

## Advanced Distributed Systems

---

Karl M. Göschka  
Karl.Goeschka@tuwien.ac.at

<http://www.infosys.tuwien.ac.at/teaching/courses/AdvancedDistributedSystems/>

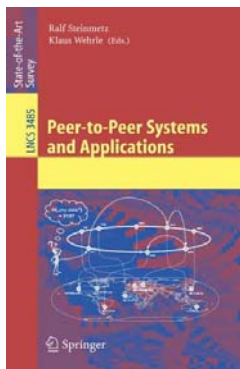
## Peer to peer systems

---

- What is P2P?
- Unstructured P2P Systems
- Structured P2P Systems

## Reference

---



- Many of the slides in this lecture are based on this book as well as accompanying material provided by the authors of this book.
- <http://www.peer-to-peer.info/>
- Online available at Springer <http://www.springerlink.com/>

## Reference

---



- Some of the slides in this lecture are based on this book as well as accompanying material provided by the authors of this book.

## What is P2P?

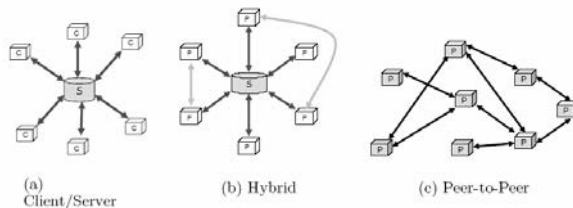
- [a Peer-to-Peer system is] a
- self-organizing system of
- equal, autonomous entities (peers) [which] aims for
- the shared usage of distributed resources in a networked environment
- avoiding central services.
- In short, it is a system with completely decentralized
  - self-organization and
  - resource usage.

## Dezentralized resource usage

- Bandwidth, storage, processing power, ...
- at the edges („end to end argument“, Saltzer, 1984)
- mutual utilization
- global distribution
- transient, dynamic connectivity
- data addressed by itself, bot by location (content-based routing)

## Dezentralized self-organization

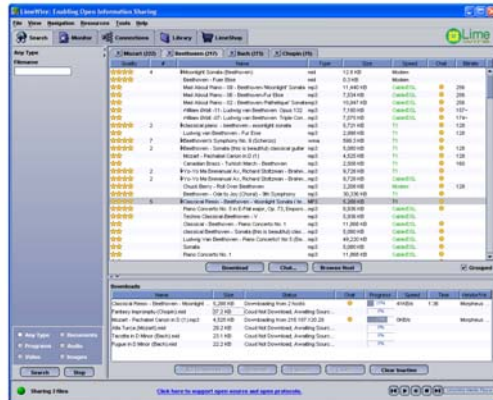
- direct interaction for resource utilization
- equal („servent“)
- autonomous
- decentralized control
- decentralized localisation



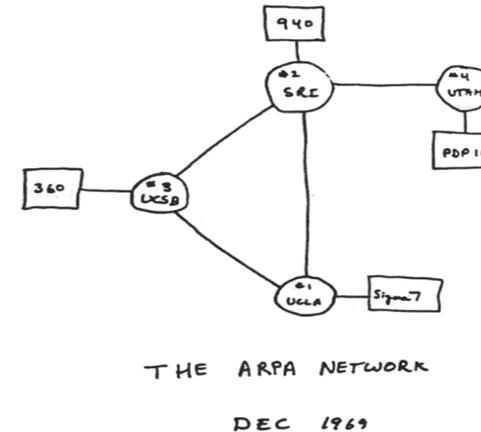
## Why P2P - from research point of view

- Challenges for future Internet applications
  - Scalability (nodes, users)
  - Dynamics (mobility, QoS-aware flexibility)
  - Heterogeneity (scopes of control)
  - Security (anonymity, censorship, availability)
  - Dependability (and performance)
- „Dependability Gap“
- Centralized systems
  - single point of failure
  - single point to attack
  - bottleneck

## Why P2P - Reality



## How it all began – ARPANET



## From ARPANET to P2P

- ❑ Goal: share computing resources and documents between research facilities
- ❑ Every host treated equally
- ❑ No overlay: the virtual network matched the physical network to a large extent
- ❑ No self-organization: Central steering committee to organize the network
- ❑ Applications: FTP and TelNet → client server mode, with no decentralized search and storage

## From ARPANET to P2P

- ❑ ~1990 rush of the general public to join the Internet (WWW)
- ❑ Still Client/server (Web-Server/Browser)
- ❑ Based on modem connections via the SLIP and PPP protocol (still slow)
- ❑ Straightforward model to administrate and control the content distribution
- ❑ Security concerns resulted in a partitioned Internet by firewalls

## Driving forces

- Development of terminal capabilities (edge)
- Development of communication networks
- Demand to circumvent copyright restrictions
- Resource sharing
  - computing power (and storage)
  - scientific computing (e.g., simulations)
  - global scale virtualization
  - similar principles, different applications
  - convergence?

