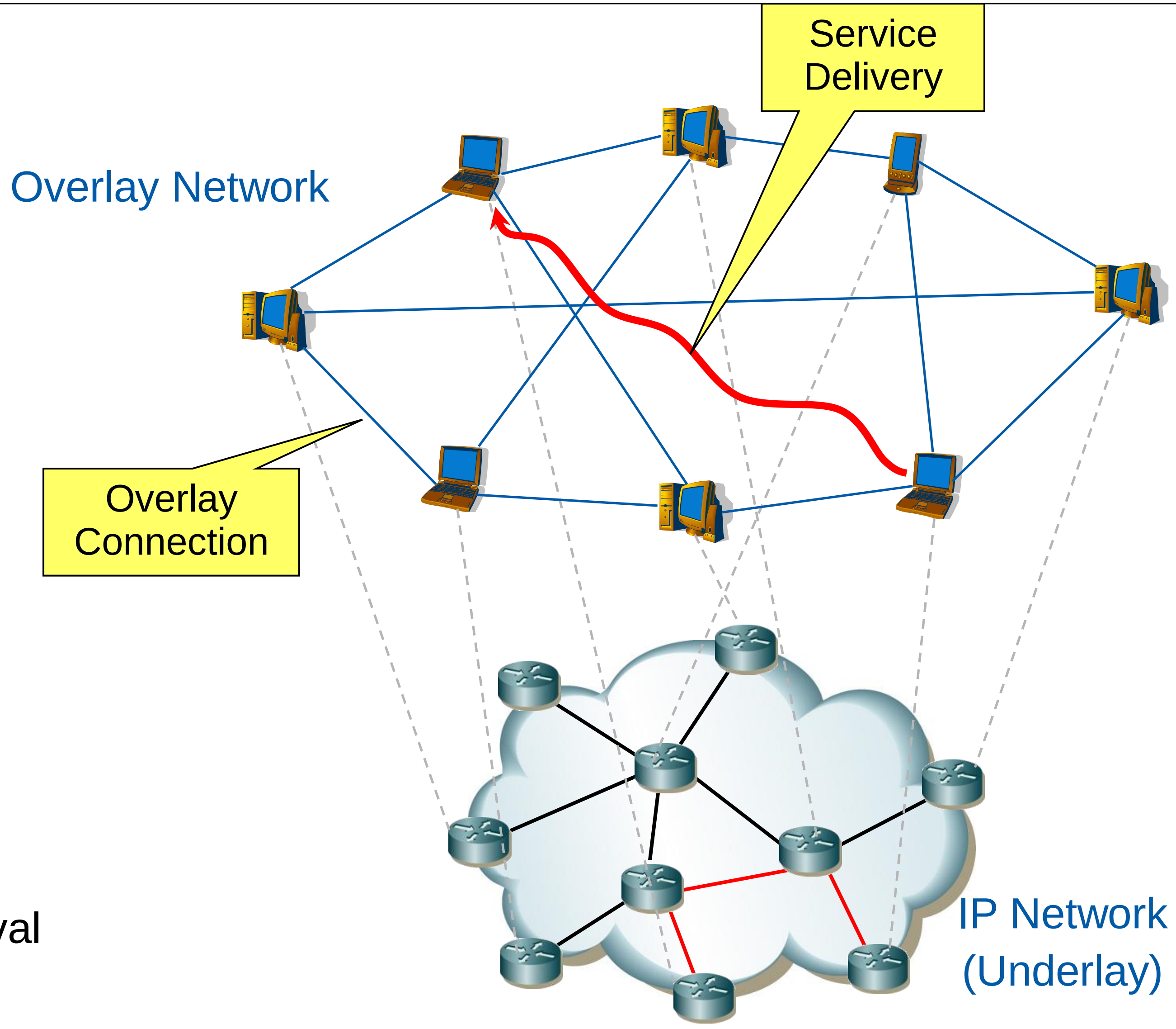
- Communication only between clients and server,
  not between clients and clients

Client / Server Model vs. P2P Technology

Client-Server	*Peer-to-Peer*
1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system 3. Clients as the lower performance system Example: WWW 	1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept)

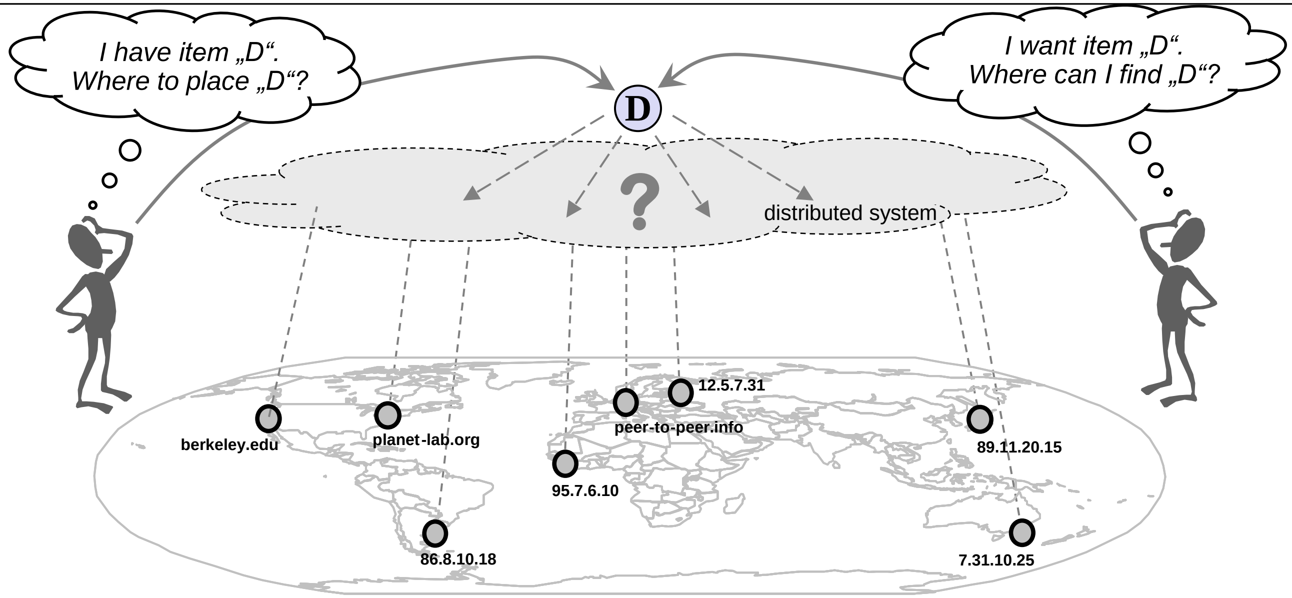
from R.Schollmeier and J.Eberspächer, TU München

2 Fundamentals of Overlay Networks



Peers provide:

- Communications
- Processing
- Storage & Retrieval
- Interaction

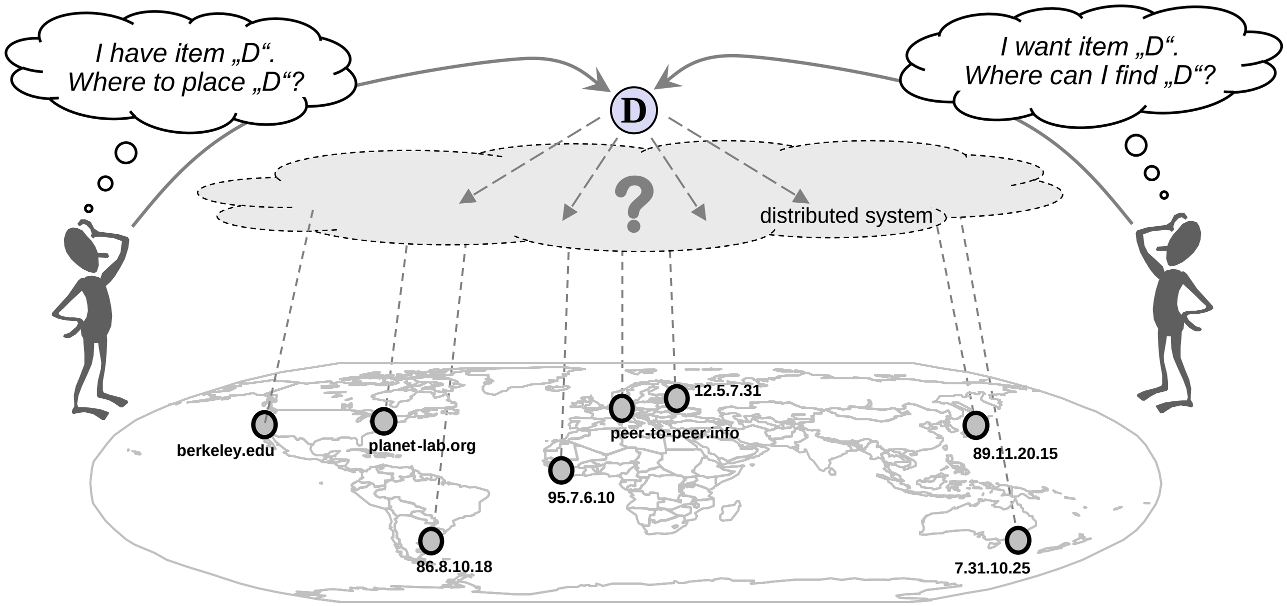


Essential challenges in (most) Peer-to-Peer systems

- Location of a data item at distributed systems
  - Where should the item be stored by the provider?
  - How does a requester find the actual location of an item?
- Scalability
  - To keep the complexity for communication and storage “scalable”
- Robustness and resilience
  - In case of faults and frequent changes

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. By Steinmetz, Wehrle

Strategies for Data Retrieval

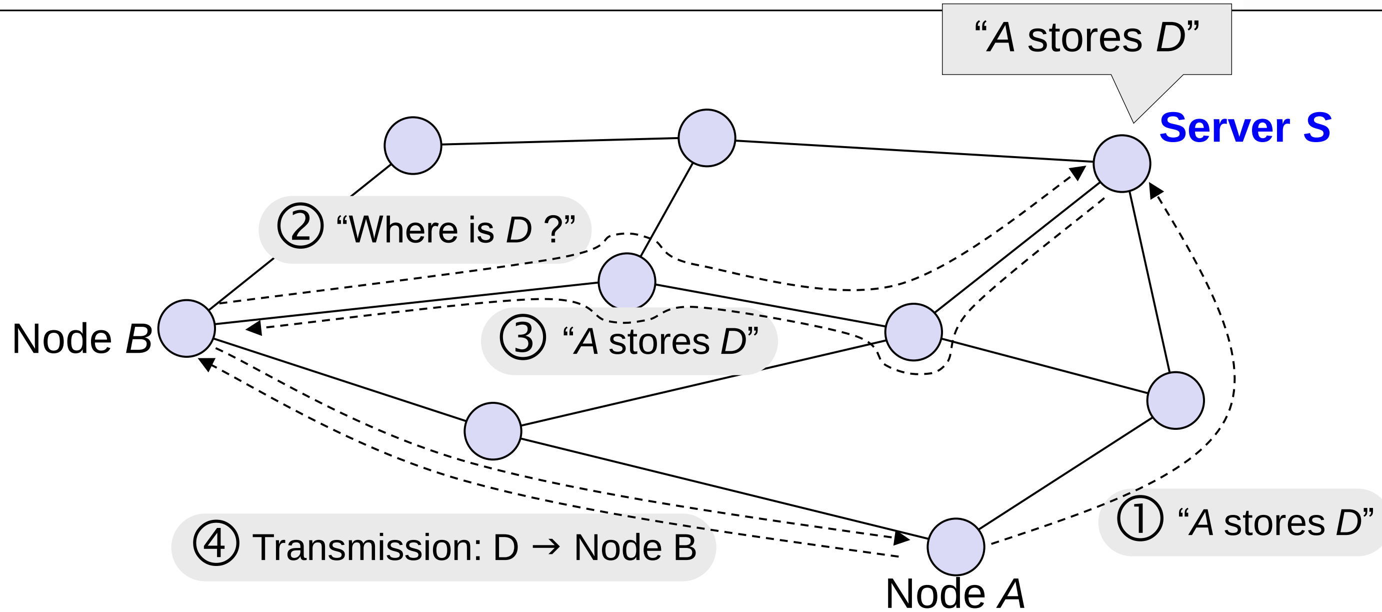


Strategies to store and retrieve data items in distributed systems

- Central server (central indexing)
- Flooding search (local indexing)
- Routing (distributed indexing)

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. By Steinmetz, Wehrle

Strategies for Data Retrieval Approach I: Central Server



Simple strategy:

- Central server stores information about locations
  - ① Node A (provider) tells server that it stores item D
  - ② Node B (requester) asks server S for location of D
  - ③ Server S tells B that node A stores item D
  - ④ Node B requests item D from node A

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. By Steinmetz, Wehrle

Approach I: Central Server

Advantages

- Search complexity of O(1) – "just ask the server"
- Complex and fuzzy queries are possible
- Simple and fast

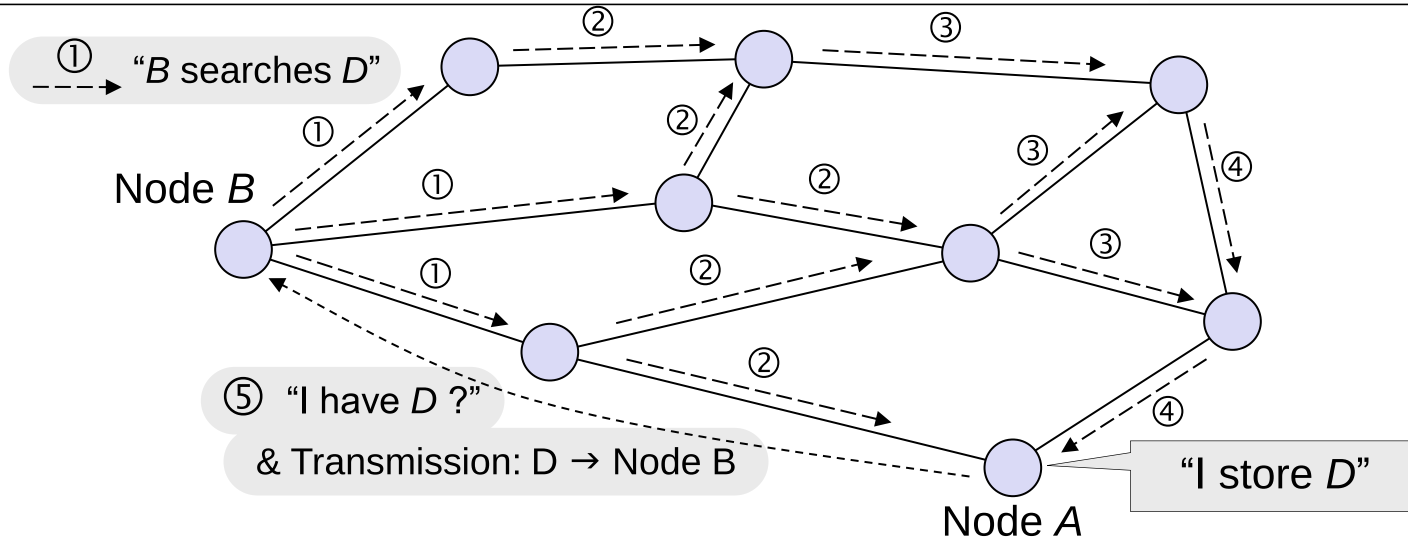
Challenges

- No intrinsic scalability
  - O(N) network and system load on server
- Single point of failure or attack
- Non-linear increasing implementation and maintenance cost
  - In particular for achieving high availability and scalability
- Central server not suitable for systems with massive numbers of users

But overall, …

- Best principle for small and simple applications

Approach II: Flooding Search



Distributed Indexing Approach

- No information about location of data at intermediate systems
- Necessity for broad search
  - ① Node B (requester) asks neighboring nodes for item D
  - ②-④ Nodes forward request to further nodes (breadth-first search / flooding)
  - ⑤ Node A (provider of item D) sends D to requesting node B

The content of this slide has been adapted from "Peer-to-Peer Systems and Applications", ed. By Steinmetz, Wehrle

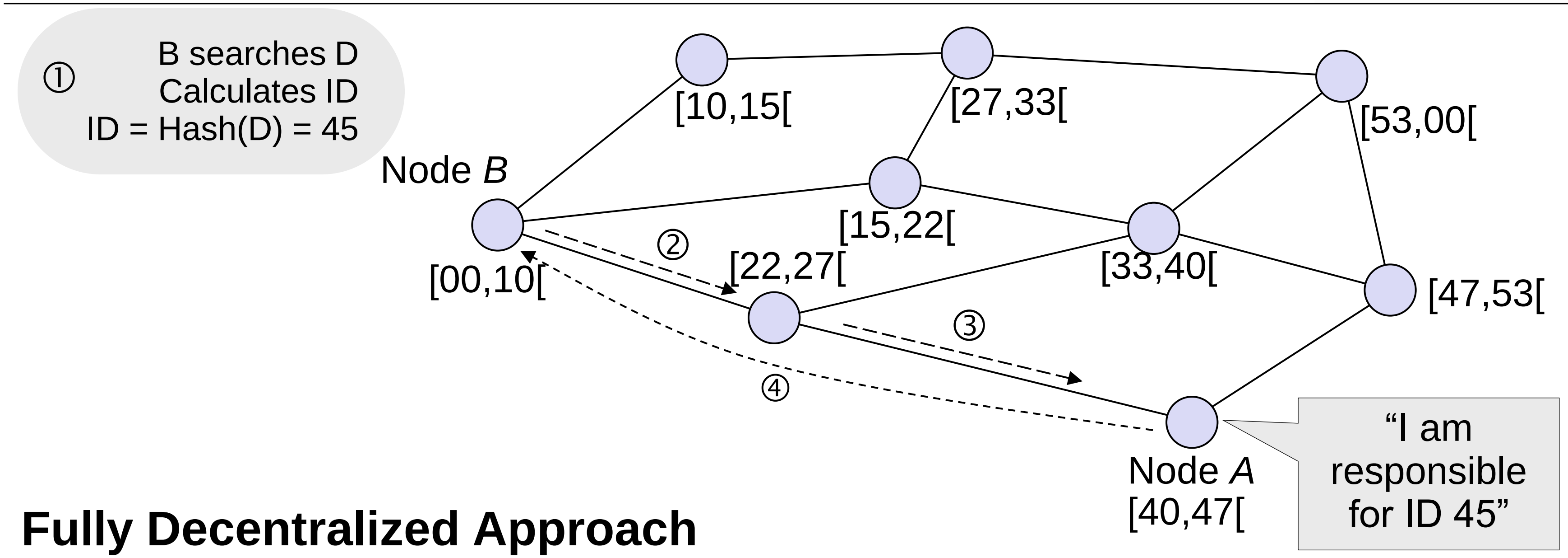
Fully Distributed Approach

- No global information available about location of a item
- Content only stored at respective node providing it

Retrieval of data

- No routing information for content
- Necessity to ask as many systems as possible / necessary
- Approach
  - Flooding: high traffic load on network, does not scale
  - Highest effort for searching
    - Quick search through large areas
    - Many messages needed for unique identification

Strategies for Data Retrieval Approach III: Routing (Distributed Indexing)



Fully Decentralized Approach

- Construction of an overlay
- Each node responsible for certain range of the ID space
- IDs are calculated using hash functions (e.g. SHA1, MD5)
- Routing is done as follows
  - ① Node B (requester) calculates ID of D
  - ②,③ Node B sends request to neighbor with responsibility interval closest to ID
  - ④ Node A is responsible for ID 45 and sends requested item D to node B

The content of this slide has been adapted from "Peer-to-Peer Systems and Applications", ed. By Steinmetz, Wehrle

Approach III: Routing (Distributed Indexing)

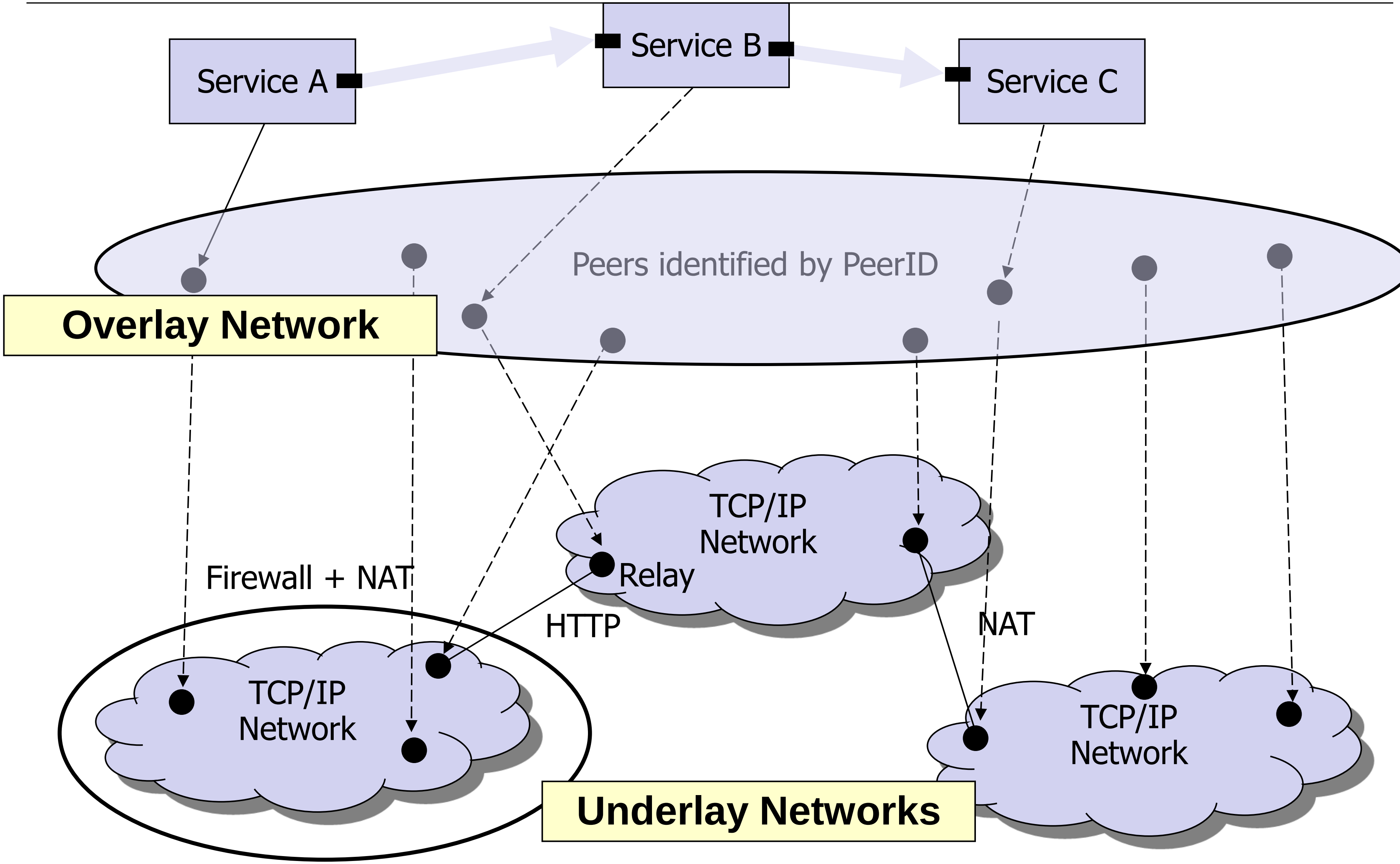
Advantages

- No single point of failure
- Scalable
- Efficient way for retrieving content

Challenges

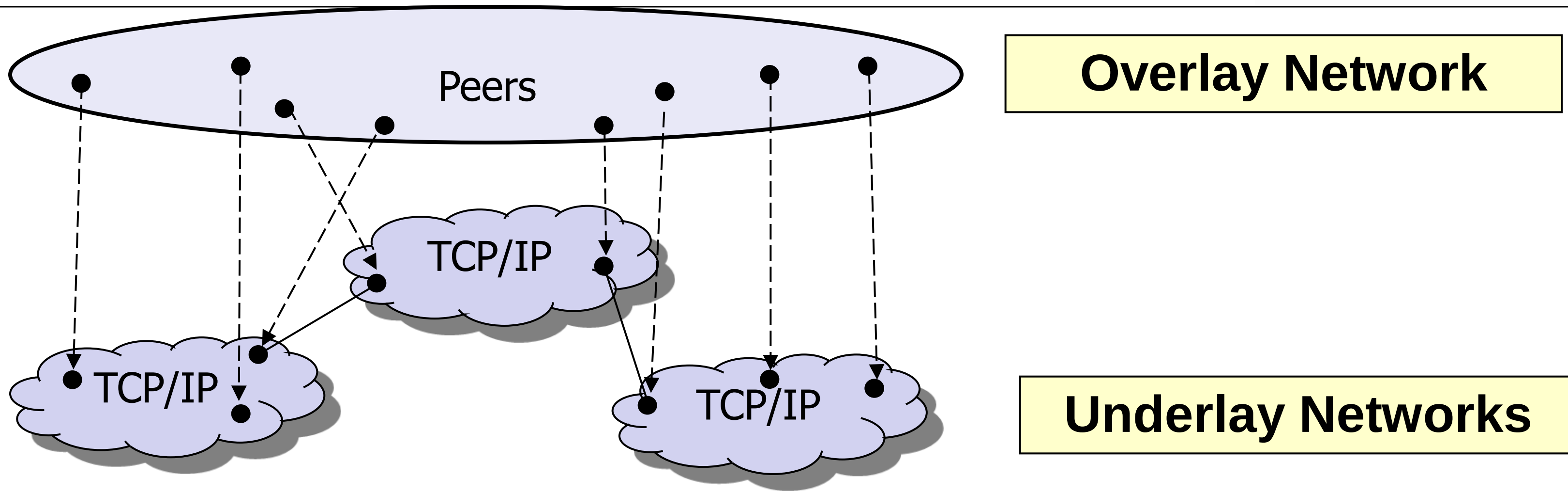
- Maintenance of ID Space necessary
- Reliable routing
- Load balancing
- Replication of data

2.2 Overlay Networks: Layer Model



Picture adapted from Traversat, et.al Project JXTA virtual network

Overlay Networks



A network

- Provides services (service model)
- Defines how nodes interact
- Is needed for addressing, routing, …

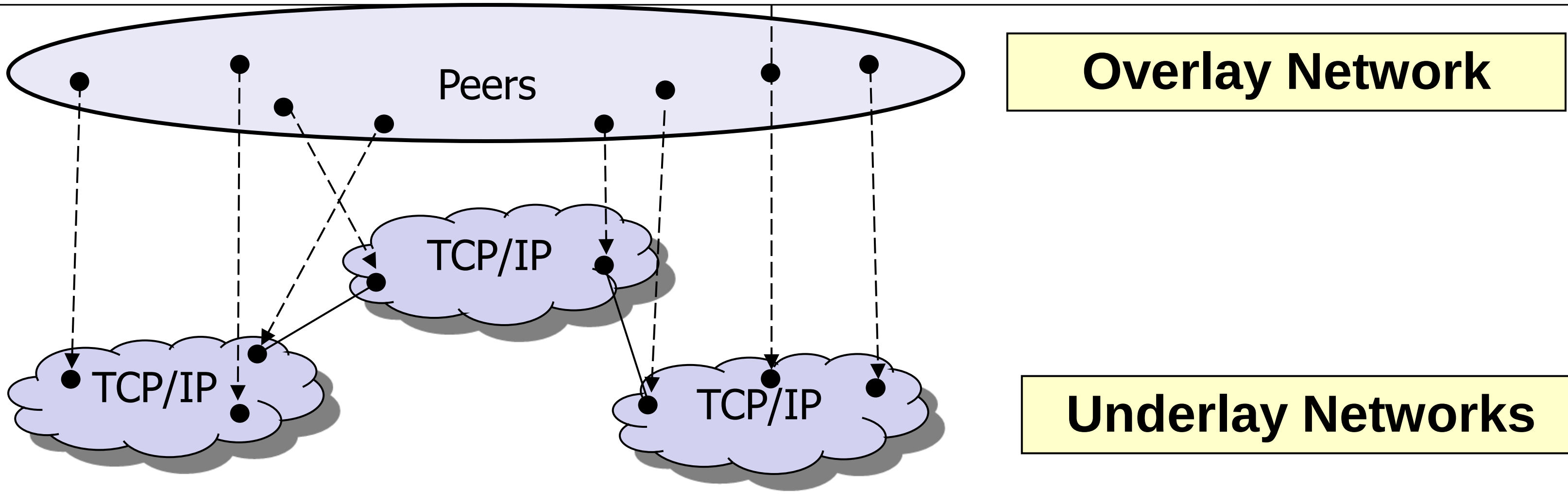
Overlay network

- = Network built ON TOP of one or more existing networks
- Adds an additional layer of
  - Abstraction
  - Indirection/virtualization

Picture adapted from Traversat, et.al Project JXTA virtual network

Overlay Networks

TECHNISCHE UNIVERSITÄT DARMSTADT



P2P networks form an overlay network

- On top of the Internet (IP network)

IP networks form an overlay network

- Politically and technically
- Over the underlying telecommunication infrastructure

Both

- Introduce their own addressing scheme: e.g. user names, IP addresses
- Emphasize fault-tolerance
- Are based on the end-to-end principle
  - i.e. provide as much intelligence as possible at end nodes

Picture adapted from Traversat, et.al Project JXTA virtual network

What kind of routing algorithms can be used in overlay networks?

Answer: Exactly the same as in IP networks :-)

When node A sends a message to its overlay neighbor B:

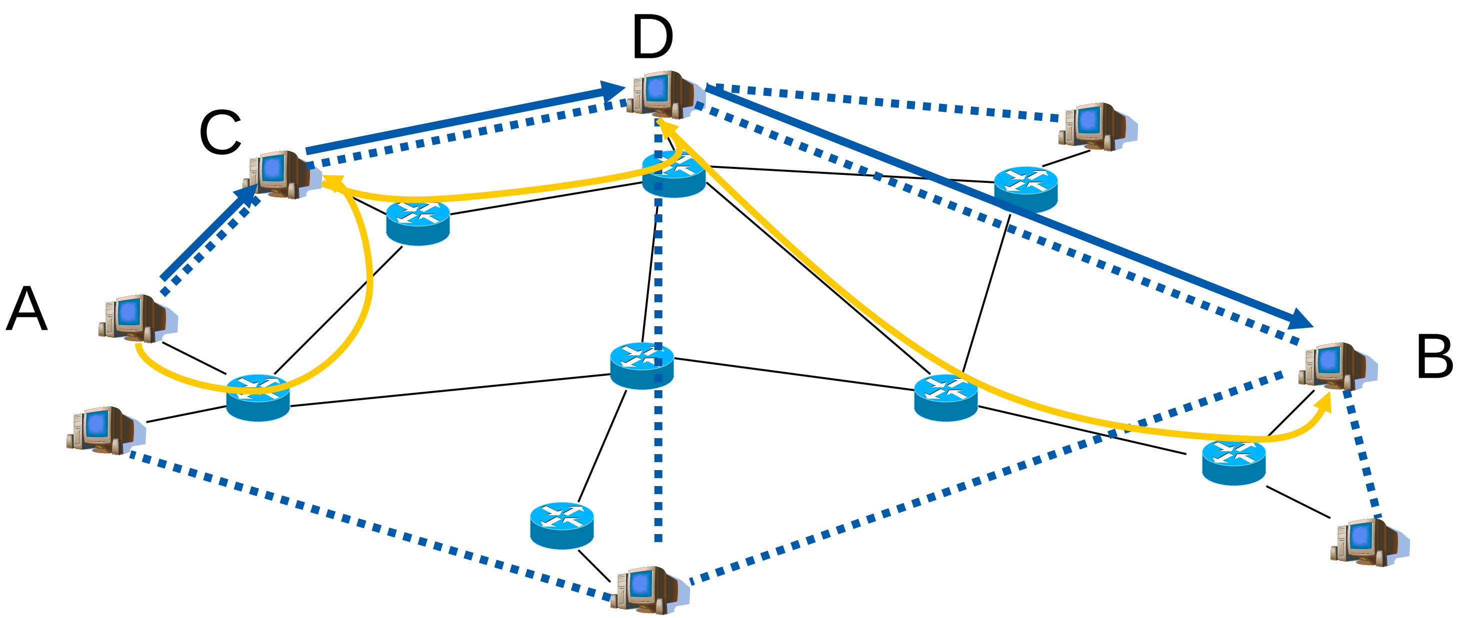
- The message gets routed over the Internet from A to B
  - Takes possibly several hops through Internet routers
- From overlay point of view, message takes only one hop

Two important metrics for overlay routing:

- How many hops in overlay?
  - Same as for traditional routing algorithms
  - First goal: Minimize hops in overlay
- How many hops in underlying network?
  - Routing in overlay from A to B to C is (likely) longer than standard Internet path from A to C (**stretch**)
  - Second goal: Minimize stretch (as much as possible)

A wants to send message to B

- Shortest path in overlay is A, C, D, B
  - A to C is 3 (IP) hops, C to D is 3 hops, D to B 4 hops, total 10 hops
  - Shortest direct IP path from A to B is 5 hops
  - Stretch factor is in this case 2



Overlay Networks: Advantages

New layer quickens search/lookup of requested information

- Additional layer solves this problem for higher layers

Do not have to

- Deploy new equipment
- Modify existing software/protocols

Allow for bootstrapping

- Make use of existing environment by adding new layer

Not all nodes have to participate in maintaining

- But free riding is still a problem

E.g.,

- Adding IP on top of Ethernet
  - Does not require modifying Ethernet protocol or driver

Overlay Networks: Disadvantages

Overhead

- Adds another layer in networking stack
- Additional packet headers, processing

Complexity

- Layering does not eliminate complexity, it only manages it
- More layers of functionality
  - More possible unintended interaction between layers
- Misleading behavior
  - E.g. corruption drops on wireless links interpreted as congestion drops by TCP

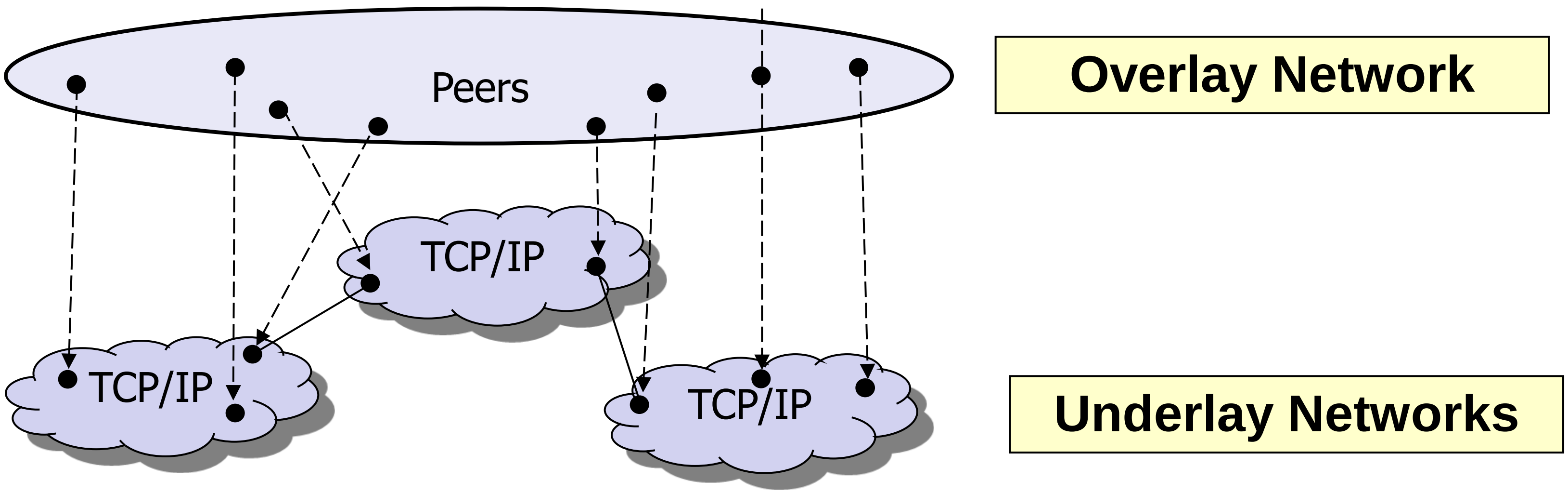
Redundancy

- Features may be available at various layers

Some restricted functionality

- Some features that a “lower layer” does not provide cannot be added on top
  E.g. no real-time capabilities (for QoS)

Overlay Networks: Others

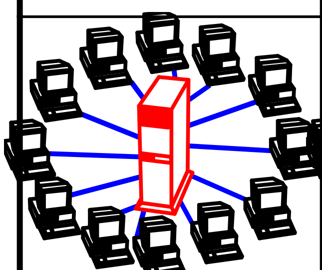
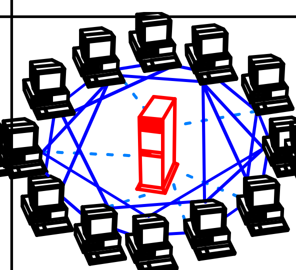
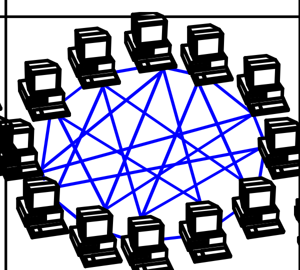
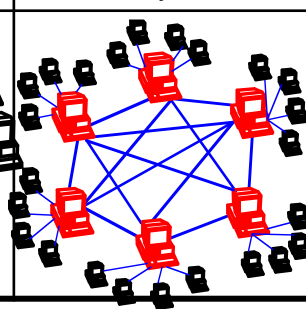
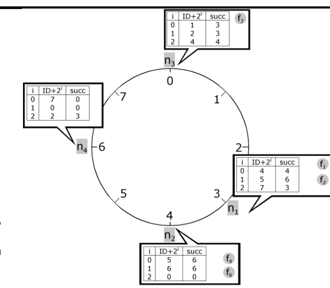
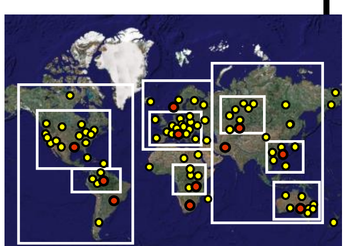


Other (non P2P) overlay networks

- VPNs (virtual private networks)
- IP over ad hoc networks
- Multicast

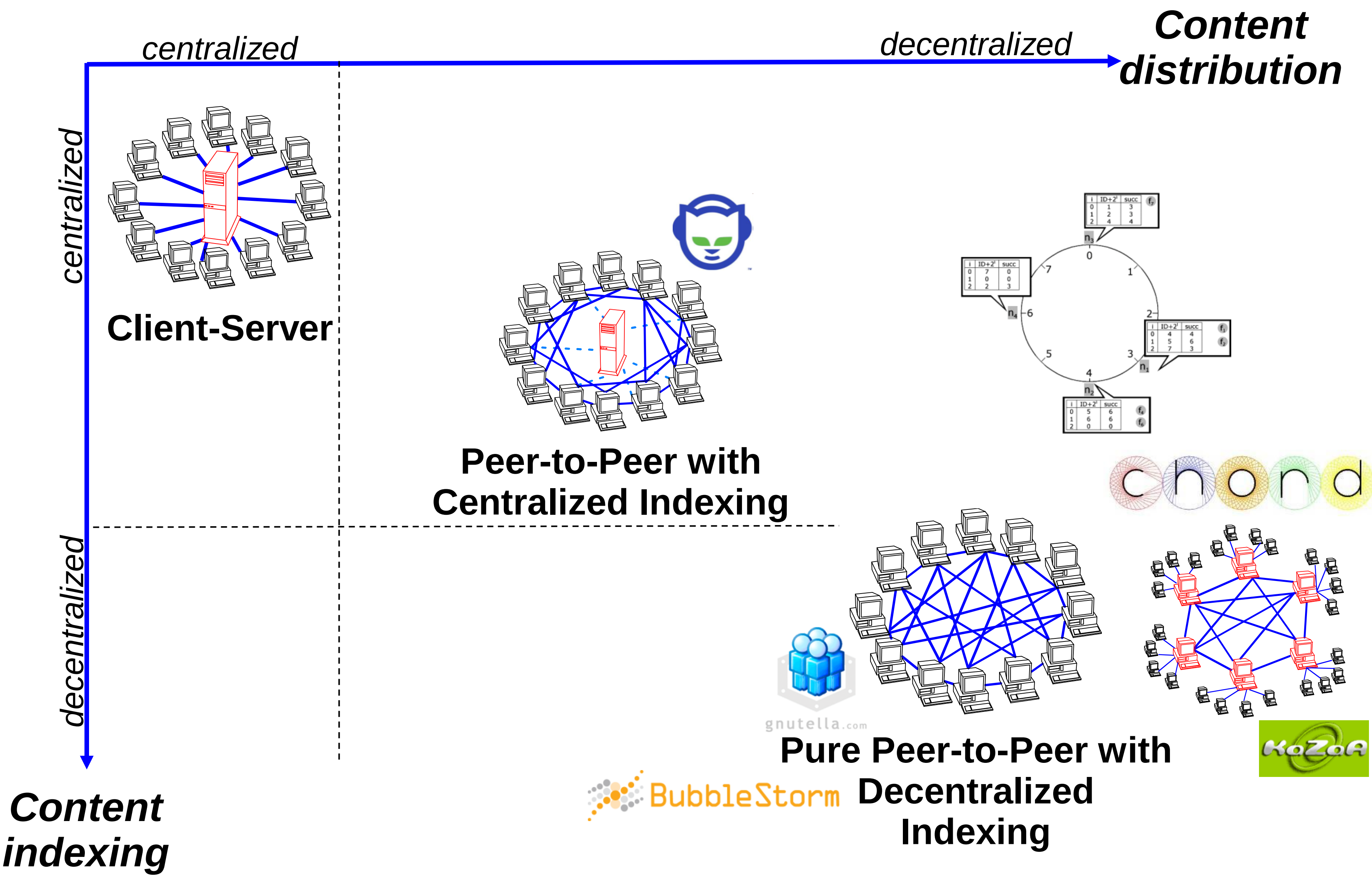
Picture adapted from Traversat, et.al Project JXTA virtual network

2.3 Overlay Networks: Structures

Client-Server	***Peer-to-Peer***				
1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system. 3. Clients as the lower performance system Example: WWW	1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept)				
	Unstructured P2P			***Structured P2P***	
	Centralized P2P	*Pure P2P*	*Hybrid P2P*	*DHT-Based*	*Hybrid P2P*
	1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Example: Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → no central entities Examples: Gnutella 0.4, Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Examples: Gnutella 0.6, Fasttrack, edonkey	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are “fixed” 5. Distributed indexing (content is not relocated) Examples: Chord, CAN	1. All features of Peer-to-Peer included 2. Peers are organized in a hierarchical manner 3. Any terminal entity can be removed without loss of functionality Examples: RecNet Globase.KOM
					

Compare to R.Schollmeier and J.Eberspächer, TU München

Peer-to-Peer Architectures



Structured and Unstructured P2P Networks

Unstructured P2P Networks

- Objects have no special identifier
- Location of desired object a priori not known
- Each peer is only responsible for objects it submitted

Structured P2P Networks

- Peers and objects have identifiers
- Objects are stored on peers according to their ID:
  responsibleFor(ObjID) = PeerID
- Distributed indexing points to object location

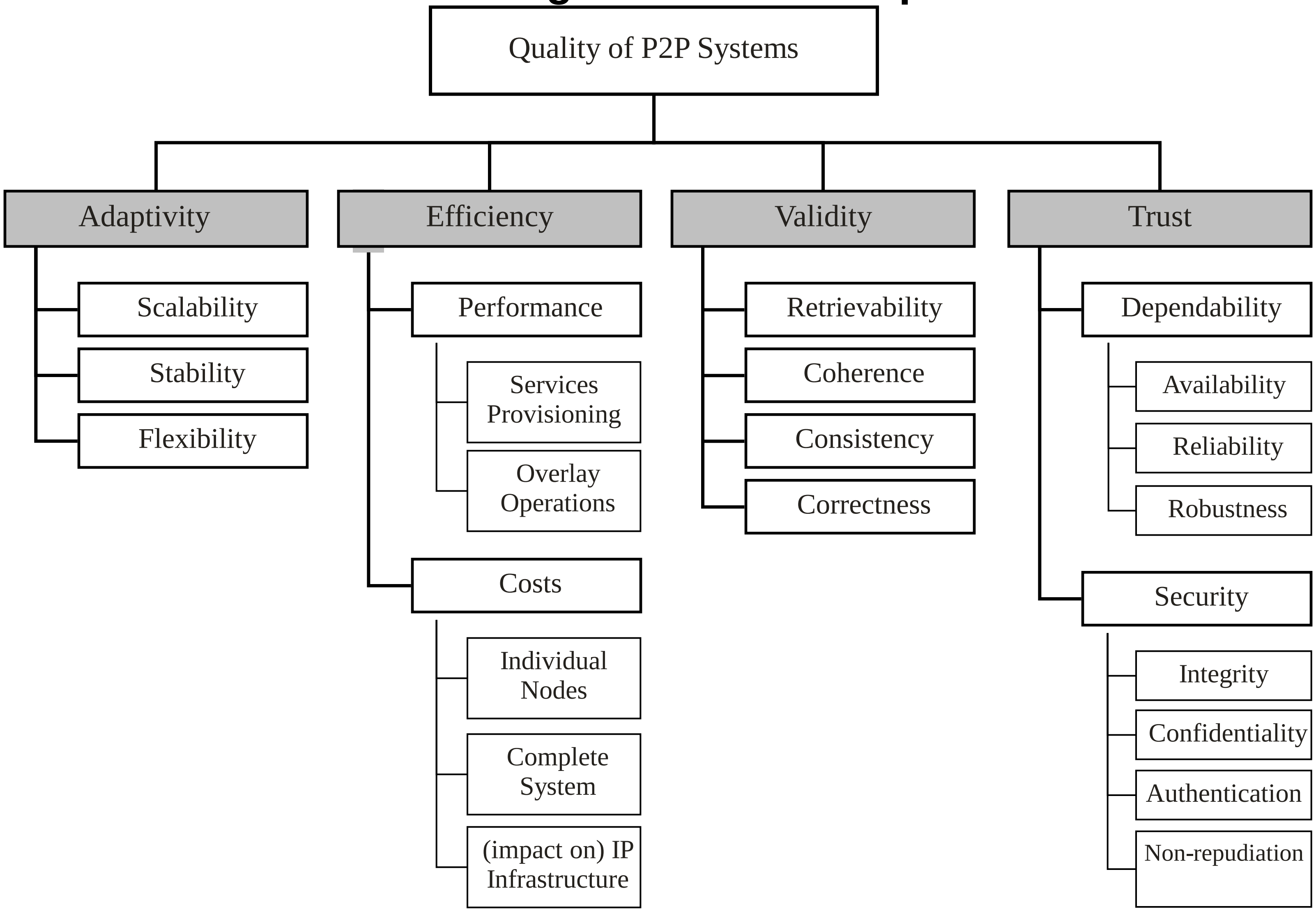
Search:

Find all (or some) objects in the P2P network which fit the given criteria

Lookup / Addressing:

Retrieve the object which is identified with a given identifier

To handle trade-offs among several P2P requirements



Requirements for Overlay Networks

Efficiency

- Ratio of
  - Performance to
  - Required effort

$$\text{Efficiency} = \frac{\text{Performance}}{\text{Required Effort}}$$

Scalability (expendability, enhancements)

- Ease with which the system may adapt itself to larger sizes
  - e.g. with respect to the amount of
    - Nodes
    - Shared resources

Adaptability

- Ease with which a system or component can be modified to fit the context

Stability

- Preserving the/an overlay structure when network changes
  - e.g.
    - Nodes join/leave
    - Network grows

Requirements for Overlay Networks

Fault-tolerance

- Resilience of the connectivity when failures are encountered
  - By arbitrary leave of peers

Heterogeneity

- Considering variations in physical capabilities and peer behavior (e.g. file fishing)

Fairness

- Evenly distributing workload across nodes

Security

- Ability of a system to manage, protect and distribute sensitive information

Privacy

- Degree to which a system or component allows for (or supports) anonymous transactions

Requirements for Overlay Networks: Trade-offs

Time – Space

- e.g. local information vs. complete replication of indices

Security – Privacy

- e.g. fully logged operations vs. totally untraceable

Efficiency – Completeness

- e.g. exact key-based matching vs. partial matching (use of wildcards)

Scope – Network load

- With TTL (time to live)
- e.g. TTL based requests vs. exhaustive search

Efficiency – Autonomy

- e.g. hierarchical vs. pure P2P overlays

Reliability – Low maintenance overhead

- e.g. deterministic vs. probabilistic operations

Metrics, Searching and Addressing

Probability of success

- Structured
  - Protocols guarantee results, if target exists
    - Assuming absence of malicious peers
- Unstructured
  - Protocols require exhaustive search