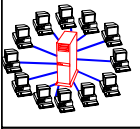
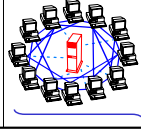
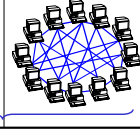
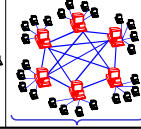
## Application Areas

- Information
  - Presence (instant messaging, resource sharing)
  - Document management
  - Collaboration and groupware (shared spaces)
- Files
  - Media (up to 70% of network traffic in the Internet can be attributed to the exchange of files)
  - Updates for software, games, anti-virus
- Bandwidth
  - Load balancing (streaming, video on demand)
  - Shared bandwidth (parts → BitTorrent)
- Storage and backup
- Computing power → GRID (SETI@home)

## Peer to peer systems

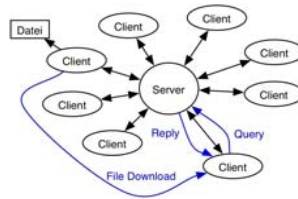
- What is P2P?
- Unstructured P2P Systems
- Structured P2P Systems

## Architectures of 1<sup>st</sup> and 2<sup>nd</sup> Gen. P2P

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	Unstructured P2P			Structured P2P	
	Centralized P2P	Pure P2P	Hybrid P2P	DHT-Based	
	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 Example: Gnutella 0.6, JXTA	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	
					
	1 <sup>st</sup> Gen.		2 <sup>nd</sup> Gen.		

## Centralized: Napster

- ❑ May 1999 – July 2001
- ❑ Today a commercial service
- ❑ concept still used: Audiogalaxy, WinMX, BitTorrent, ...
- ❑ star overlay
- ❑ HTTP-based content download
- ❑ centralized: efficient in terms of signalling effort



## Napster: Discussion

- ❑ Disadvantages
  - Single Point of Failure and Attack
  - Bottleneck, scalability
  - Central server in control of all peers (censorship)
- ❑ Advantages
  - Fast and complete lookup (one hop lookup)
  - Central managing/trust authority
  - No keep alive necessary, beyond content updates
  - Fuzzy query is possible
  - Best principle for small and simple applications

## Pure P2P: Gnutella

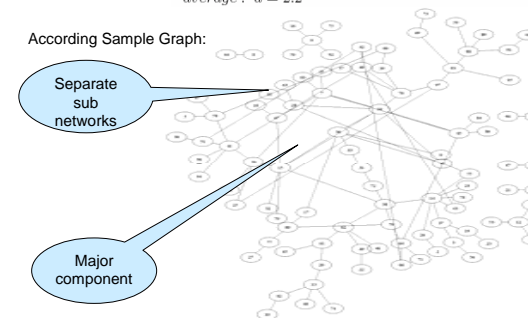
- ❑ March 2000, open source, Gnutella 0.4
- ❑ → FastTrack/KaZaA (encrypted)
- ❑ → Freenet (anonymous remote storage)
- ❑ → Edonkey
- ❑ → BitTorrent
- ❑ Any terminal entity can be removed without loss of functionality
- ❑ No central entities employed in the overlay
- ❑ Peers establish connections between each other randomly

## Model of Pure P2P Networks

Degree distribution: 
$$p(d) = \begin{cases} c \cdot d^{-1.4}, & 0 < d \leq 7 \\ 0, & \text{in any other case} \end{cases}, \text{ with } c = \left( \sum_d \frac{p(d)}{c} \right)^{-1}$$

average:  $\bar{d} = 2.2$

According Sample Graph:



## Bootstrapping

---

- ❑ Necessary to know at least one active participant of the network
- ❑ Bootstrap cache: Try to establish one after another a connection to a node seen in a previous session
- ❑ Bootstrap server: “well known host”, node-addresses which recently used this bootstrap → centralized
- ❑ Broadcast on the IP layer
  - Use multicast channels
  - Use IP broadcasting (-limited to local network)

## Five steps

---

- ❑ Connect to at least one active peer (address received from bootstrap)
- ❑ Explore your neighborhood (PING/PONG)
- ❑ Submit Query with a list of keywords to your neighbors (they forward it)
- ❑ Select “best” of correct answers (which we receive after a while)
- ❑ Connect to providing host/peer

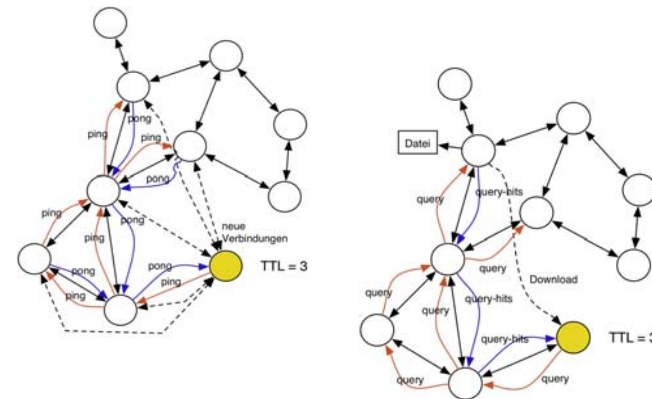
## Routing

---

- ❑ Completely decentralized
- ❑ Reactive protocol: routes to content providers are only established on demand, no content announcements
- ❑ Requests: **flooding** (limited by TTL and GUID)
- ❑ Responses: routed (Backward routing with help of GUID)
- ❑ Keep-alive
- ❑ Content-search and fuzzy search

## Neighborhood and Query

---



## Gnutella: Discussion

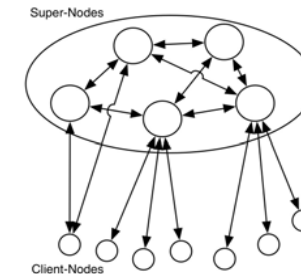
- Disadvantages
  - High signaling traffic (flooding)
  - Modern nodes may become bottlenecks
  - Overlay topology not optimal, as
    - no complete view available,
    - no coordinator
  - If not adapted to physical structure → delay and total network load increase
    - Zigzag routes
    - loops
- Advantages
  - No single point of failure or attack
  - Can be adapted to physical network
  - Can provide anonymity
  - Can be adapted to special interest groups
- Application areas
  - File-sharing
  - Context based routing (fuzzy search)

## How it works

- Upon connection to the network via a Superpeer, each node is a leafnode
- It announces its shared content to the Superpeer it connected to
- Superpeer thus updates its routing tables
- Election mechanism decides which node becomes a Superpeer or a leafnode (depending on capabilities (storage, processing power) network connection, the uptime of a node,...), if
  - Too many nodes are connected to one Superpeer
  - A Superpeer leaves the network
  - Too less nodes are connected to a Superpeer

## Hybrid approaches

- Introduction of another dynamic hierarchical layer
- Hub based network
- Reduces the signaling load without reducing the reliability
- Election process to select and assign **Superpeers**
- Superpeers: high degree (degree >> 20, depending on network size)
- Leafnodes: connected to one or more Superpeers (degree < 7)
- → FastTrack/KaZaA, Gnutella-2 (Gnutella 0.6), Edonkey

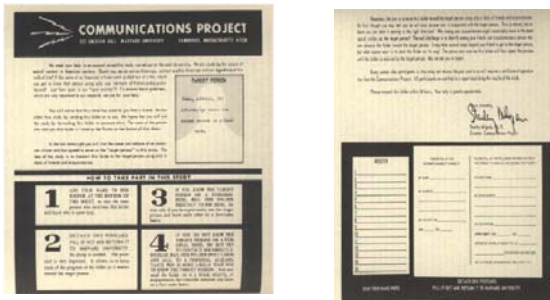


## Discussion

- Disadvantages
  - Still High signaling traffic, because of decentralization
  - No definitive statement possible if content is not available or not found
  - Modern nodes may become bottlenecks
  - Overlay topology not optimal, as
    - no complete view available,
    - no coordinator
  - If not adapted to physical structure → delay and total network load increases
    - Zigzag routes
    - Loops
  - Can not be adapted to physical network completely because of hub structure
  - Asymmetric load (Superpeers have to bear a significantly higher load)
- Advantages
  - No single point of failure
  - Can provide anonymity
  - Can be adapted to special interest groups
- Application areas
  - File-sharing
  - Context based routing (fuzzy search)

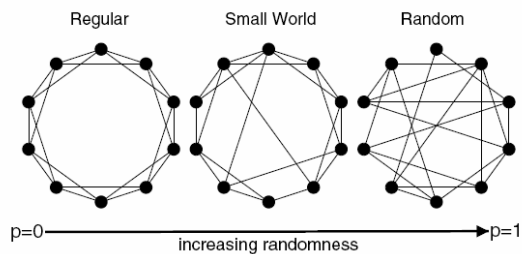
## The Riddle: Milgram's Experiment

- Analysis of real networks



## Watts-Strogatz Models

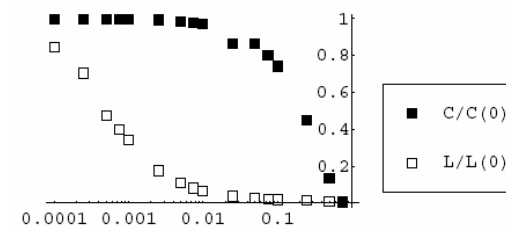
- This can neither be explained by a grid-like network model nor by a full random network:
  - **Grid-like** network show regularity and locality, but have a **high average path length** (and diameter).
  - **Random graphs** have a small clustering coefficient.