Protocol metrics

- Average number of messages per node
- Visited nodes
- Peak number of messages
- Congestion

Quality of results

- Completeness (are all results returned)
- Correctness (are all returned entries valid)
- Operation latency (time needed to solve the query)

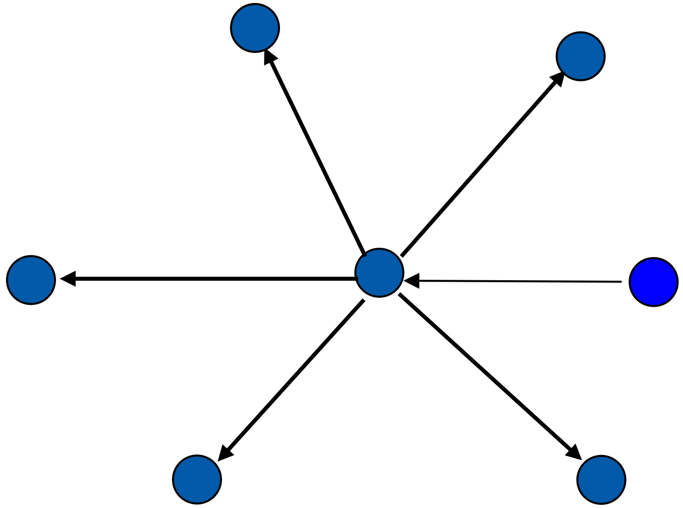
Search Mechanisms in P2P Overlays

- A. Broadcast
- B. Expanding Ring
- C. Random Walk
- D. Rendezvous Idea

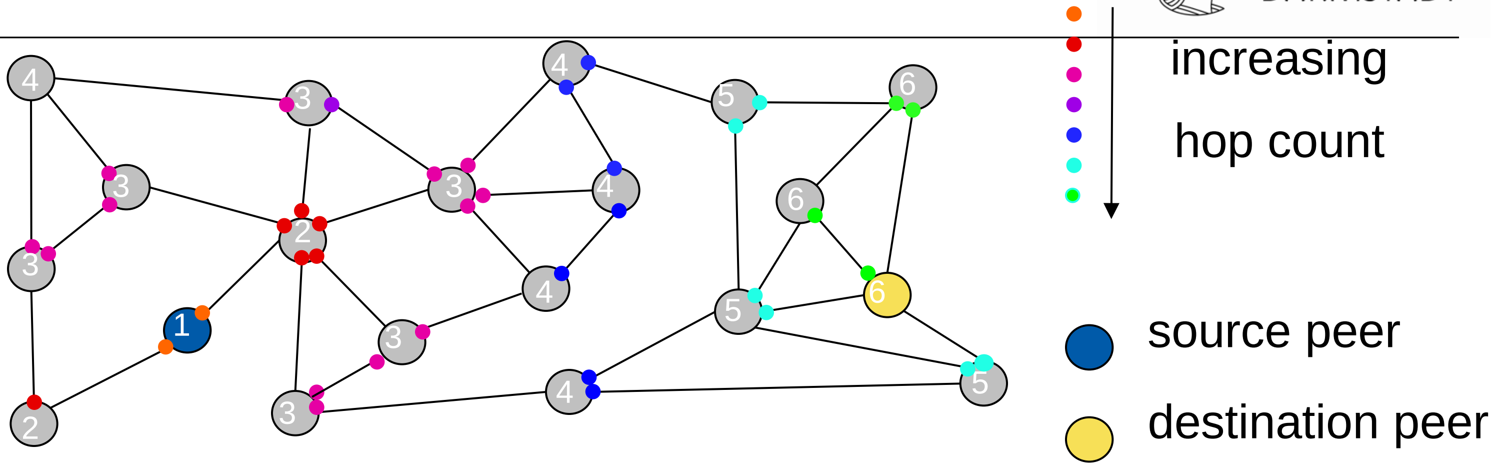
A. Search Mechanisms: Broadcast

Breadth-first search (BFS)

- Use system-wide maximum TTL to control communication overhead
- Send a message to all neighbors except the one who delivered the incoming message
- Store message identifiers of routed messages or use non-oblivious messages to avoid retransmission cycles



Example



Overhead

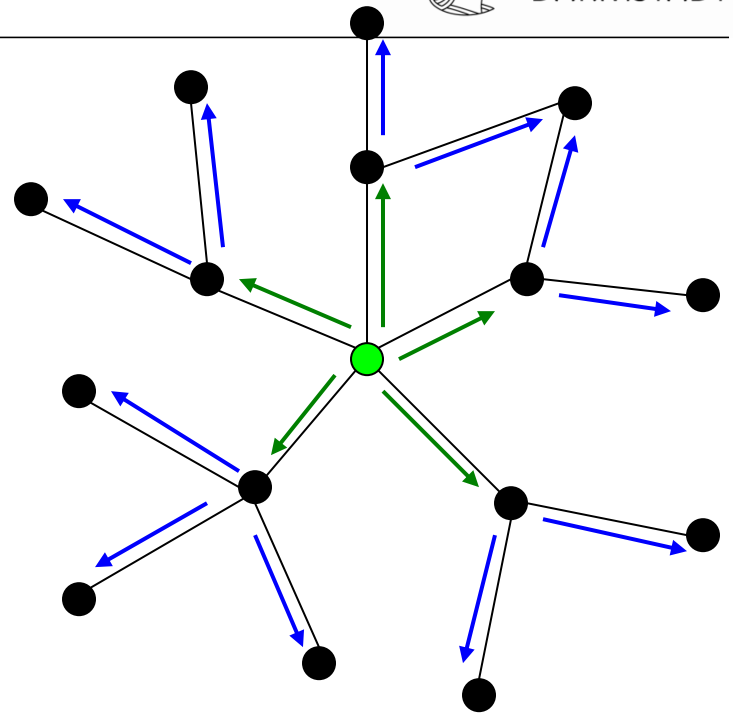
- Large, here 43 messages sent

Length of the path:

- 5 hops

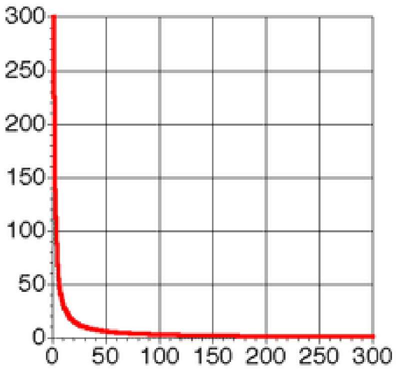
B. Search Mechanisms: Expanding Ring

Mechanism
- Successive floods with increasing TTL
  - Start with small TTL
  - If no success increase TTL
  - .. etc.



Properties
- Improved performance
  - If objects follow Zipf law popularity distribution and are located accordingly
- Message overhead is high

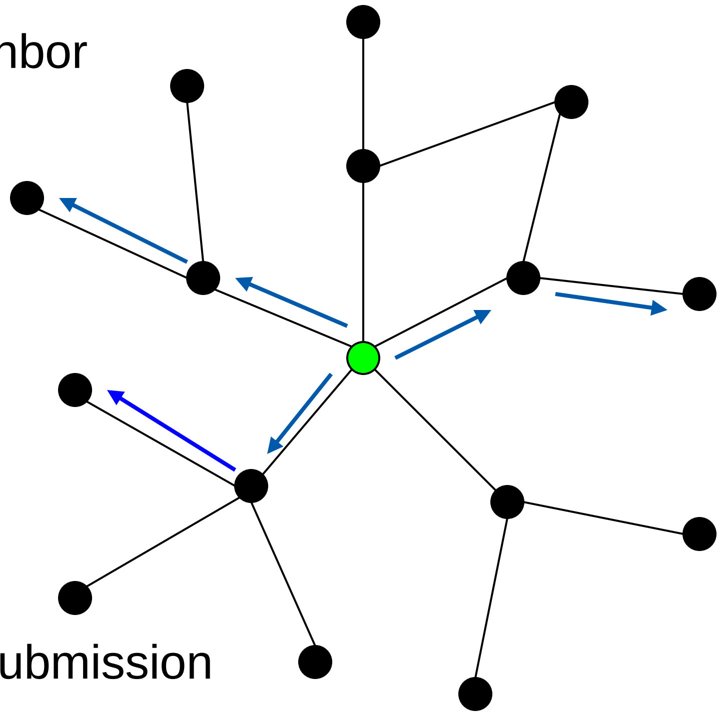
Zipf-law example



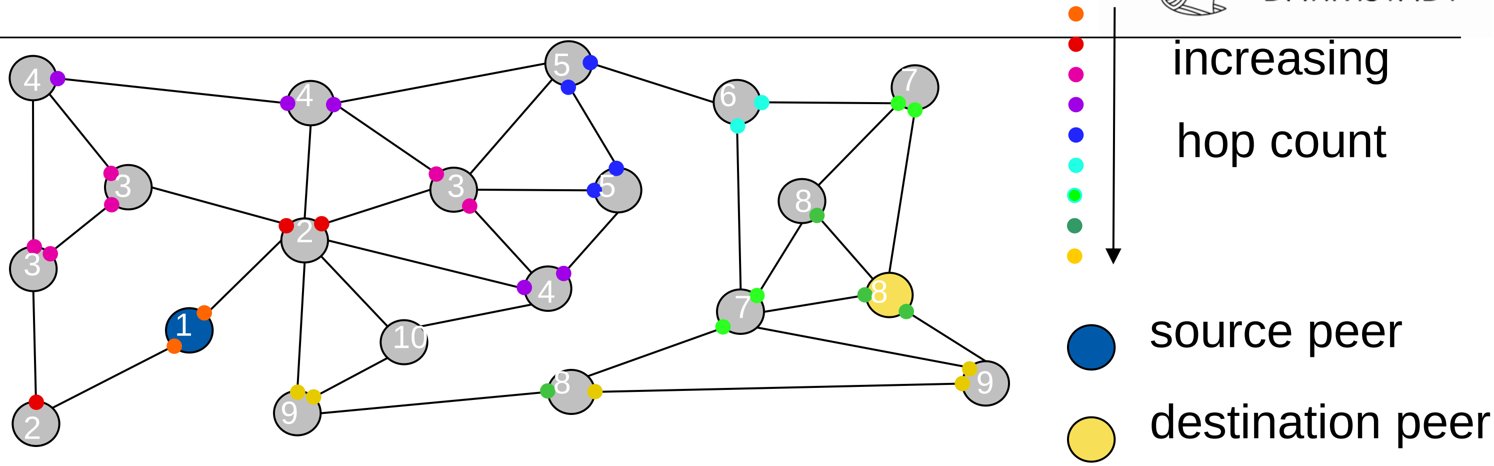
TECHNISCHE UNIVERSITÄT DARMSTADT

Random walks

- Forward the query to a randomly selected neighbor
  - Message overhead is reduced significantly
  - Increased latency
- Multiple random walks (k-query messages)
  - Reduces latency
  - Generates more load
- Termination mechanism
  - TTL-based
  - Periodically checking requester before next submission



Example



Random walk with n=2

- (each incoming message is sent twice out)

Overhead

- Smaller, here e.g. 30 messages sent until destination is reached

Length of the path found

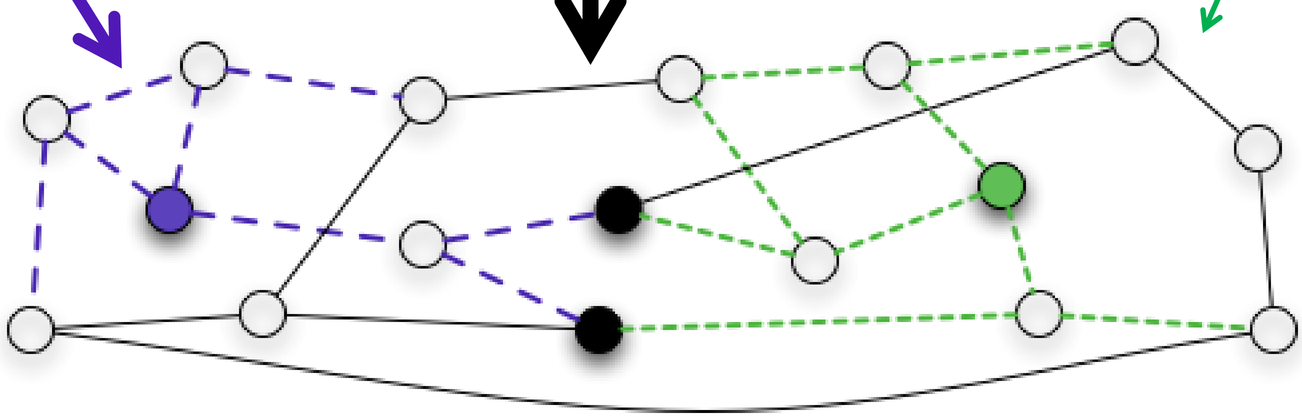
- e.g.
  - 7 hops

D. Search Mechanisms: Rendezvous Point

Storing node (green/light grey on right side) propagate content on all nodes within a predefined range

Requesting node (blue/dark grey on left side) propagates his query to all neighbors within a predefined range

A query hit can be found at the Rendezvous Point (black)



Principle of unstructured overlay networks

- Location of resource only known to submitter
- Objects have no special identifier (hence, unstructured)
- Each peer is responsible only for the objects it submitted
- Introduction of new resource at any location

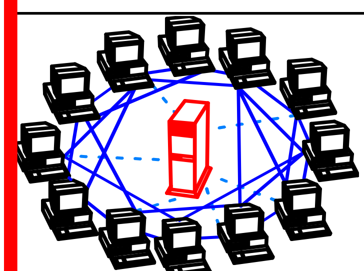
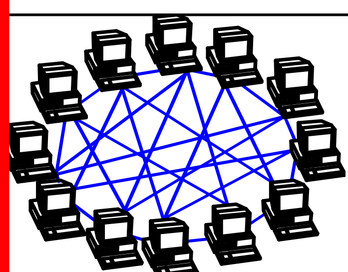
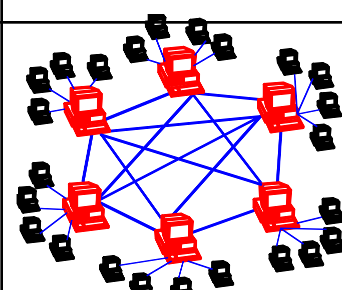
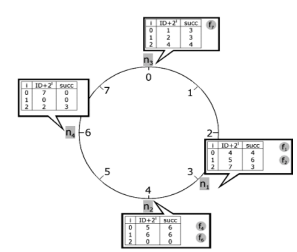
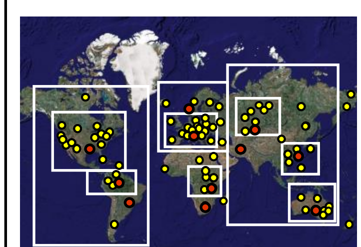
Comments

- Known difficulties like broadcasting & flooding
  - Network load, scalability
- Excellent if high robustness needed, but simple

Main task:

- To find all peers storing objects fitting some criteria
- To communicate P2P having identified these peers

3.1 Unstructured Centralized P2P Systems

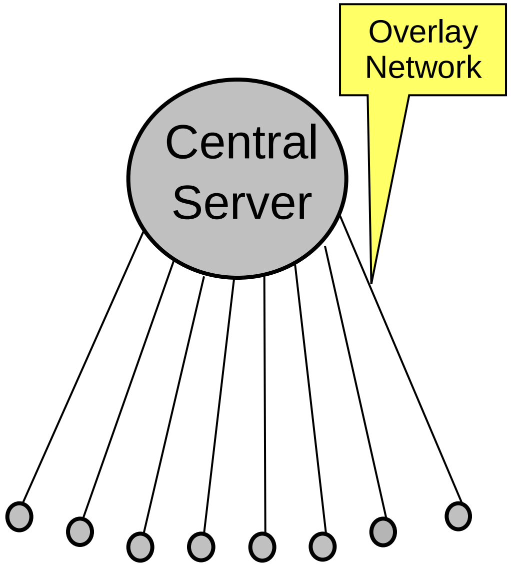
Unstructured P2P			*Structured P2P*	
Centralized P2P	***Pure P2P***	***Hybrid P2P***	***DHT-Based***	***Hybrid P2P***
1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database	**1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → no central entities**	**1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities**	**1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed"**	**1. All features of Peer-to-Peer included 2. Peers are organized in a hierarchical manner 3. Any terminal entity can be removed without loss of functionality**
Examples: ▪ Napster	**Examples: ▪ Gnutella 0.4 ▪ Freenet**	**Examples: ▪ Gnutella 0.6 ▪ Fasttrack ▪ eDonkey**	**Examples: ▪ Chord ▪ CAN ▪ Kademlia**	**Examples: • RecNet • Globase.KOM**
				

from R.Schollmeier and J.Eberspächer, TU München

Centralized P2P Networks

Central index server, maintaining index:

- What:
  - Object name, file name, criteria (ID3) …
- Where:
  - (IP address, Port)
- Search engine, combining both information
- Global view of the network



Normal peer, maintaining the objects:

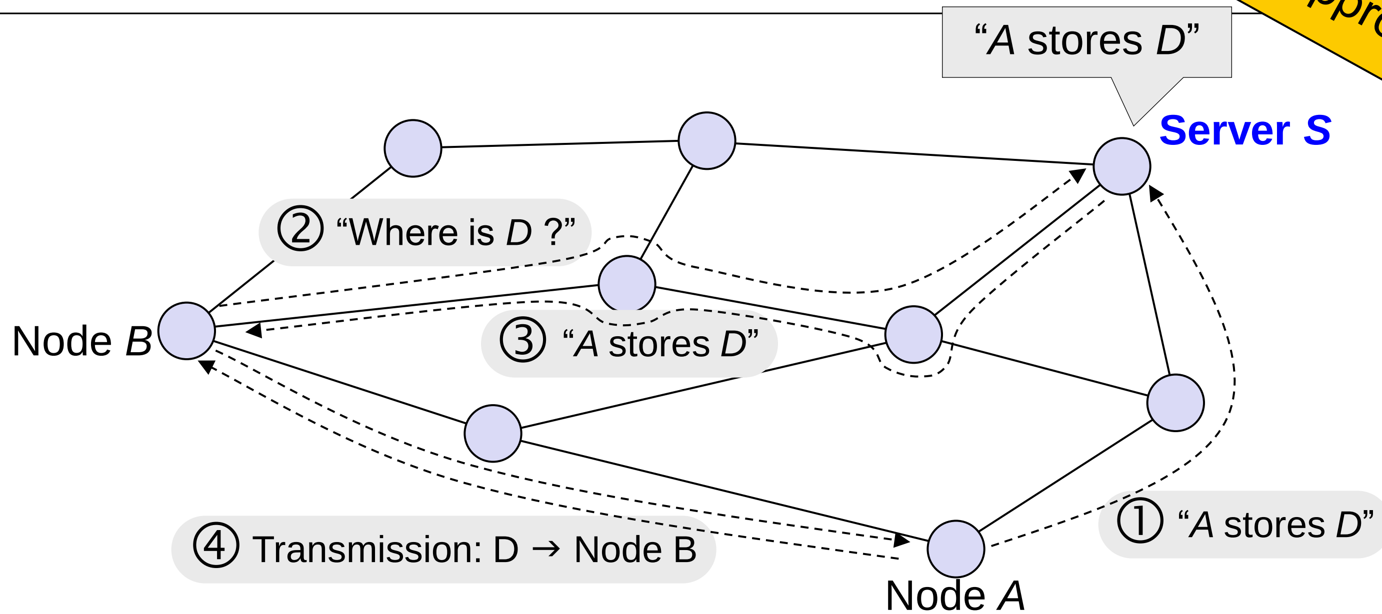
- Each peer maintains only its own objects
- Decentralized storage (content at the edges)
- File transfer between clients (decentralized)

Issues:

- Unbalanced costs: central server is the bottleneck
- Security: server is single point of attack

Centralized P2P Network

See Data Retrieval Approach I



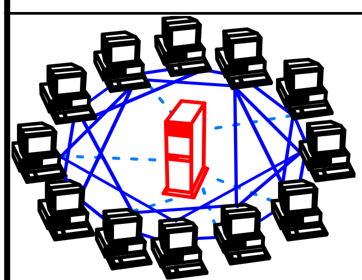
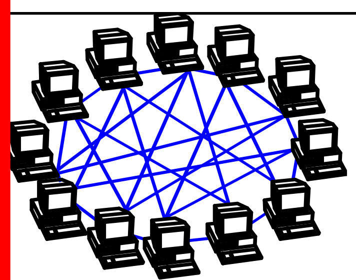
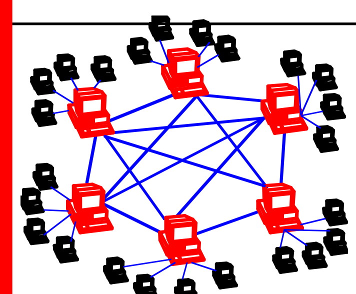
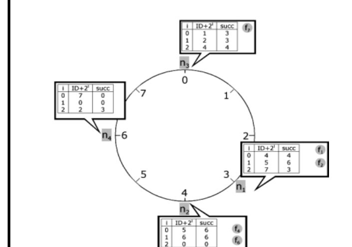
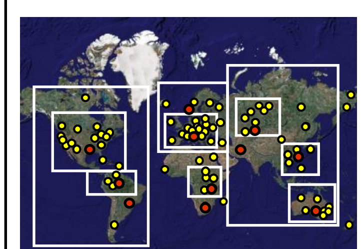
Simple strategy:

- Central server stores information about locations
  - ① Node A (provider) tells server that it stores item D
  - ② Node B (requester) asks server S for location of D
  - ③ Server S tells B that node A stores item D
  - ④ Node B requests item D from node A

The content of this slide has been adapted from "Peer-to-Peer Systems and Applications", edt. By Steinmetz, Wehrle

3.2 Unstructured Pure P2P Systems

TECHNISCHE UNIVERSITÄT DARMSTADT

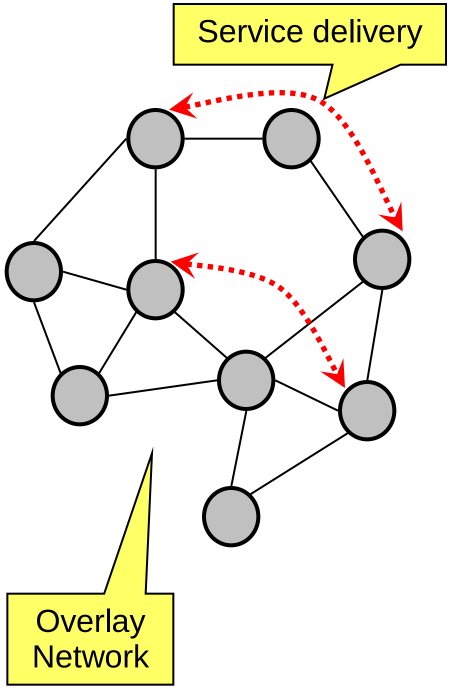
Unstructured P2P			*Structured P2P*	
Centralized P2P	***Pure P2P***	***Hybrid P2P***	***DHT-Based***	***Hybrid P2P***
1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Examples: ▪ Napster	**1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → no central entities Examples: ▪ Gnutella 0.4 ▪ Freenet**	**1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Examples: ▪ Gnutella 0.6 ▪ Fasttrack ▪ eDonkey**	**1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are “fixed” Examples: ▪ Chord ▪ CAN ▪ Kademlia**	**1. All features of Peer-to-Peer included 2. Peers are organized in a hierarchical manner 3. Any terminal entity can be removed without loss of functionality Examples: • RecNet • Globase.KOM**
				

from R.Schollmeier and J.Eberspächer, TU München

Distributed / Pure P2P Systems

Characteristics

- All peers are equal
  - (in their roles)
- Search mechanism is provided by cooperation of all peers
- Local view of the network



Organic growing: Just append to current network

- No special infrastructure element needed

Motivation:

- To provide robustness
- To have self organization

Tasks to solve

1. To connect to the network

- No central index server → Joining strategies needed
- To join → knowledge of at least 1 peer in the network
- Local view of network → advertisements needed

2. To search

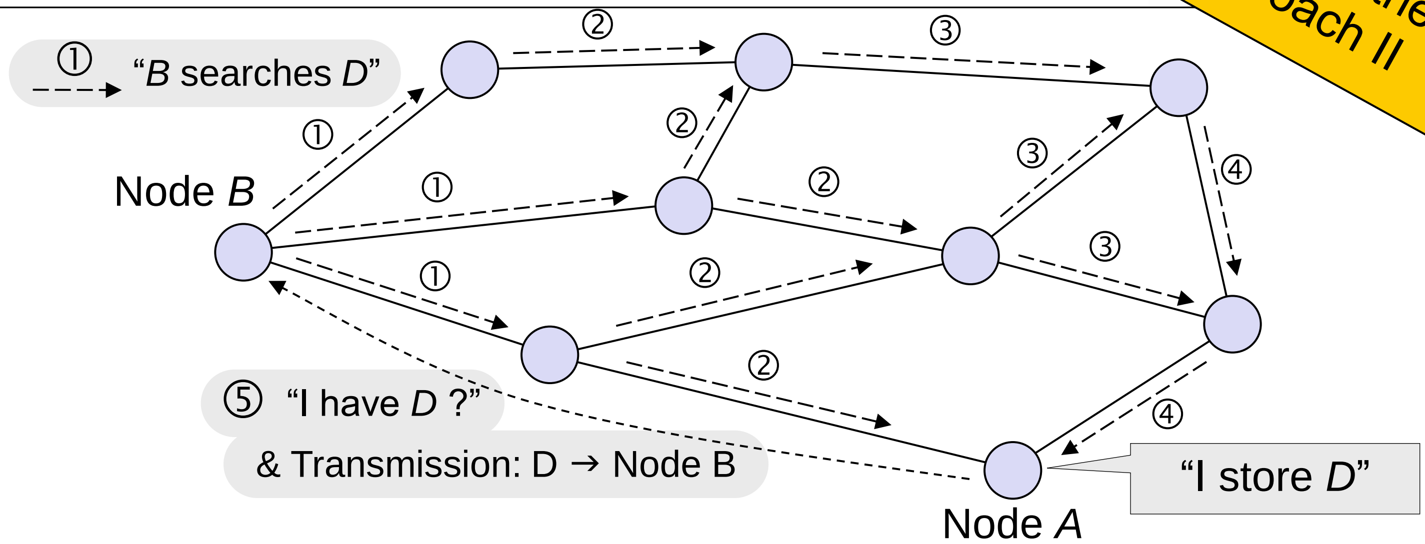
- Different search strategies available
- Providing different benefits & drawbacks

3. To deliver the service

- Establish connection to other node(s)
- Peer-to-peer communication

Search in Distributed / Pure P2P Systems

See Data Retrieval Approach II



Fully Decentralized Approach

- No information about location of data at intermediate systems
- Necessity for broad search
  - ① Node B (requester) asks neighboring nodes for item D
  - ②-④ Nodes forward request to further nodes (breadth-first search / flooding)
  - ⑤ Node A (provider of item D) sends D to requesting node B

The content of this slide has been adapted from "Peer-to-Peer Systems and Applications", edt. By Steinmetz, Wehrle

Properties of Distributed / Pure P2P Networks

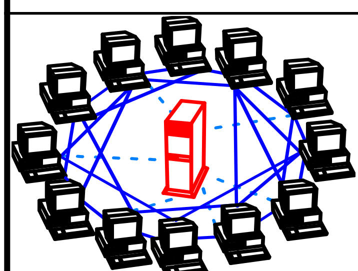
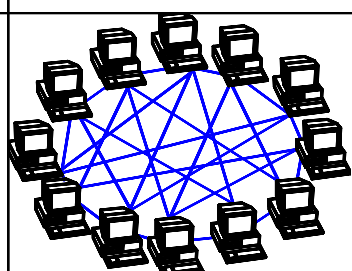
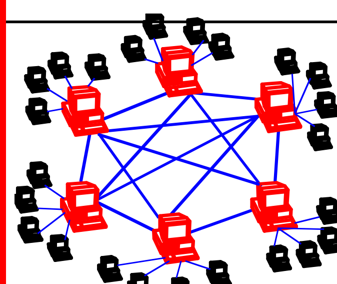
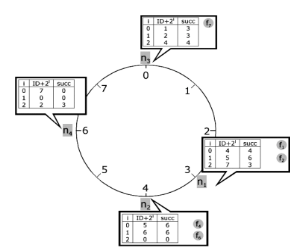
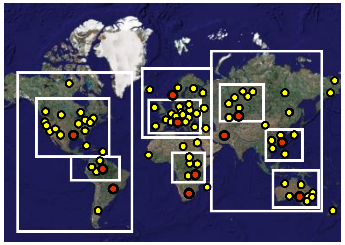
Benefits:

- Robustness: Every peer is dispensable
  - Switch off peer → no effect for network
- Balanced costs:
  - Each peer (generally) contributes the same
- Self organization

Drawbacks:

- Slow and expensive search
  - Flooding (to all connected nodes) is used to distribute information
- Finding all objects fitting to search criteria is not guaranteed
  - Object out of reach for search query

3.3 Unstructured P2P: Hybrid Systems

Unstructured P2P			Structured P2P	
Centralized P2P	*Pure P2P*	*Hybrid P2P*	*DHT-Based*	***Hybrid P2P***
1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Examples: • Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → no central entities Examples: • Gnutella 0.4 • Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Examples: • Gnutella 0.6 • Fasttrack • eDonkey	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" Examples: • Chord • CAN • Kademlia	1. All features of Peer-to-Peer included 2. Peers are organized in a hierarchical manner 3. Any terminal entity can be removed without loss of functionality Examples: • RecNet • Globase.KOM
				

from R.Schollmeier and J.Eberspächer, TU München

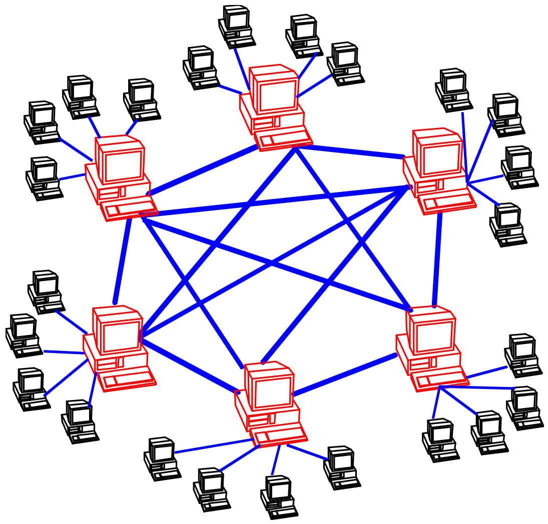
Principles

Approach:
To combine best of both worlds

- Robustness by distributed indexing
- Fast searches by server queries

Components

- Supernodes
  - Mini servers / super peers
  - Used as servers for queries
    - To build a sub-network between supernodes
    - Queries distributed at sub-network between supernodes
- "Normal" peers
  - Have only overlay connections to supernodes



++ Advantages

- More robust than centralized solutions
- Faster searches than in pure P2P systems

-- Disadvantages

- Need of algorithms to choose reliable supernodes

Picture from R.Schollmeier and J.Eberspächer, TU München

Types of Unstructured Hybrid P2P

TECHNISCHE UNIVERSITÄT DARMSTADT

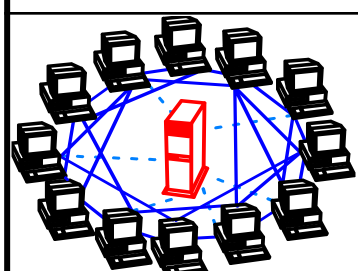
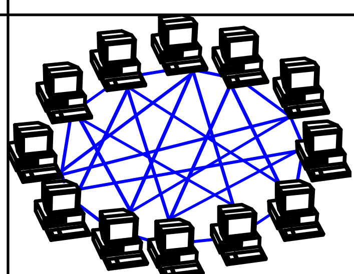
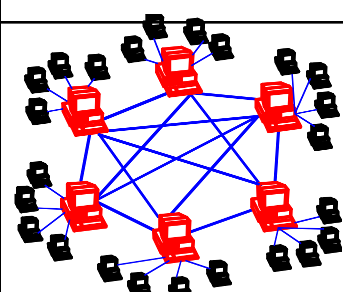
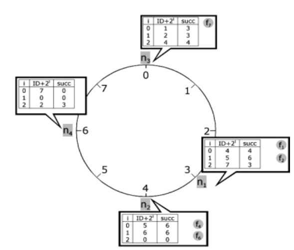
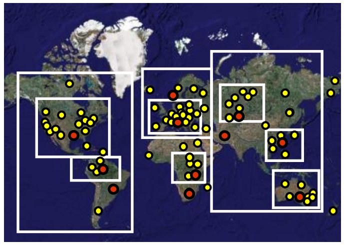
Performance improvements over centralized/pure systems:

- Decentralized by networks with supernodes / distributed servers
  - Decentralized File Sharing
    with Distributed Servers (like eDonkey 2000)
  - Decentralized File Sharing
    with Super Nodes (like KaZaA)

Incentives for Sharing (battling free riders)

  - Others: File Sharing with Charging like Mojo Nation
  - Cooperative File Sharing (like BitTorrent)

4 Structured Overlay Networks: DHT Systems

Unstructured P2P			Structured P2P	
Centralized P2P	***Pure P2P***	***Hybrid P2P***	***DHT-Based***	***Hybrid P2P***
1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Examples: ▪ Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → no central entities Examples: ▪ Gnutella 0.4 ▪ Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Examples: ▪ Gnutella 0.6 ▪ Fasttrack ▪ eDonkey	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" Examples: ▪ Chord ▪ CAN ▪ Kademlia	1. All features of Peer-to-Peer included 2. Peers are organized in a hierarchical manner 3. Any terminal entity can be removed without loss of functionality Examples: • RecNet • Globase.KOM
				

from R.Schollmeier and J.Eberspächer, TU München

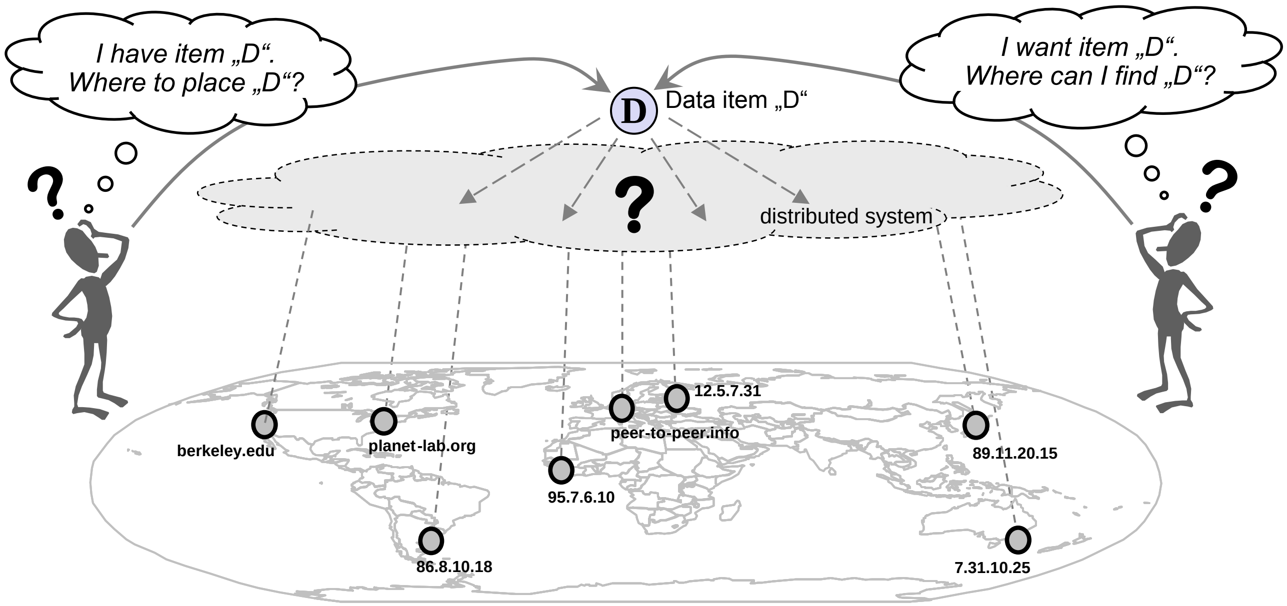
Motivation

- Efficient data location and retrieval
- Utilized for structured overlay P2P networks
- Relevant principle: Key / Value

Several approaches

- Chord
- Pastry
- CAN
- Tapestry
- Kademlia
- Omicron
- Viceroy
- …

Strategies for Data Retrieval



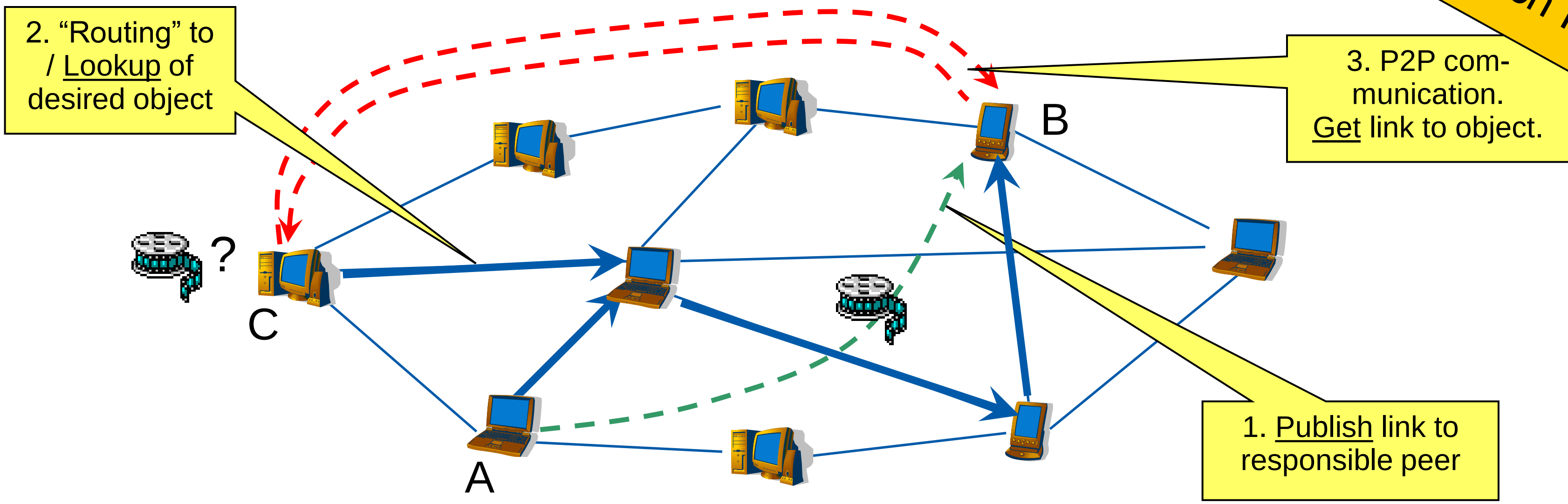
Strategies to store and retrieve data items in distributed systems

- Central server (central indexing)
- Flooding search (local indexing)
- Distributed indexing

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Look up in Structured P2P Systems

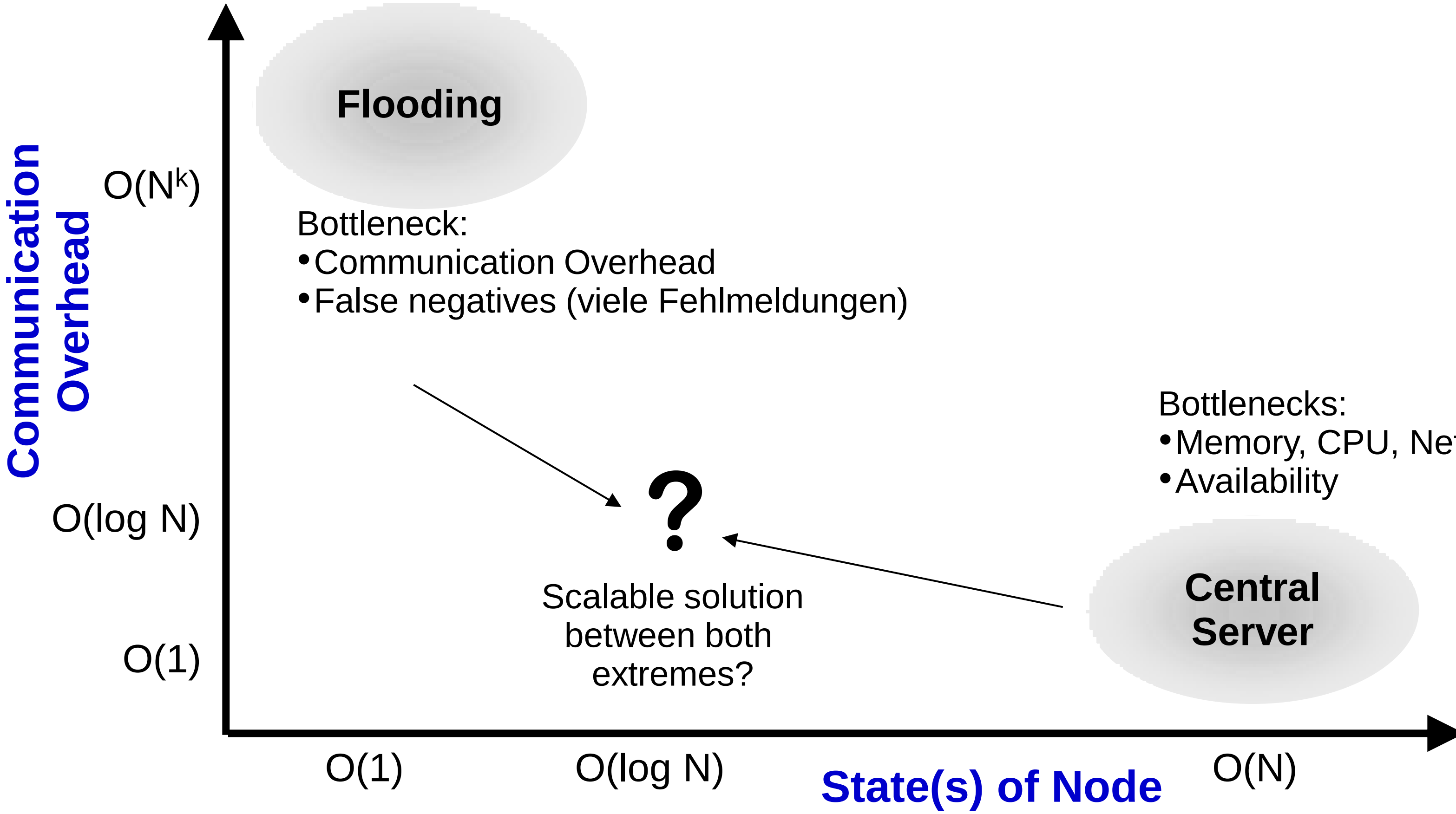
See Data Retrieval Approach III



Principle

- Location of the objects is found via routing
  - ① Node A (provider) advertises object at responsible peer B
    - Advertisement is routed to B.
  - ② Node C looking for object sends query
    - Query is routed to responsible node.
  - ③ Node B replies to C by sending contacting information of A

Motivation Distributed Indexing

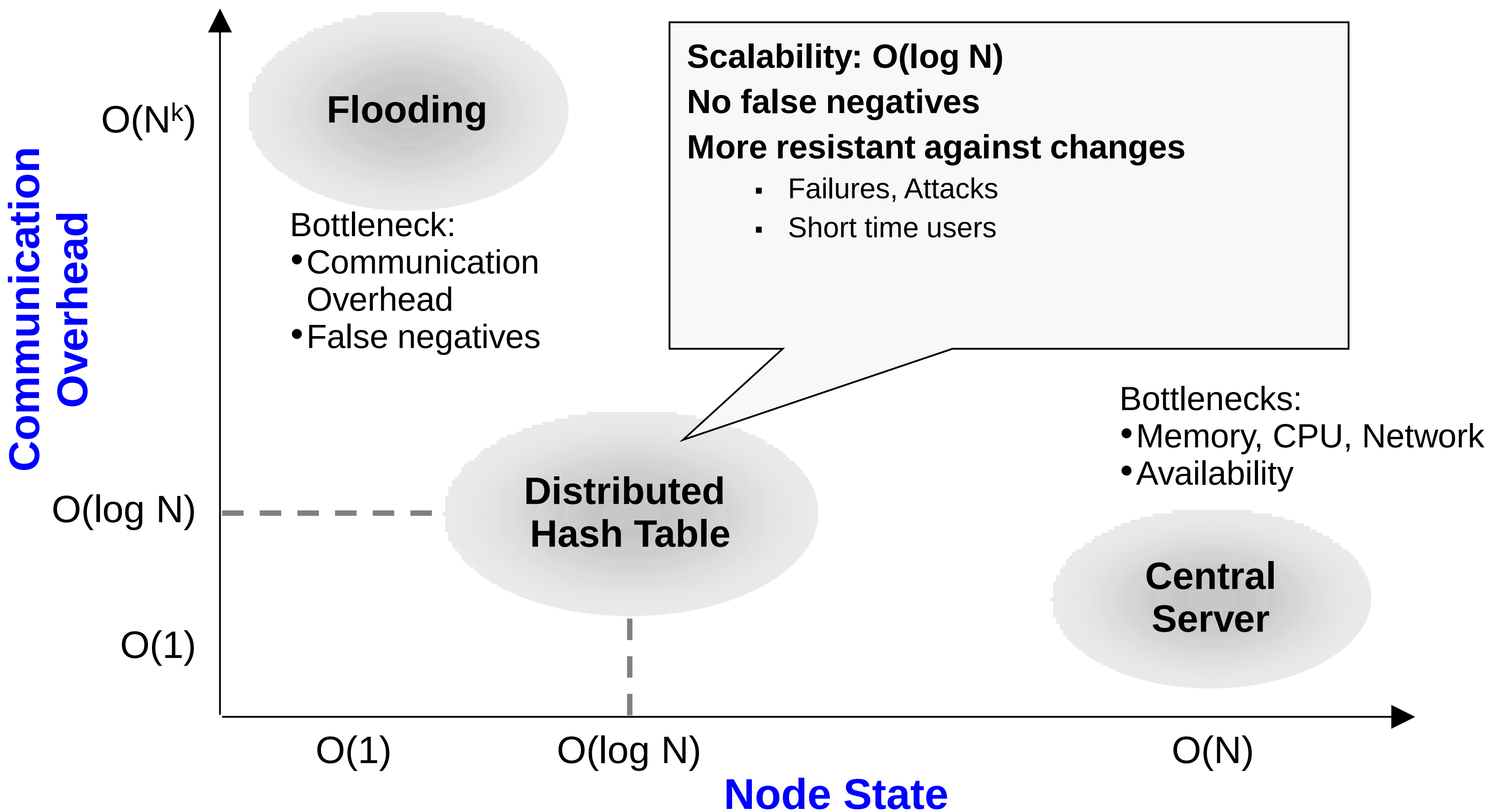


Communication overhead i.e.

- No. of hops vs.
- State(s) of node
  - (i.e. amount of routing entries stored in node, e.g. server)

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

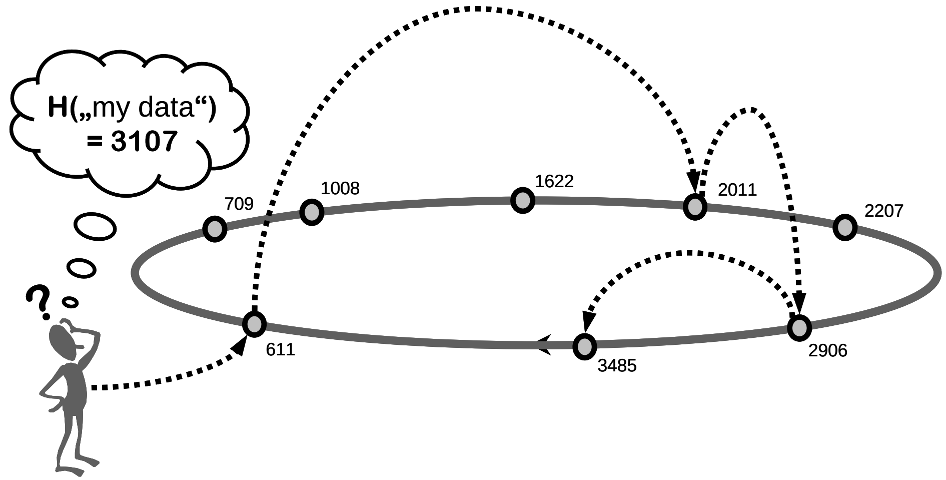
Communication overhead vs. node state



The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Distributed Indexing

TECHNISCHE UNIVERSITÄT DARMSTADT



Approach of distributed indexing schemes

- Data (resources, content) and nodes
  - Mapped onto same address space
- Intermediate nodes maintain routing information to target nodes
  - Efficient forwarding to „destination“
    - Content routing – not location-based routing
      - To reduce time needed to access content
  - Definitive statement of existence of content

Challenges & drawbacks

- Maintenance of routing information required
- Fuzzy queries not primarily supported
  - e.g., wildcard searches

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Sequence of operations

(at beginning) Mapping of nodes and data → same address space

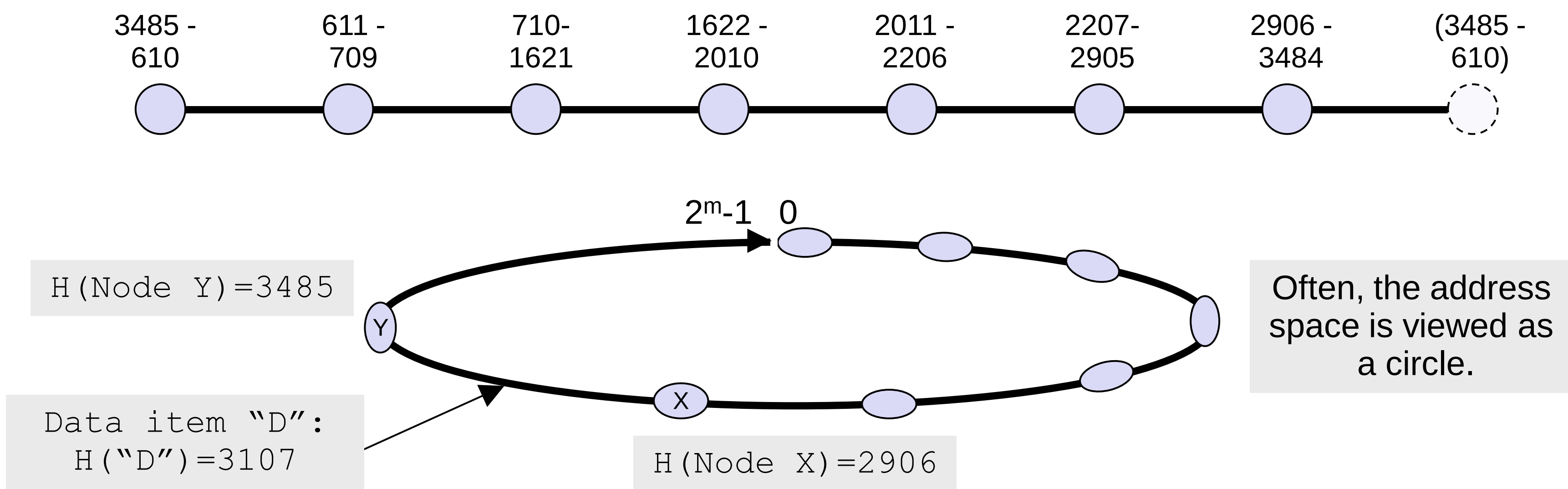
- Peers and content are addressed using flat identifiers (IDs)
- Common address space for
  - Data and nodes
- Nodes are responsible for data in certain parts of the address space
- Association of data to nodes may change since nodes may disappear

(later) Storing / Looking up data in the DHT

- “Look up” for data = routing to the responsible node
  - Responsible node not necessarily known in advance
  - Deterministic statement about availability of data

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Step 1: Addressing in Distributed Hash Tables

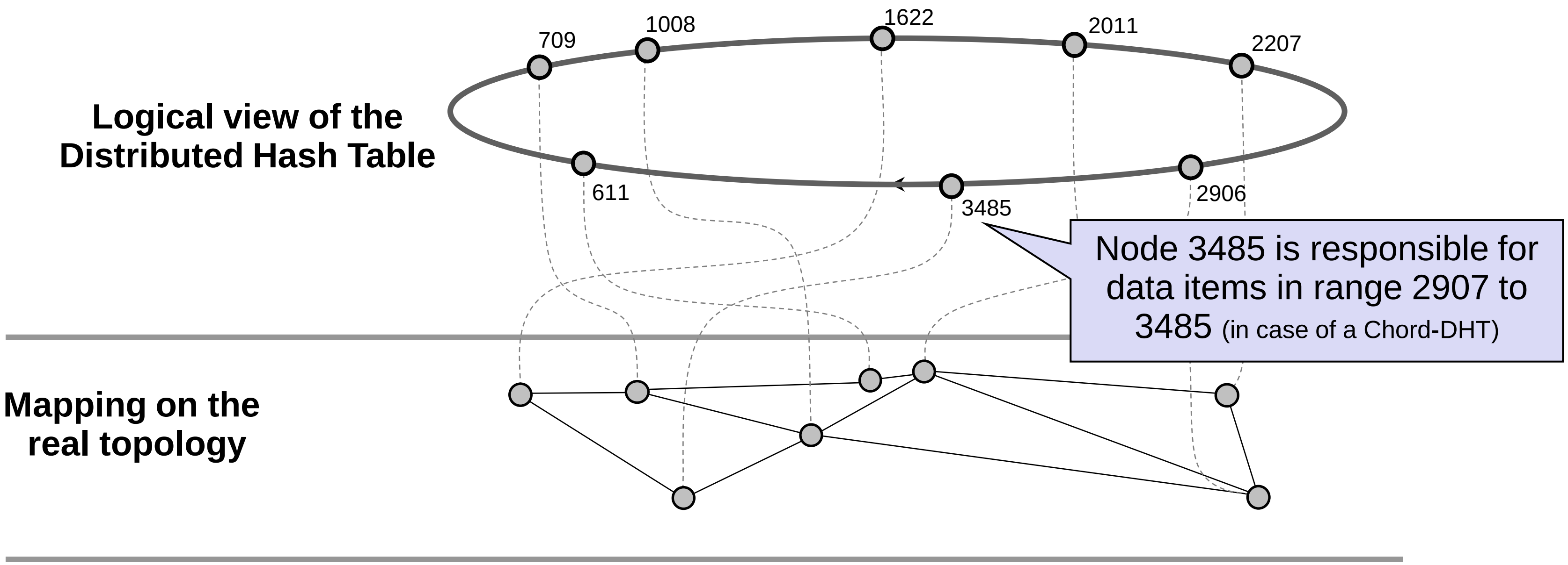


Mapping of content/nodes into linear space

- Usually: 0, …, 2^m-1 >> number of objects to be stored
- Mapping of data **and** nodes → onto same address space (e.g. 0 to 2^m-1)
  - With hash function
  - e.g., Hash(*string*) mod 2^m:
    - H(„*my data*") → 2313
- Association of parts of address space to DHT nodes

The content of this slide has been adapted from "Peer-to-Peer Systems and Applications", ed. by Steinmetz, Wehrle

Step 2: Association of Address Space with Nodes



Arrangement of the range of values

- Each node is responsible for part of the value range
  - Often with redundancy (overlapping of parts)
  - Continuous adaptation
- Real (underlay) and logical (overlay) topology are (mostly) uncorrelated

The content of this slide has been adapted from "Peer-to-Peer Systems and Applications", ed. By Steinmetz, Wehrle

Locating the data

- Content-based routing

Goal: Small and scalable effort

- O(1) with centralized hash table
  - But: Management of a centralized hash table too costly (server)

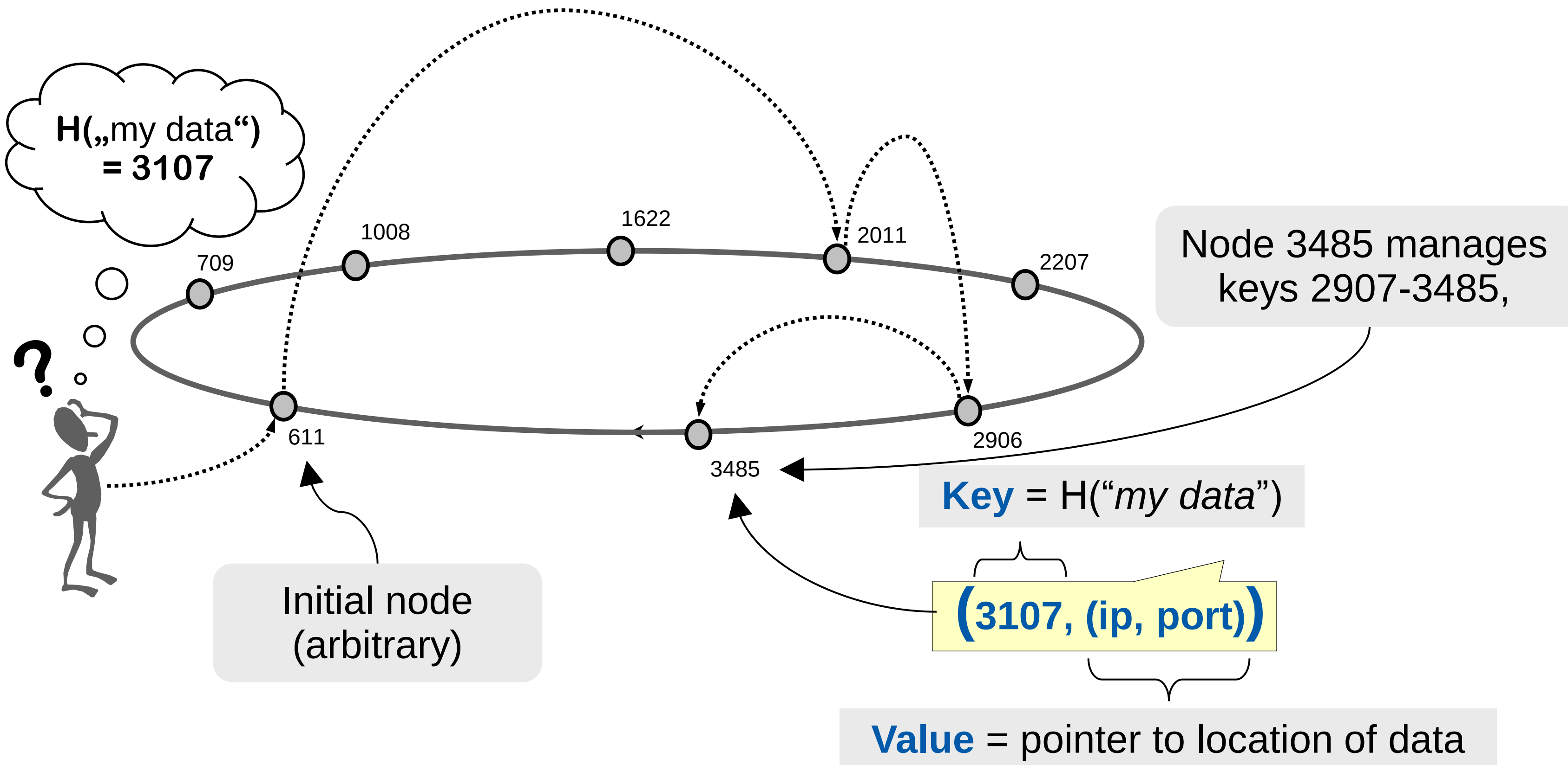
- Minimum overhead with distributed hash tables
  - O(log N):
    - DHT hops to locate object
  - O(log N):
    - Number of keys and routing information per node
      - (N = no. of nodes)

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

TECHNISCHE UNIVERSITÄT DARMSTADT

Routing to a key/value-pair

- Start lookup at arbitrary node of DHT
- Routing to requested data item (key)

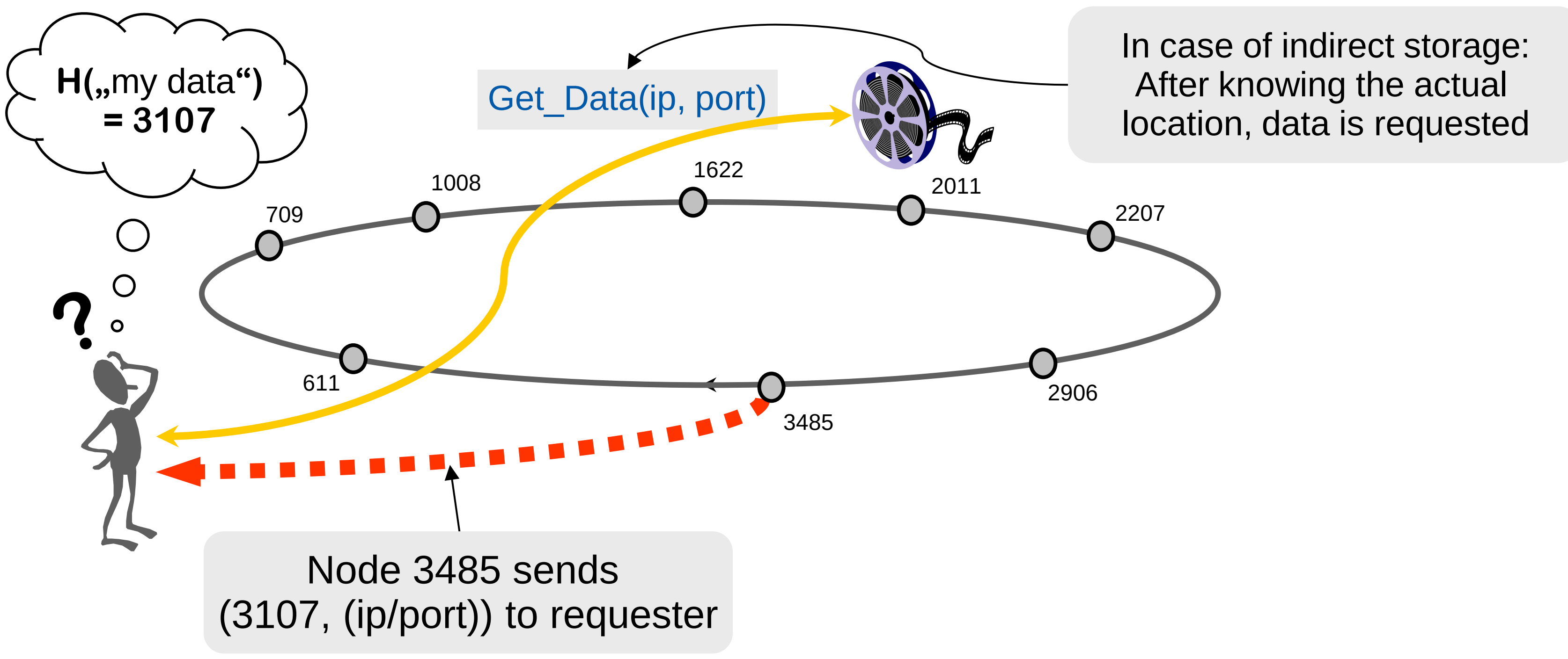


The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Step 5: Data Retrieval – Usage of located Resource

Accessing the content

- Key/value-pair is delivered to requester
- Requester analyzes key/value-tuple
  (and downloads data from actual location – in case of indirect storage)

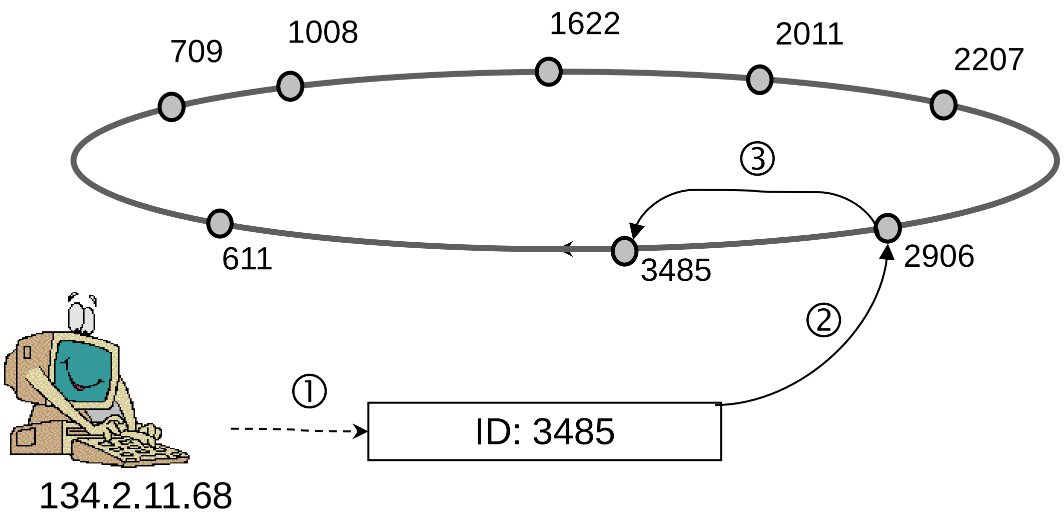


The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Distributed Hash Table: to Insert and to Delete a Node

Join of a new node

- 1. Calculation of node ID
- 2. New node contacts DHT via arbitrary node
- 3. Assignment of a particular hash range
- 4. Copying of key/value-pairs of hash range (usually with redundancy)
- 5. Binding into routing environment



The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Node Failure and Node Departure

Failure of a node

- Use of redundant key/value pairs (if a node fails)
- Use of redundant / alternative routing paths
- Key-value usually still retrievable if at least one copy remains

Departure of a node

- Partitioning of hash range to neighbor nodes
- Copying of key/value pairs to corresponding nodes
- Unbinding from routing environment

The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

Recall: Unstructured vs. Structured Overlay Networks

Unstructured overlay networks

- Location of resource ONLY known to submitter
- Peers & resources have NO SPECIAL identifier
- Each peer is responsible ONLY for the resources it submitted
- Introduction of new resource
  - At any location

Main task:
→To search

  - To find all peers storing/being in charge of resources fitting to some criteria
- And later to communicate directly peer-to-peers having identified these peers

Structured overlay networks

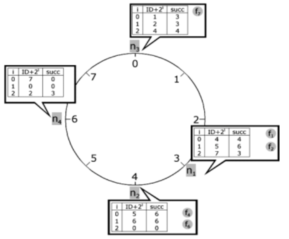
- Location of resources NOT only known to submitter
- Each peer may well be responsible for resources IT HAS NOT submitted
- Introduction of new resource(s)
  - At SPECIFIC location

i.e. to give peers and resources (unique) identifier

  - PeerIDs and ObjectIDs shall be from the same key set
  - Each peer is responsible for a specific range of ObjectIDs

Challenge: to find peer(s) with specific ID in overlay
→ To lookup

  - To “route” queries across the overlay network to peer with specific ID
- i.e. no search needed anymore



4.3 Chord: An Efficient Lookup Network

TECHNISCHE UNIVERSITÄT DARMSTADT

Chord uses SHA-1 hash function

- Results in a 160-bit object/node identifier
- Same hash function for objects and nodes

Node ID hashed from e.g., IP address

Object ID hashed from object name

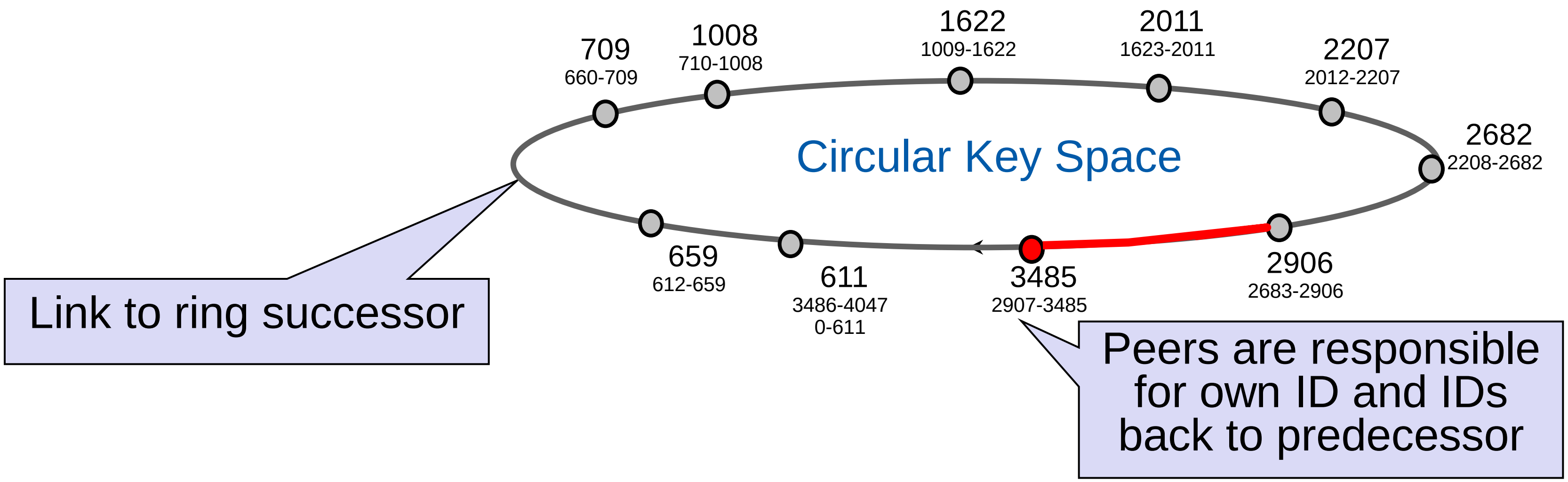
- Object names assumed to be known

Chord is organized in a ring which wraps around

- Nodes keep track of predecessor and successor
  - System invariant for valid network operation
- Node responsible for
  - Objects between its predecessor and itself
- Fingers used to enable efficient content addressing
  - O(log(N)) fingers lead to lookup operation of O(log(N)) length

Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications (2001) by Ion Stoica, et.al.

Chord: Network Topology



Uses SHA-1 (secure hash algorithm) to map

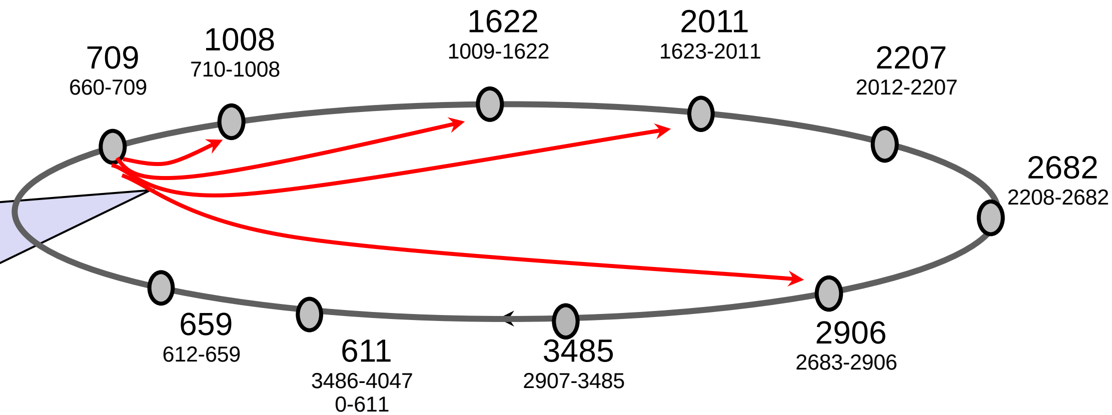
- IP address/object name onto
- 160 Bit ID

Basic ring topology

- Successor/ Predecessor

Chord: Network Topology

Fingers points to peers with ObjectIDs increasing exponentially.
Here: $709 + 2^k$
= …, 965, 1221, 1733, 2757

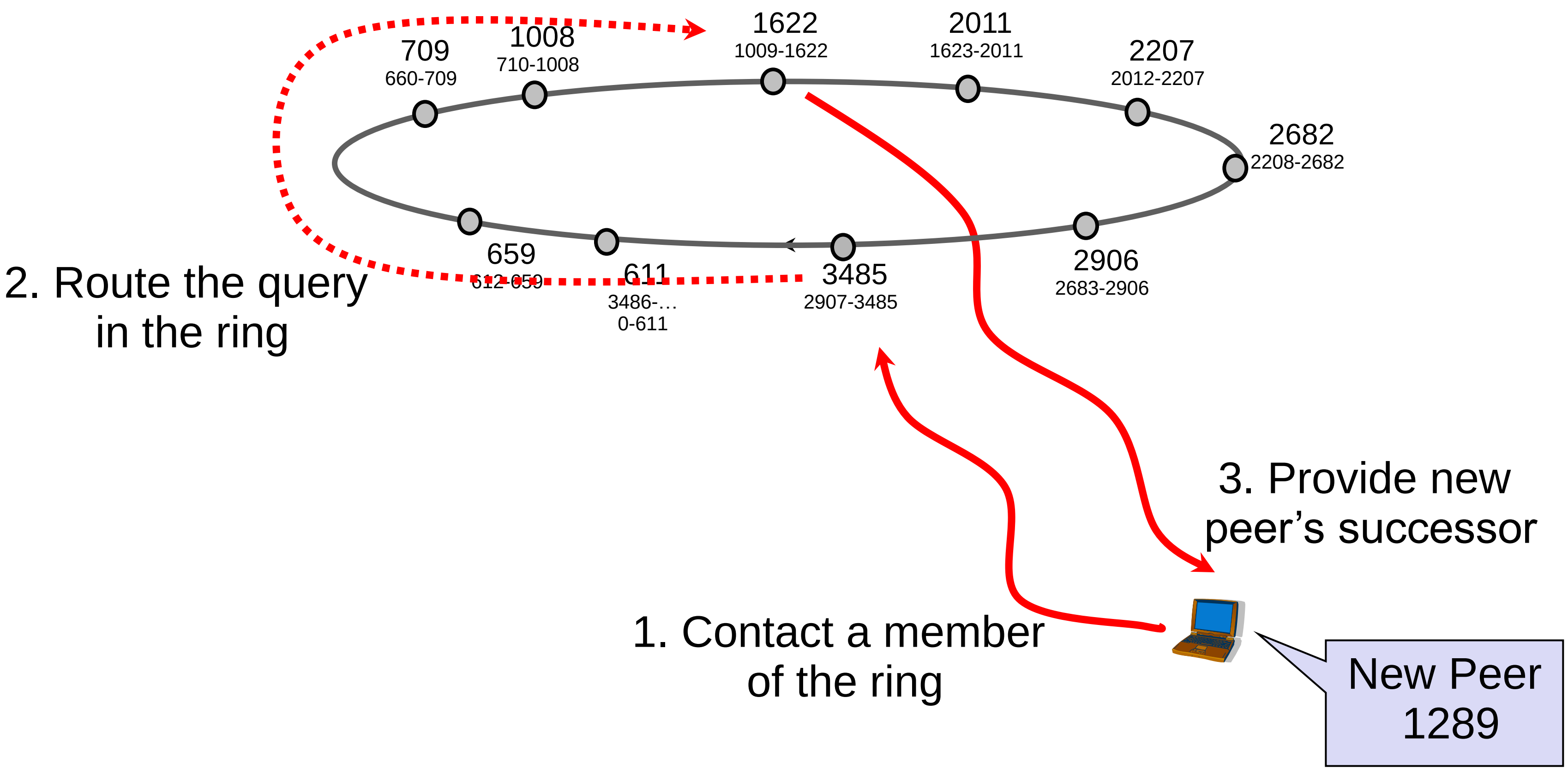


Enhanced topology

- k^{th} finger of Peer n is shortcut pointing to peers being responsible for Object ID $(n + 2^k)$
- k ranges from 0 to log(N)
- O(log(N)) fingers lead to lookup operation of O(log(N))

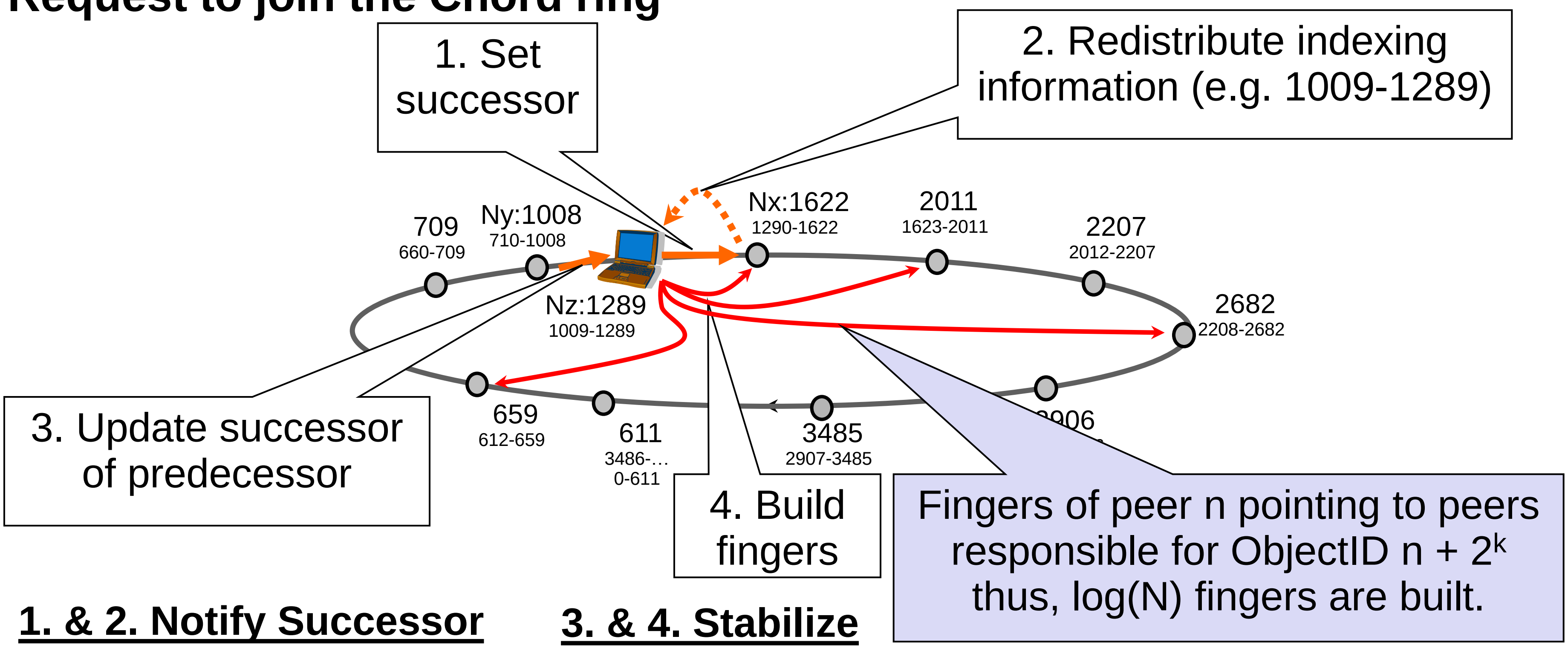
Chord: Join Procedure (1)

Request to join the Chord ring



Chord: Join Procedure (2)

Request to join the Chord ring



<u>1. & 2. Notify Successor</u>

Actions:

N_Z: Set Successor (N_X)

N_Z: Notify N_X

N_X: Set Predecessor

N_X: Copy items (index) to N_Z

<u>3. & 4. Stabilize</u>

Actions:

N_Y: Ask Predecessor of N_X

N_Y: Set Successor (N_Z)

N_Y: Notify N_Z

N_Z: Set Predecessor (N_Y)

N_X: Clear moved items

All: Fix Fingers

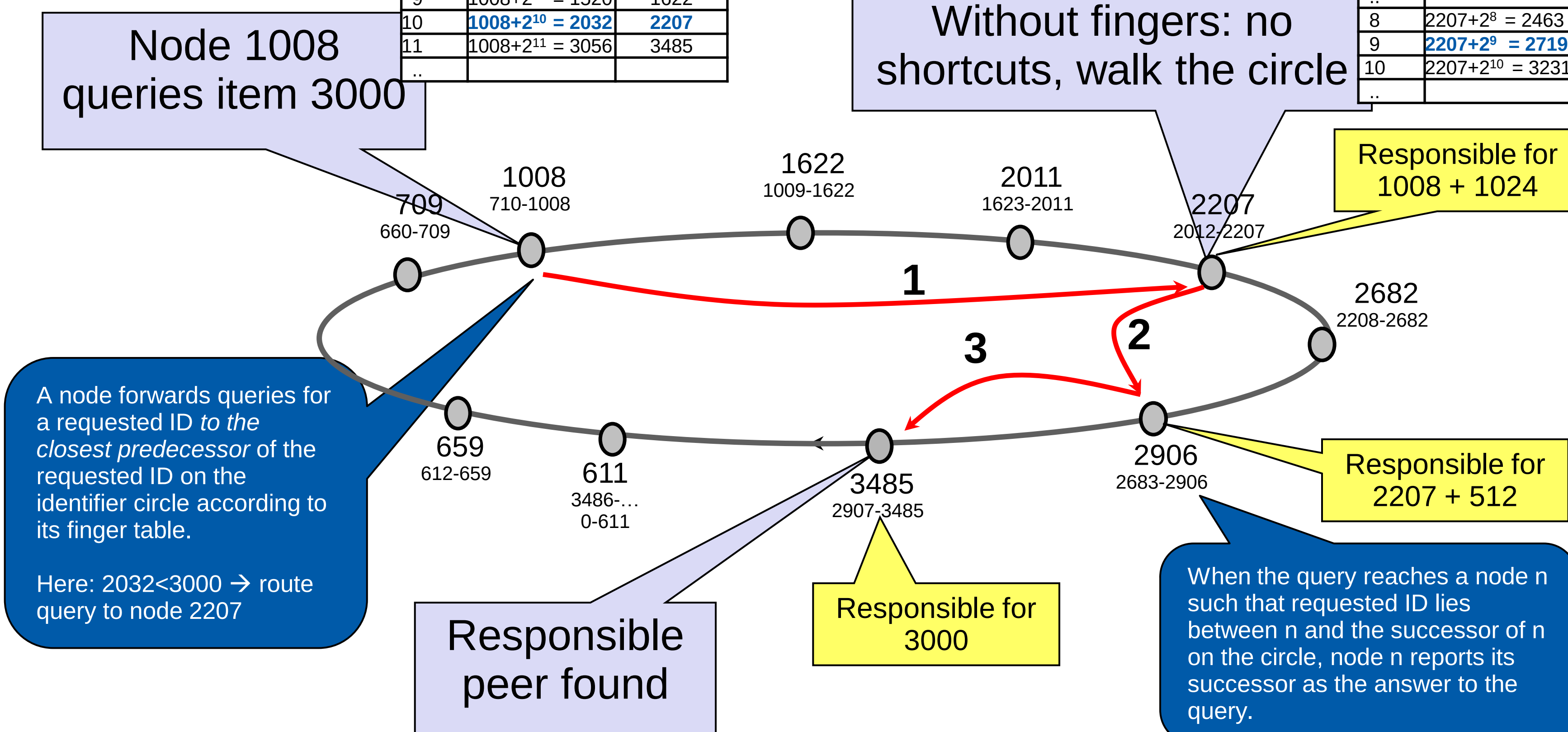
Chord: Addressing Content

Query

- Contains the hash value of the queried content
- On each step the distance from the destination is halved (remember fingers)

Finger	Target ID	Real peer ID
0	$1008+2^0 = 1009$	1622
1	$1008+2^1 = 1010$	1622
..		
9	$1008+2^9 = 1520$	1622
10	**$1008+2^{10} = 2032$**	**2207**
11	$1008+2^{11} = 3056$	3485
..		

Finger	Target ID	Real peer ID
0	$2207+2^0 = 2208$	2682
1	$2207+2^1 = 2209$	2682
..		
8	$2207+2^8 = 2463$	2682
9	**$2207+2^9 = 2719$**	**2906**
10	$2207+2^{10} = 3231$	3485
..		



Properties of Chord

Advantages

Efficient look up functionality

- Messages are routed within O(log N) steps

Low maintenance overhead

Easy to implement

Intuitive concept due to ring structure

Disadvantages

Not churn resistant

- Chord ring is likely to fail
- Insufficient stabilization mechanism

No support for heterogeneity

- All peers are treated equally
- Overloading of peers may happen

No built-in security mechanisms

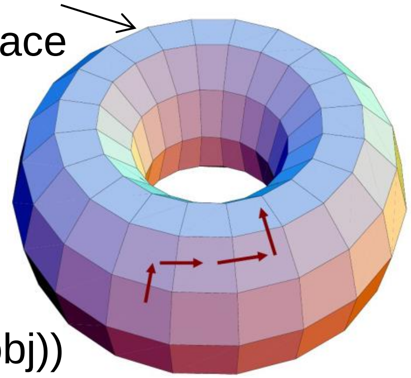
- Sensitive to malicious nodes

4.4 Content Addressable Network (CAN)

Architecture:

A hash-table in a d-dimensional Cartesian coordinate space, over a D-dimensional torus

- Cyclical d-dimensional space
- d hash-functions, 1 per coordinate
  - PeerID(p) = (h_1(p),h_2(p),... h_d(p))
  - ObjID(obj) = (h_1(obj),h_2(obj),..., h_d(obj))

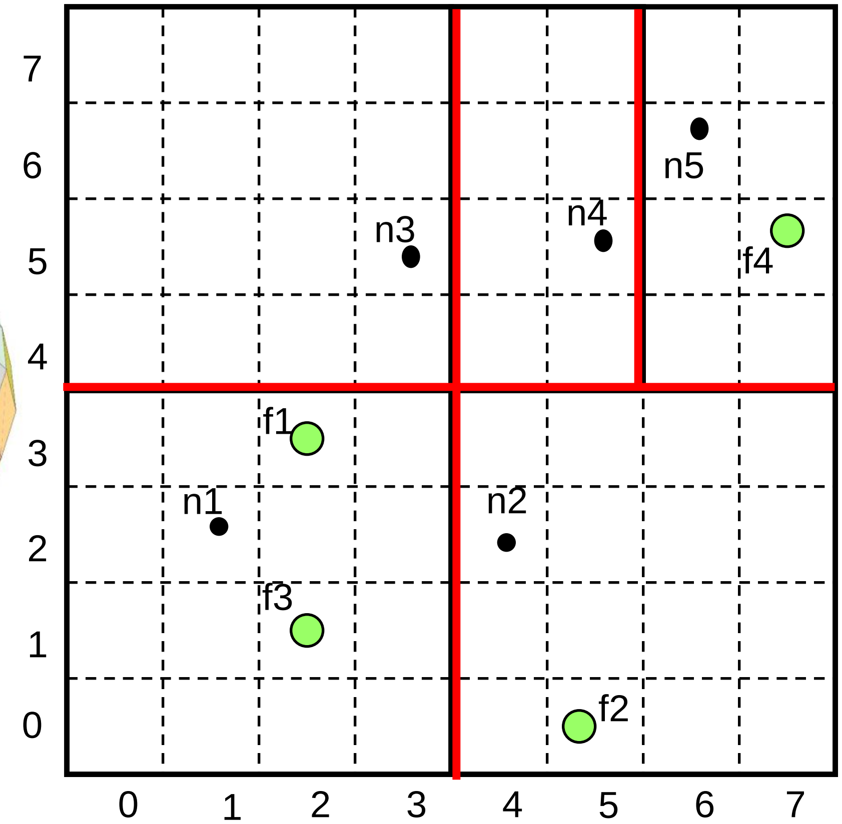


CAN nodes

- Each node is responsible for a distinct rectangular zone of the space
- Store all the files that hash into its zone

Nodes cover together the entire space

2-dimensional CAN



e.g.

- node n1 responsible for content f1 an f3

2 CAN nodes are neighbors if

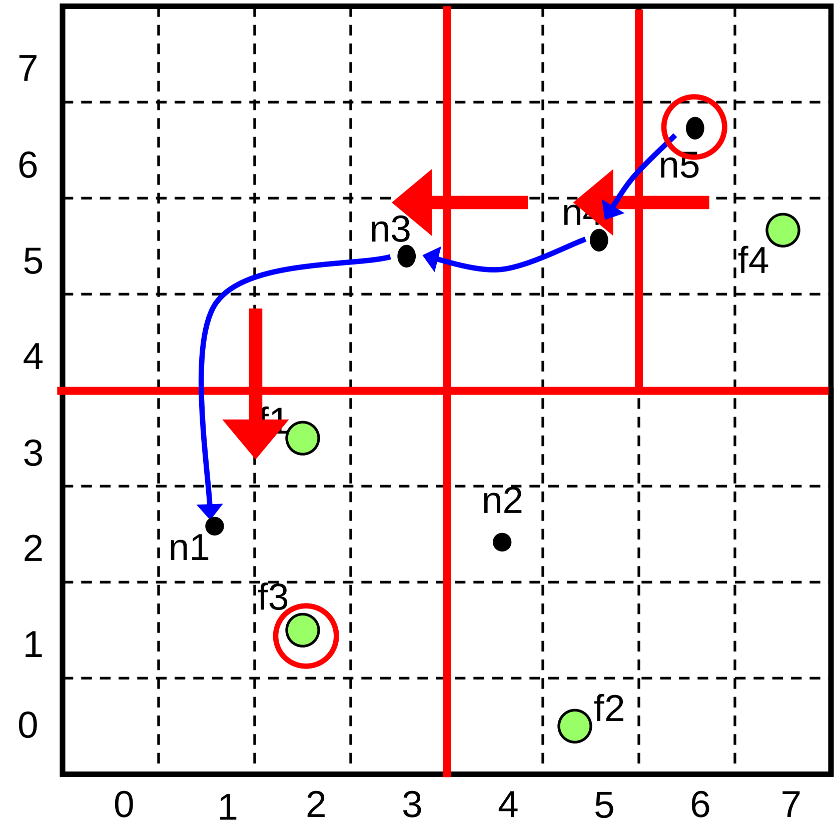
- Their zones overlap along d-1 dimensions and
- Abut along one dimension

→

- Every node knows
  - The IP addresses of its neighbors
  - The coordinates of neighboring zones
- Nodes can communicate only with their neighbors

Properties

- Routing table size O(d)
- Guarantees that a file is found in at most … steps, where n is the total number of nodes



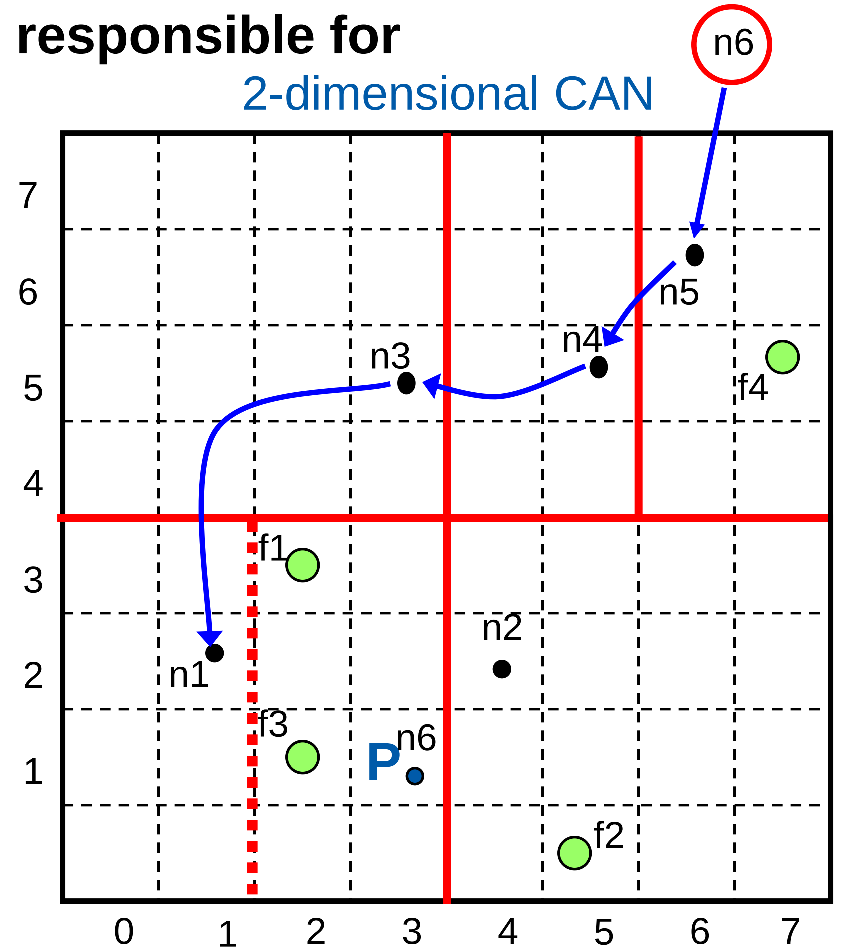
… at most $d*n^{1/d}$ steps

..

Abut = direkt angrenzen

CAN: New Peer Join

New node has to acquire a zone to be responsible for



Steps:

- Node chooses randomly a point P in the space
- Zone which includes P will be split in 2 halves

New node n6 requests to join:

1. contacts a node (e.g. n5)

2. selects point P

3. n5 routes the join query to n1

4. n1 splits its zone

5. n6 is responsible for

- The new zone (at point P)

CAN: Peer Crash / Leave

n7 crashes

- File f4 is lost

n4 and n5 realize failed node n7

n2, n4 and n5 start a timer for takeover

- Duration of timer is proportional to current size of zones

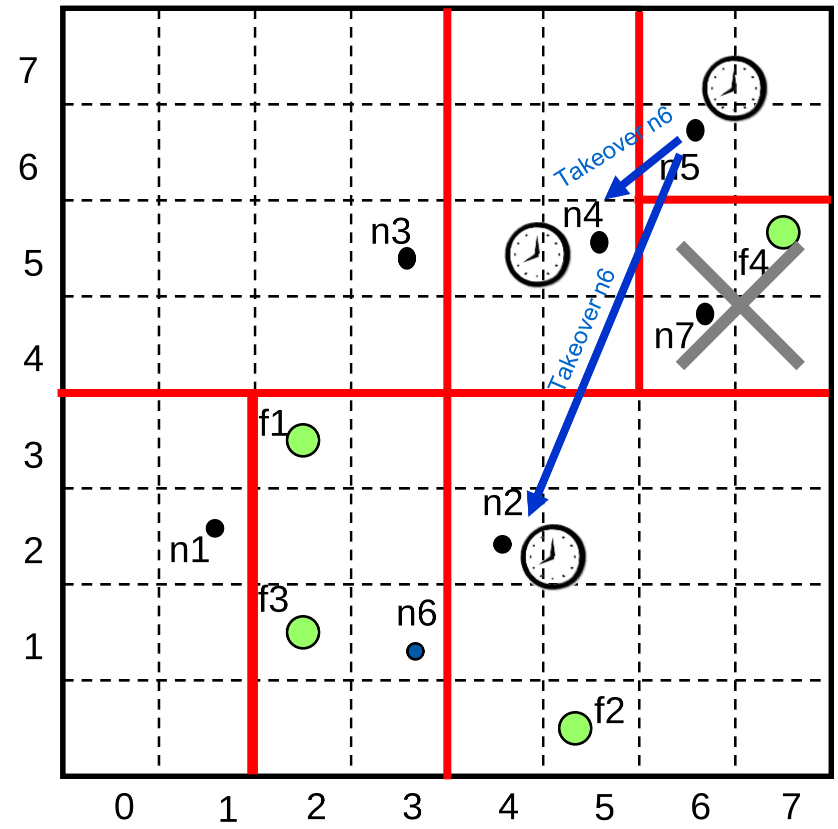
→ Preference towards nodes with smaller zones

- n5 will take over zone of crashed node

n5 sends takeover message to former neighbors of n7

- to n2 and n4 which stop their timers

2-dimensional CAN

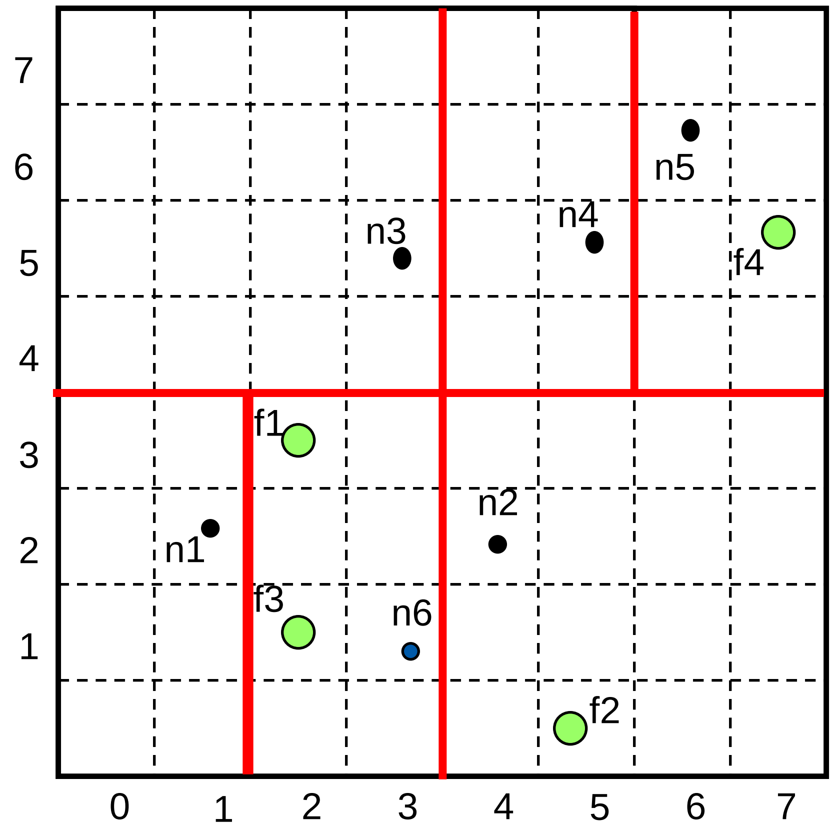


CAN: Zone Merging

n5 tries to merge zone with its own zone

- Try to get valid CAN splitting again
  - If possible: manage one big new zone
  - If not: manage both zone separately
- n5 merges zone of n7 and gets valid splitting

2-dimensional CAN



Properties of CAN

Advantages

Support for application layer multicast

Can be easily adapted to support spatial range queries

Small routing table size

- O(d) with d = number of dimensions
- Independent from the number of nodes

Intuitive routing concept

Disadvantages

Number of peers have to be known a priori to make CAN routing efficient

- Adapt number of dimensions
- Routing usually requires O(dn^(1/d)) hops

Merging of zones once a node goes offline/crashes is

- Difficult
- Time consuming

Proven not to be robust in case of high node churn

No support for heterogeneity

- All nodes are treated equally

No built-in security mechanisms

- Sensitive to malicious nodes

4.5 Kademlia

Use

- NodeID-based routing
  - tree-based routing table
- α parallel (simultaneous) ITERATIVE lookup to locate data
  - Retrieve data faster
  - Overcome faulty nodes
  - Usually α = 3 (parallel .. lookups)

Every node

- (and resource) has a 160-bit ID
- maintains information about resources most near-by

DHT-based overlay network using the XOR distance metric

- Simple operation
- XOR: Symmetrical routing paths
  A → B == B → A
  due to $d_{XOR}(A,B) == d_{XOR}(B,A)$

Store data with key X on the k nodes closest to X according to XOR metric

- built-in replication ensuring data availability
- Usually k = 20 closest nodes

Use lookup messages to maintain the overlay network

- To exchange routing table entries with look up messages
- i.e. to learn useful routing information from received lookup requests

XOR Distance Calculation:

ID Node A: 110101
ID Node B: 010001

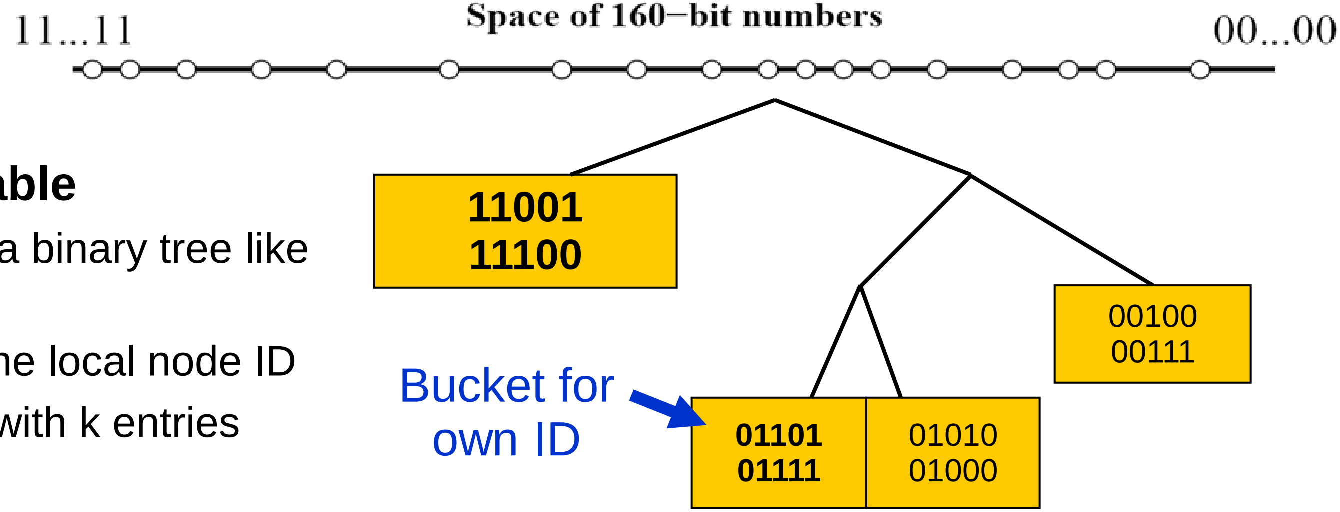
$d_{XOR}(A,B) = d(110101,010001)$

1 1 0 1 0 1
XOR
0 1 0 0 0 1
↓
1 0 0 1 0 0

$d_{XOR}(A,B) = 100100_2 = 36_{10}$

Concept of the Kademlia Routing Process

Example routing table for node 01110 with bucket factor k = 2



Structure of routing table

- Every node maintains a binary tree like routing table
- Tree branches along the local node ID
- Every leaf is a bucket with k entries

<u>Lookup procedure for a key X</u>

Step 1: Traverse routing table tree and pick bucket with node IDs closest to key X

Step 2: Put node IDs in a node list

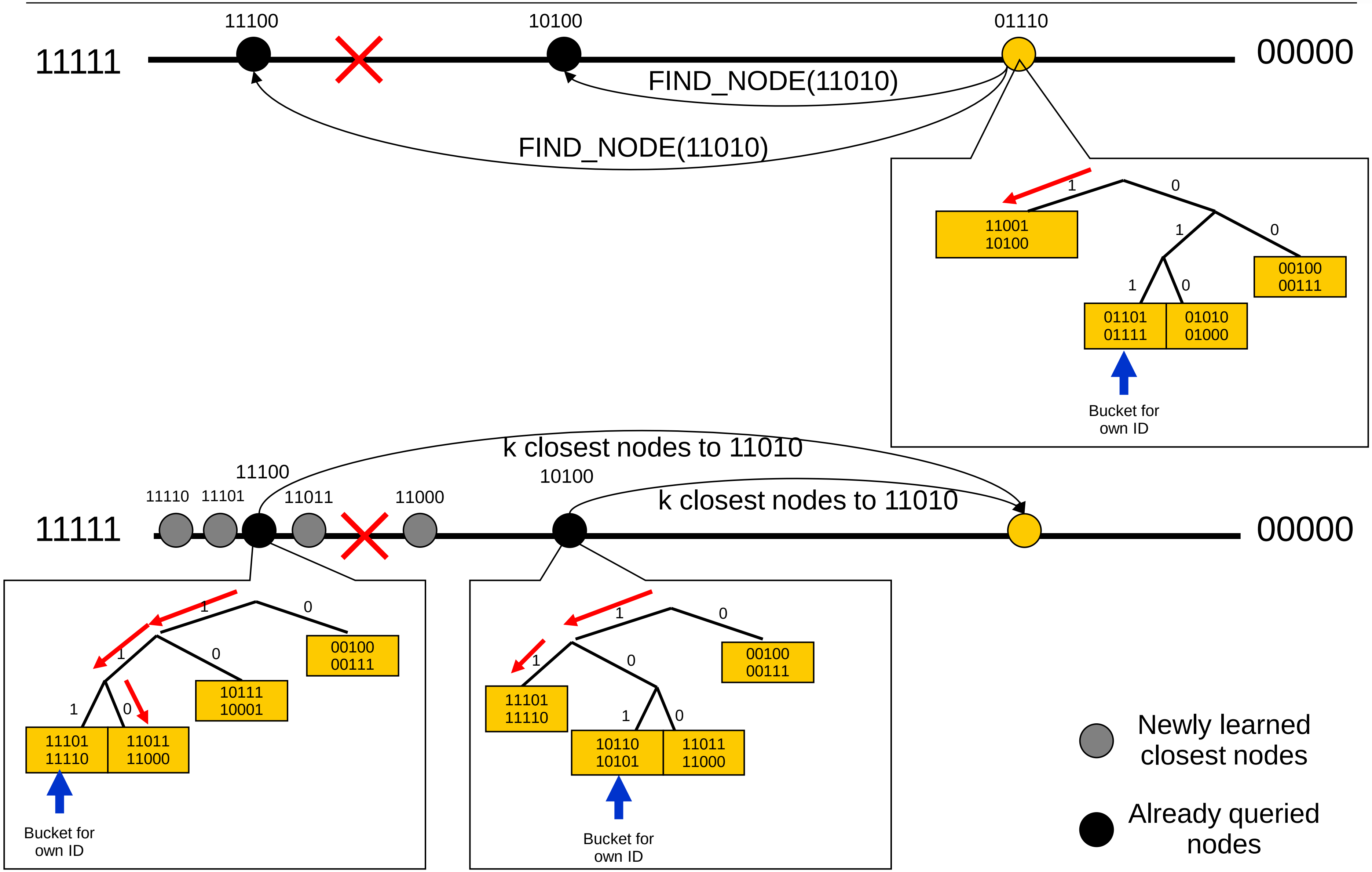
Step 3: Send parallel requests to the first α unvisited nodes in the node list

Step 4: Put received node IDs in node list

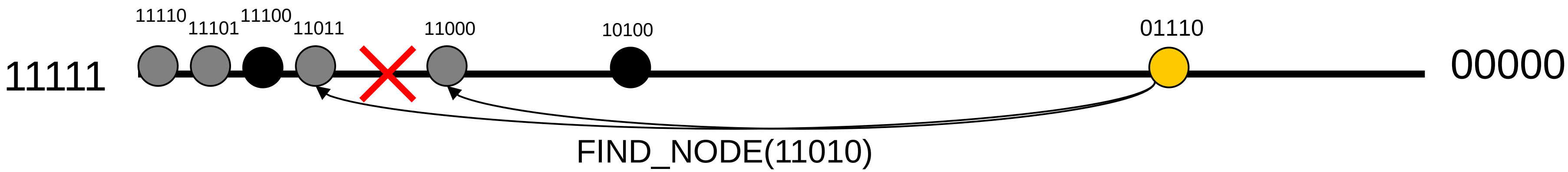
Step 5: Repeat Step 3 and 4 until set of k-closest nodes do not change anymore

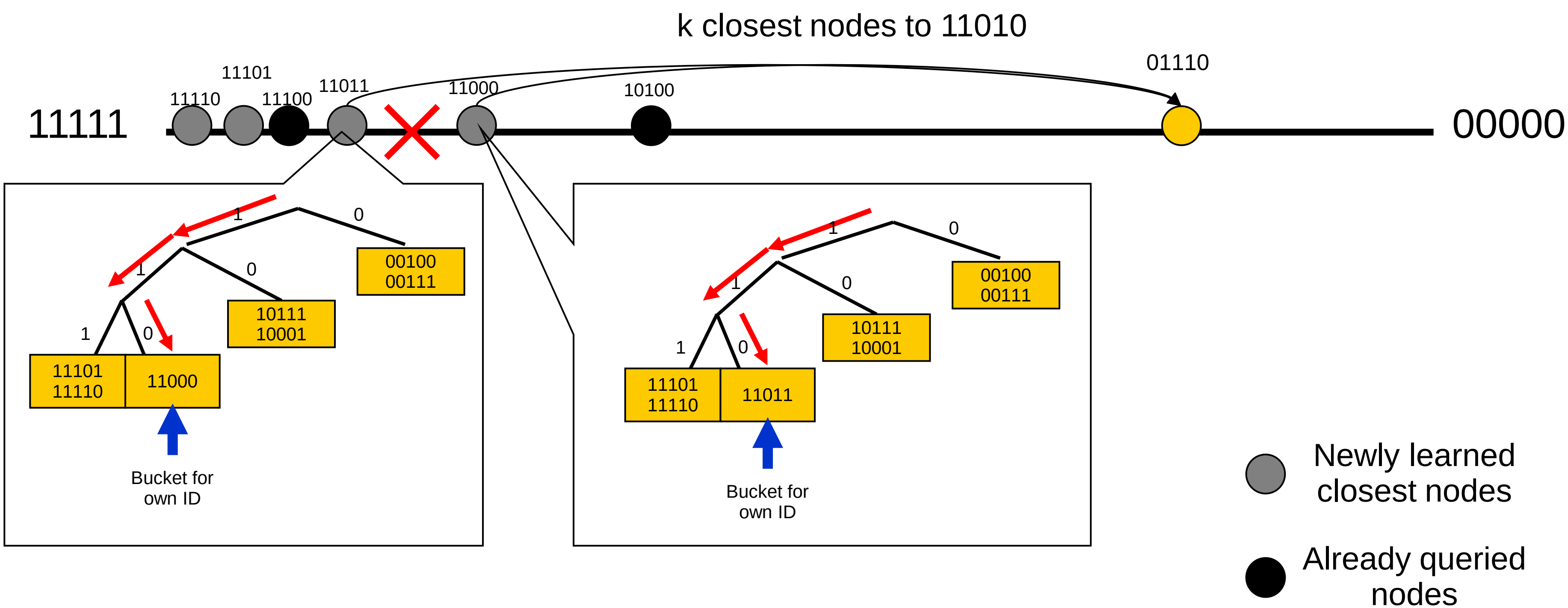
Step 6: Pick k-closest nodes from node list and send store or get data request to them

Routing from 01110 to 11010, α = 2, k = 2

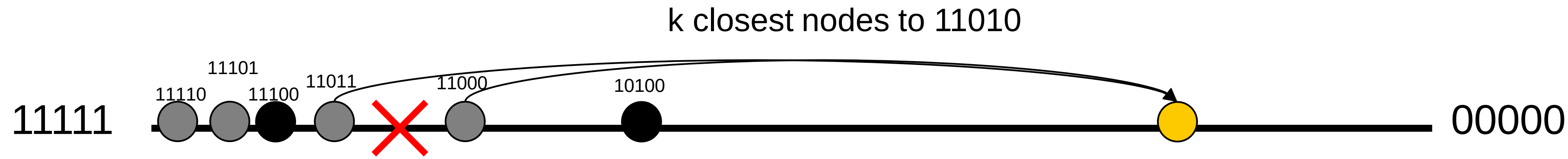


Routing from 01110 to 11010, α = 2, k = 2

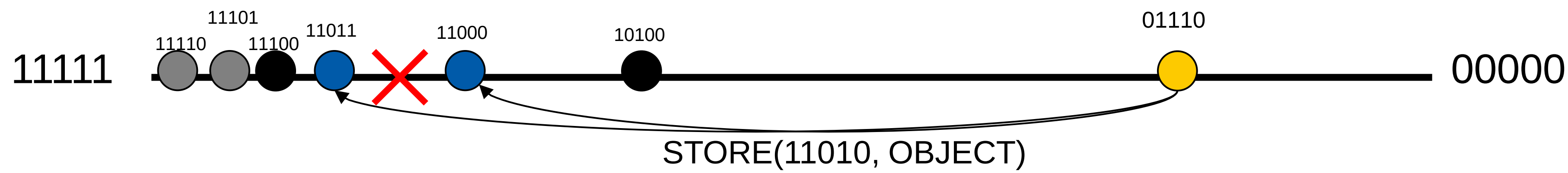


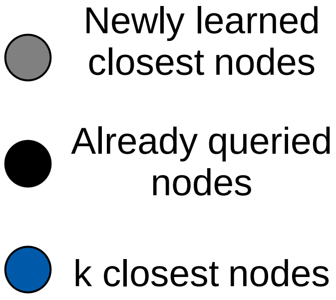


Routing from 01110 to 11010, α = 2, k = 2



No new nodes discovered → stop lookup process





The Kademlia protocol consists of 4 RPCs:

- FIND_NODE(KEY):
  - Recipient returns <IP Address, UDP Port, Node ID> triples for k closest nodes he knows about
- FIND_VALUE(KEY):
  - Like FIND_NODE
  - With exception:
    - If recipient already stores the value, the value is returned instead of k closest nodes
- PING(IP ADDRESS)
  - Probes a node to see if it is online
- STORE(KEY, VALUE)
  - Instructs a node to store a <key,value> pair

Construction of Routing Table

TECHNISCHE UNIVERSITÄT DARMSTADT

Each node maintains routing table (k buckets)

- Routing tables of different peers may be different

For each 0 <= i < 160 of the identifier space every node

- keeps a list of <IP Address, UDP Port, Node ID> triples
- for k nodes within range [2^i ; 2^(i+1)[
- in total k * 160 contacts

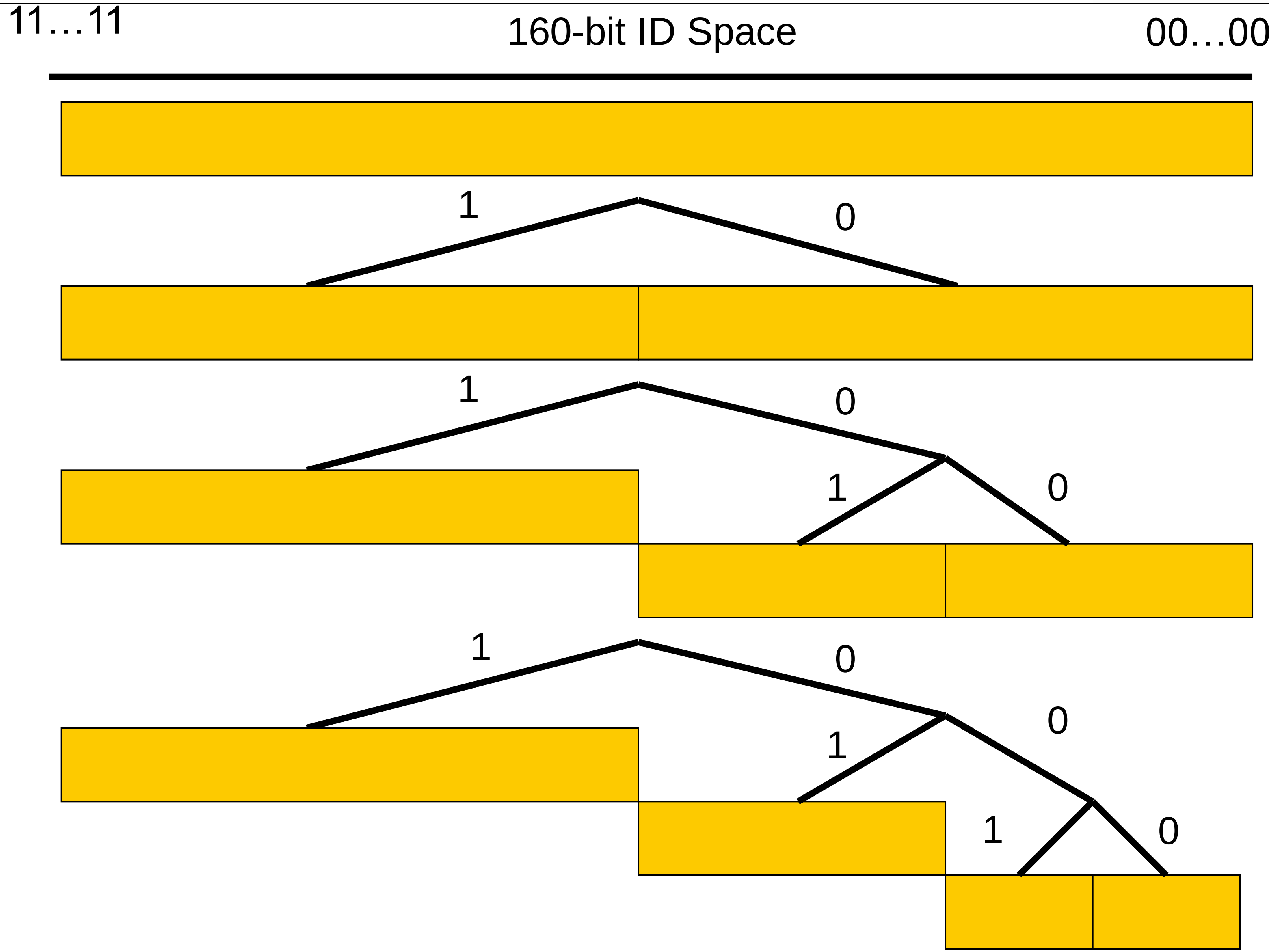
Nodes learn from

- messages they receive or
- using the FIND_NODE method

Preference towards old contacts

- Study has shown that the longer a node has been up, the more likely it is to remain up another hour
- Resistance against DoS attacks by flooding the network with new nodes

Evolution of the k Buckets



Example for Node ID 01110, k=2

11111 5-bit ID Space 00000

	01110 own ID

New node 11001:

11001	01110 own ID

New node 01101:

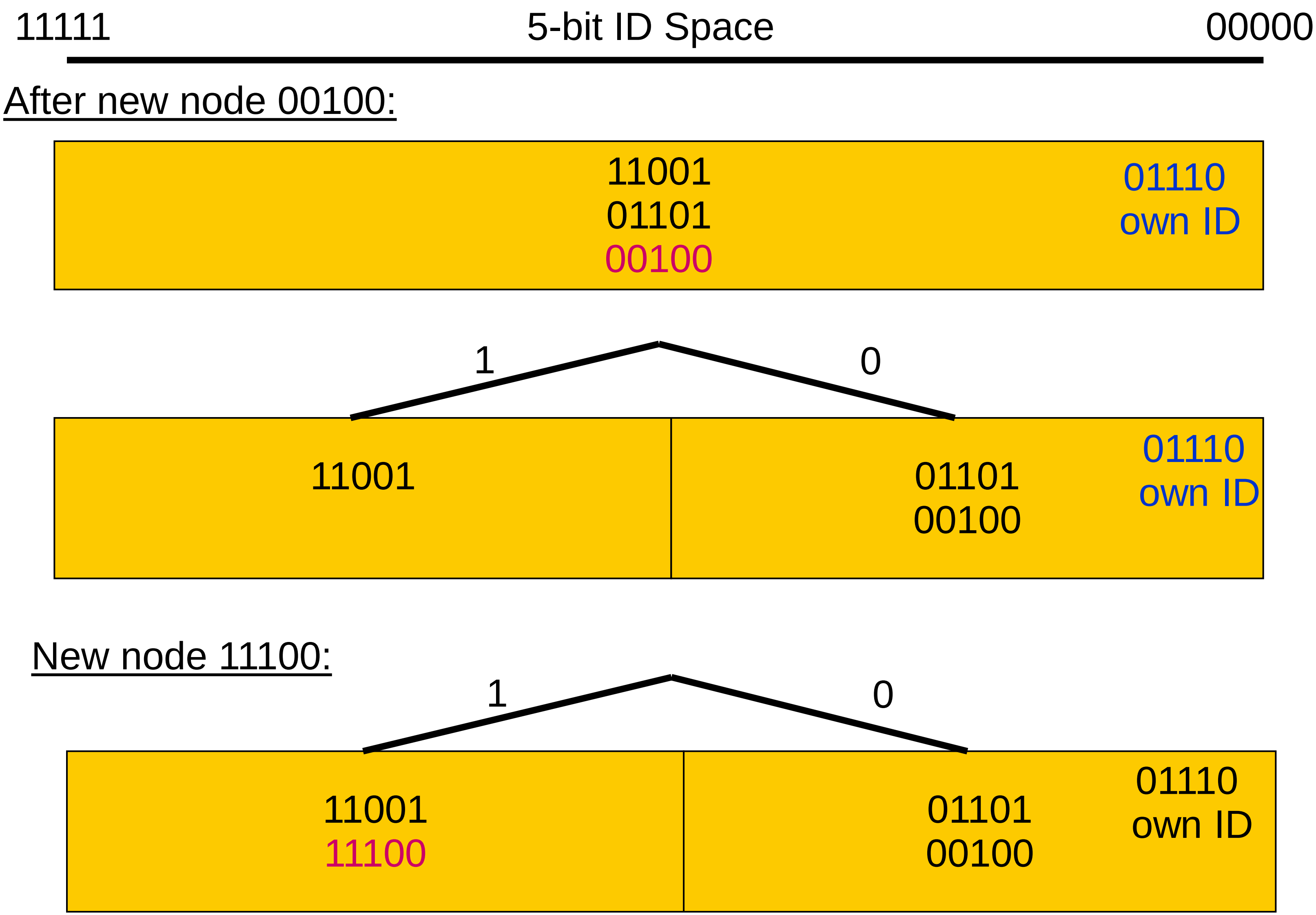
11001 01101	01110 own ID

New node 00100:

11001 01101 00100 but … no because 3rd, and k=2 is max.	01110 own ID

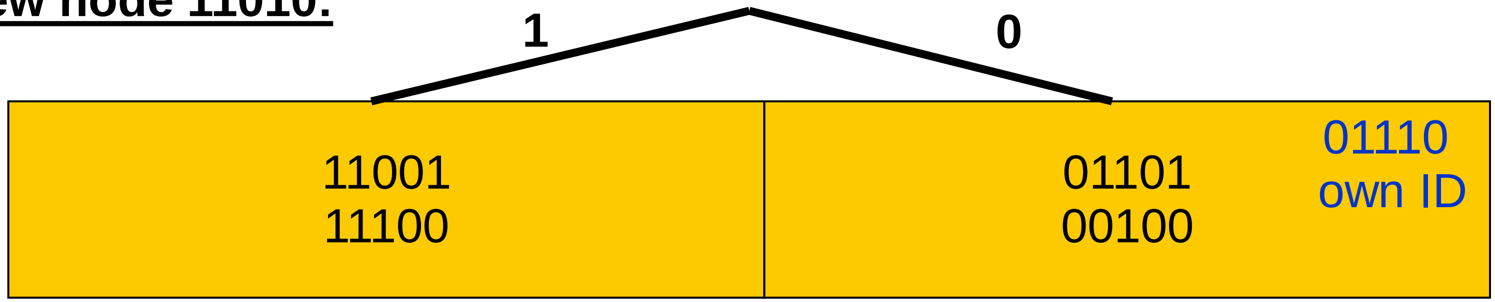
k-bucket is full → split necessary

Example for Node ID 01110, k=2



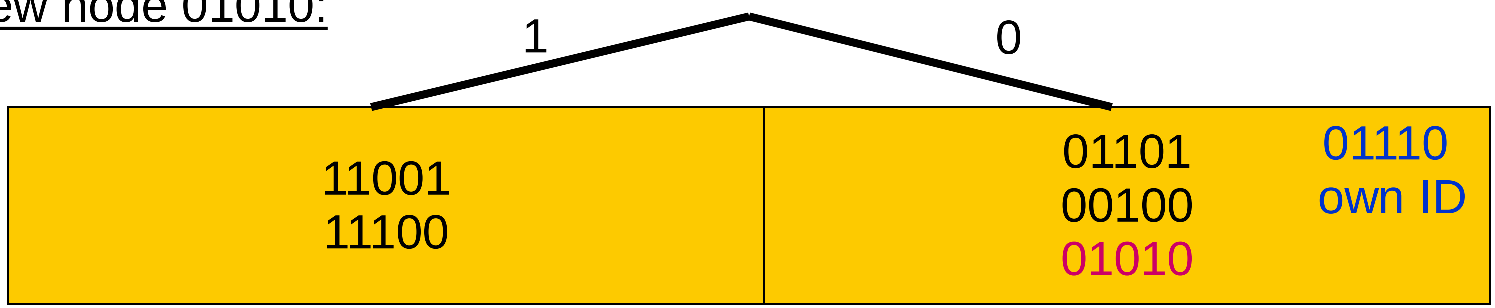
Example for Node ID 01110, k=2

New node 11010:



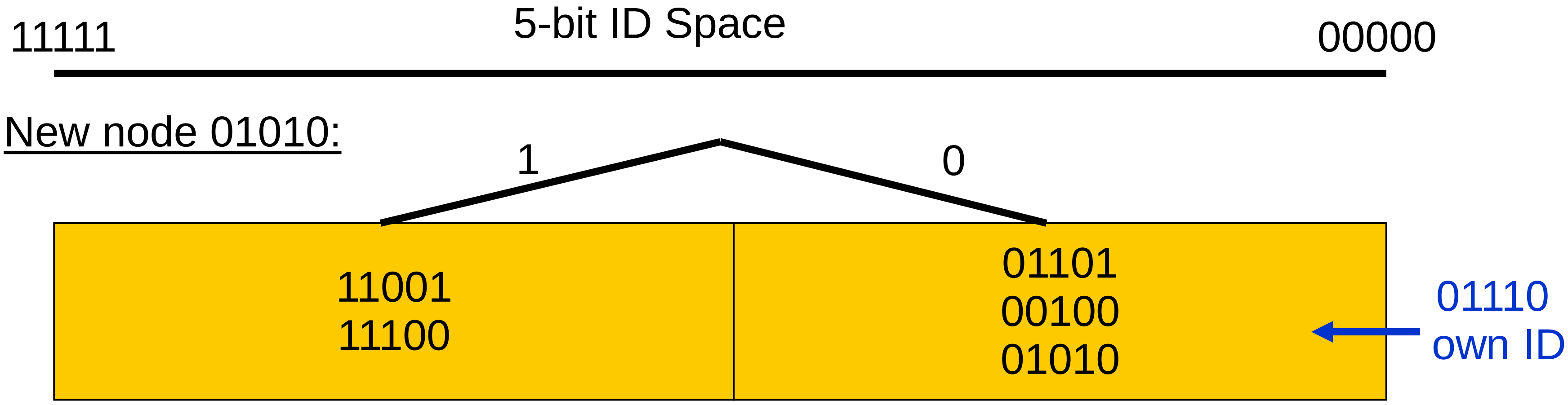
Left k-bucket full and 11010 NOT in k-bucket-range of 01110 and node 11001 still alive → node is dropped

New node 01010:

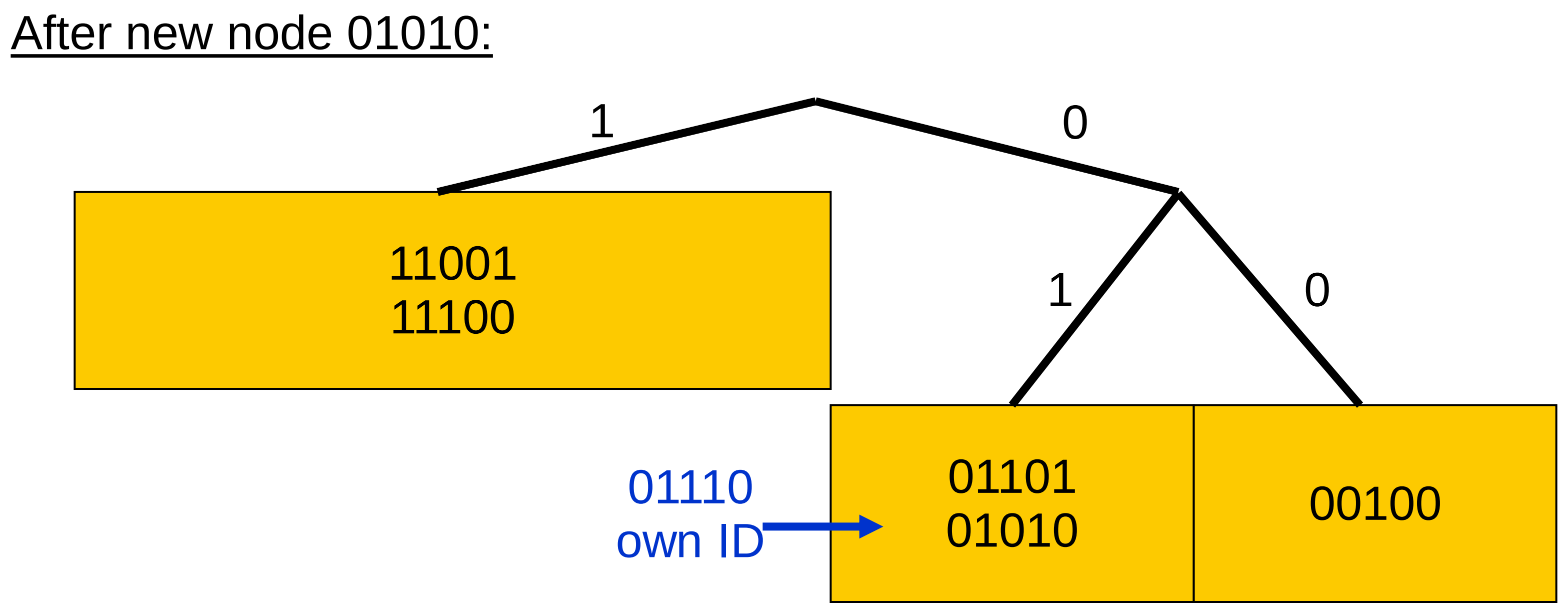


Right k-bucket full and 01010 in k-bucket-range of 01110 → split necessary

Example for Node ID 01110, k=2



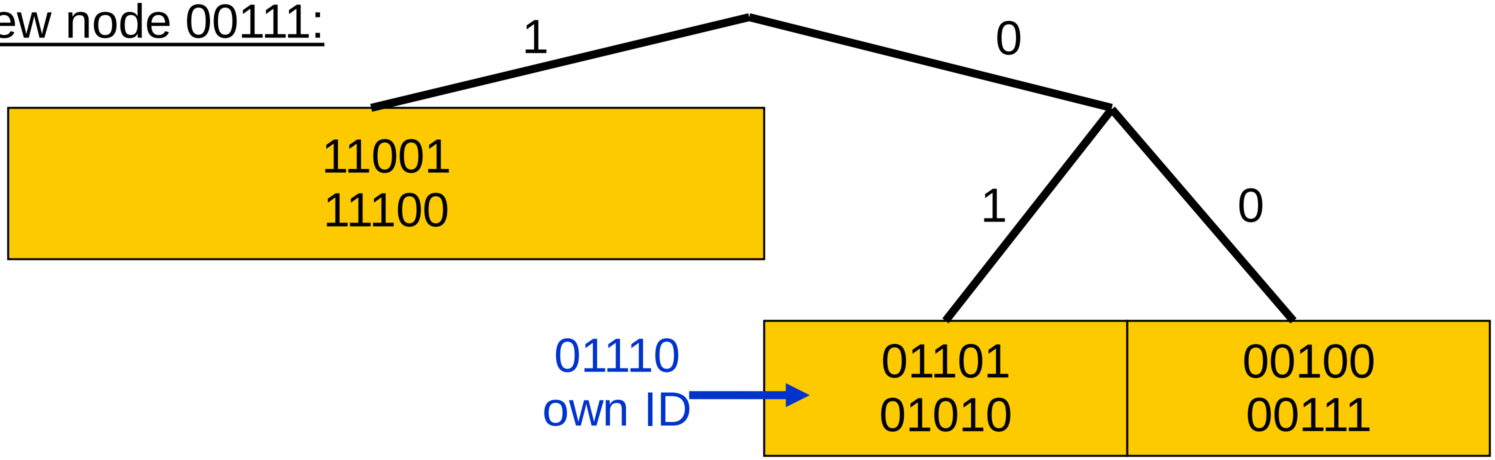
Right k-bucket full and 01010 in k-bucket-range of 00110 → split necessary



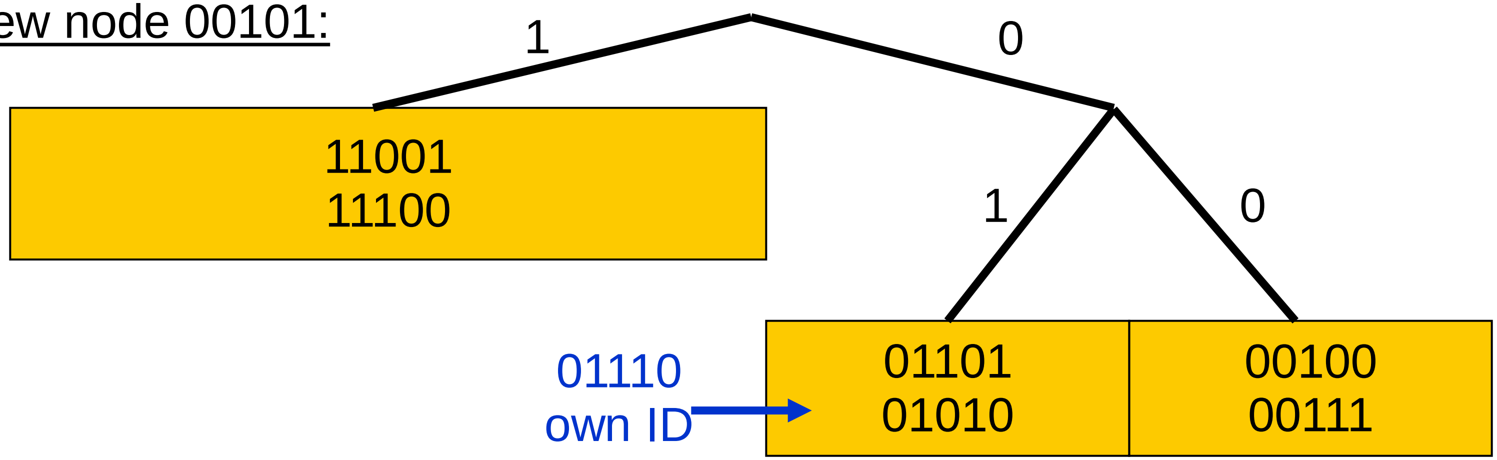
Example for Node ID 01110, k=2

11111 5-bit ID Space 00000

New node 00111:

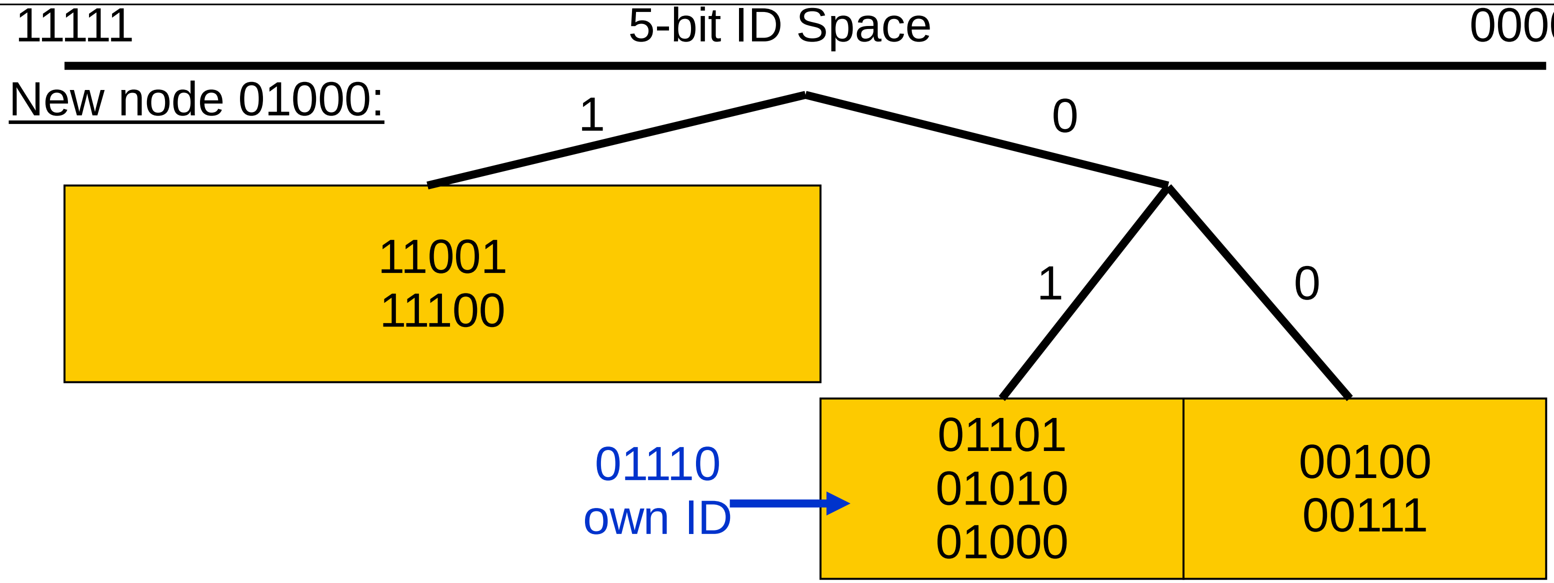


New node 00101:

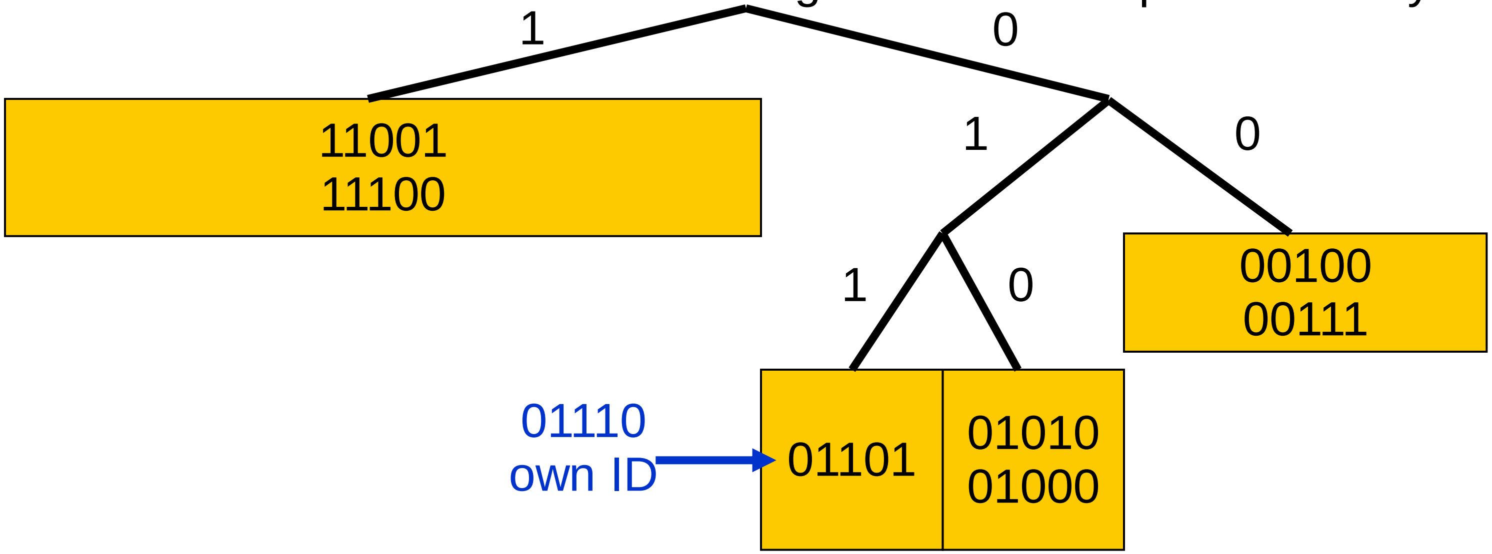


Left k-bucket full and 00101 NOT in k-bucket-range of 01110
and least recently seen node 00100 alive → node is dropped

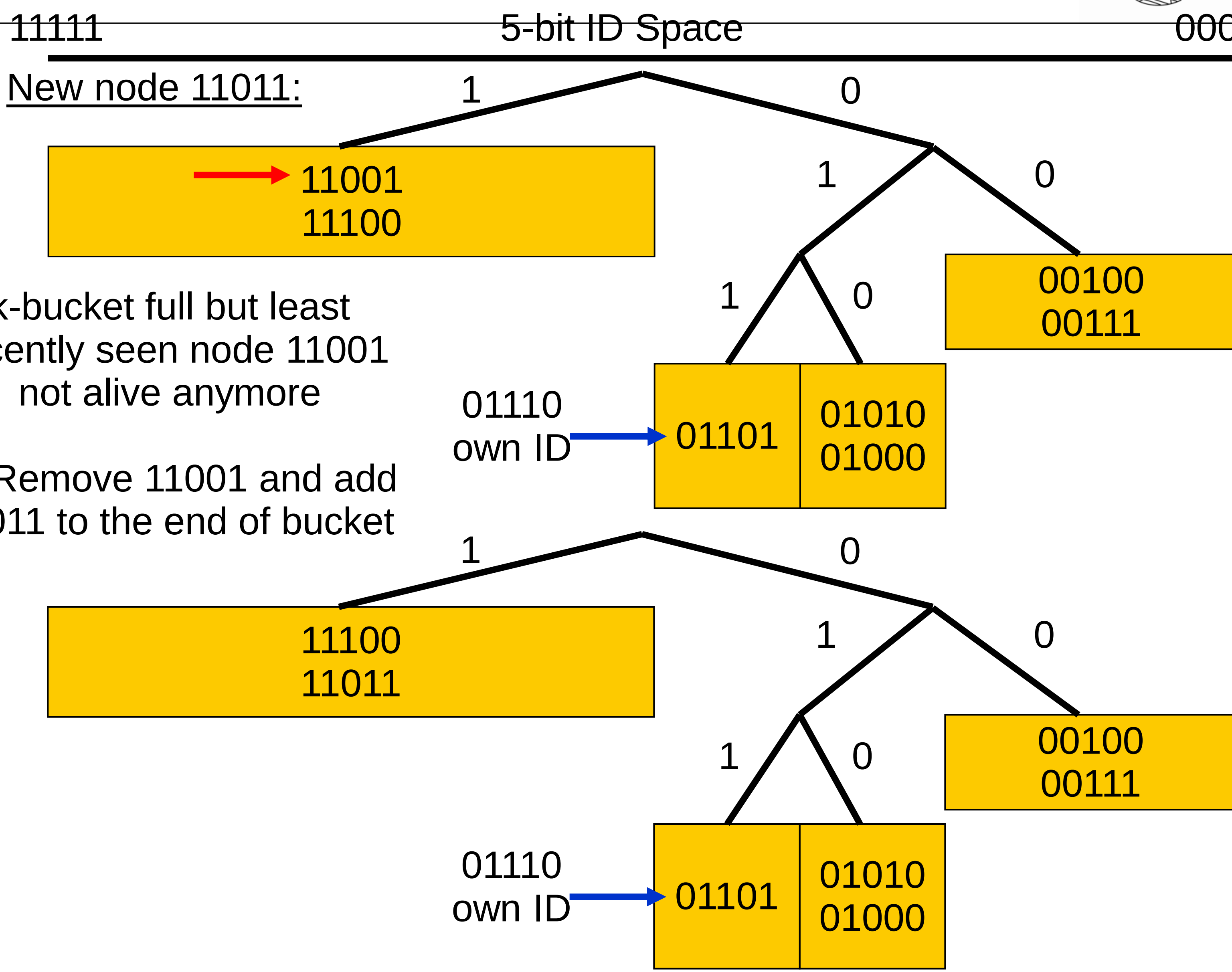
Example for Node ID 01110, k=2



k-bucket full and 01000 in k-bucket-range of 01110 → split necessary



Example for Node ID 01110, k=2



k-bucket full but least recently seen node 11001 not alive anymore

→ Remove 11001 and add 11011 to the end of bucket

k Buckets: Peer Selection Policy

**If node u receives a message from node v
then it adds node v to its k-bucket
according to the following rules:**

- IF v is already in a k-bucket
  THEN move v to the tail of the bucket

- IF v is not in the k-bucket and the bucket has fewer than k entries
  THEN insert recipient to the tail of list

- IF the appropriate k bucket is full AND least recently seen node is alive
  THEN move least recently seen node to tail of bucket and discard node v

- IF the appropriate k bucket is full AND least recently seen node is <u>not</u> alive
  THEN remove least recently seen node from bucket and add node v at the tail

Note: approx. k = 20 in the real world

Kademlia: Enhancements

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel queries

- For one query, α (alpha) concurrent lookups are sent
- More traffic load, but lower response times

Network maintenance

- In Chord: active fixing of fingers
- In Kademlia: learning for bypassing queries
- Check if peer IDs fit better in routing table

Large routing tables

- In Chord: 1 finger per distance 2^i to 2^(i+1)
- In Kademlia: k contacts per distance 2^i to 2^(i+1)
- Increased robustness

Ensuring persistency of stored data

- The owner of a particular content republishes it every 24 hours
- Every node republishes its stored <key, value> pairs once per hour

Summary

Fundamentals of Overlay Networks

- **Challenges in Data Management and Retrieval**
- **Overlay Networks: Layer Model**
- **Overlay Networks: Structures**
- **Overlay Networks: Requirements and Design**
- **Search Mechanisms in P2P Overlays**

Unstructured P2P

- **Unstructured Centralized P2P Systems**
- **Unstructured Pure P2P Systems**
- **Unstructured Hybrid P2P Systems**

Structured Overlay Networks: DHT Systems

- **Principles of Distributed Indexing**
- **Principles of DHT**
- **Examples**
  - **Chord**
  - **CAN**
  - **Kademlia**