## Watts-Strogatz Models

- Watts and Strogatz built on this result. They measured two things in real-world networks:
  - The clustering coefficient as a measure for the 'regularity', or 'locality' of the network.
  - The average path length between vertices.
- If the clustering coefficient is high, the average path length should also be high
- Finding:
  - **Most real-world networks have a high clustering coefficient (0.3-0.4) AND a low average path length.**

## Watts-Strogatz Models



- Watts and Strogatz
  - showed that and how they can be combined
  - did **not explain** how this happens in real networks
  - did **not explain** how people can find short paths

## More Experimental Findings

---

- The small-world network model and random graph models suggest that the degree of the vertices will not deviate much from the average degree in the graph:
- Barabasi et al. crawled a small proportion of the Web. The degree distribution was power-law distributed, i.e.,  $P(k) \sim k^{-\gamma}$  (**Scale-free network**).
- This means that most vertices have a small degree (1-2) but on the other hand, high-degree vertices are much more probable than in a normal distribution.
- High-degree vertices are called '**hubs**'.
- What kind of network model can generate this degree distribution?

## Navigability Model by Kleinberg

---

- Every member only 'sees' a local part of the network, namely its own neighbors, i.e., they route the letter in a decentralized fashion.
- The routing algorithm can be modelled as follows:
  - Let every node  $v$  have a position  $Pos(v)$  on a lattice in a  $d$ -dimensional space.
  - Every node knows the position of itself, its neighbors, and the target node.
  - Give the message (i.e., letter) to the one neighbor  $v$  that is next to the target  $t$ . (Manhattan Distance).

## Barabási-Albert Model

---

1. Start with a small network (e.g., 10 vertices, 20 edges, at random).
2. **Consider growth**: In every time step, add a new node  $x$ . Add  $m$  edges from  $x$  to the nodes  $v$  that were already there, where the edge generation between a new node and old nodes is not at random, but follows the principle: "**The rich get richer**", i.e., the higher the degree of a possible target node, the higher the probability that the new vertex will attach to it (**Preferential attachment**).

## Navigability Model by Kleinberg

---

- Let the routing algorithm take place on the following network model:
  - Start with a  $d$ -dimensional **grid**
  - Add **random edges** between nodes  $v$  and  $w$  with a probability

$$P(v, w) \sim d_M(v, w)^{-\alpha}$$

- → The routing algorithm will find 'short' paths, i.e., paths with a length of  $O(\log n)$ , if and only if  $\alpha = d$ .

## Navigability Model by Kleinberg

- The idea behind the proof is that for an  $\alpha < d$ , there are **too little random** edges to **make the paths short**.
- For  $\alpha > d$ , there are **too many random** edges, and hence **too many choices** to which the message could be passed on. The message will make a (long) random walk through the network.

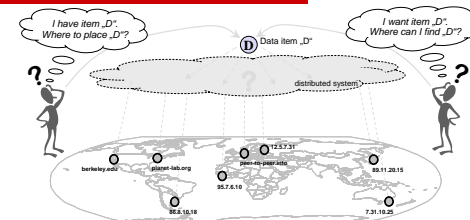
## Peer to peer systems

- What is P2P?
- Unstructured P2P Systems
- Structured P2P Systems

## Discussion

- The network structure of a peer-to-peer system influences:
  - the average path length,
  - the possibility of greedy, decentralized routing algorithms,
  - the stability against random failures,
  - the sensitivity against attacks,
  - the redundancy of edges,
- Important measures of a network structure are:
  - The average path length,
  - the clustering coefficient,
- It is necessary to influence the edge generation rules such that a network structure arises that supports the wanted properties of the system.

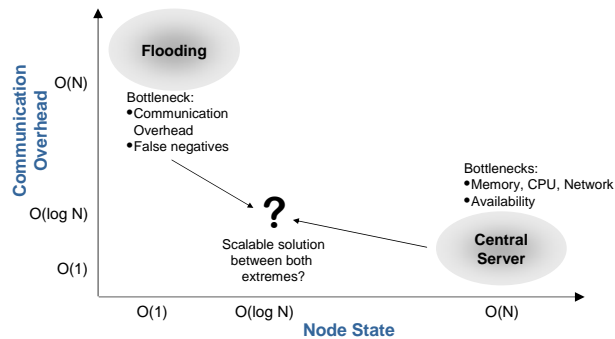
## Management and Retrieval of Data



- **Essential challenge** in (most) Peer-to-Peer systems
  - **Location of a data item**
    - Where shall the item be stored by the provider?
    - How does a requester find the actual location of an item?
  - **Scalability**: keep the complexity for communication and storage scalable
  - **Robustness** and **resilience** in case of faults and frequent changes

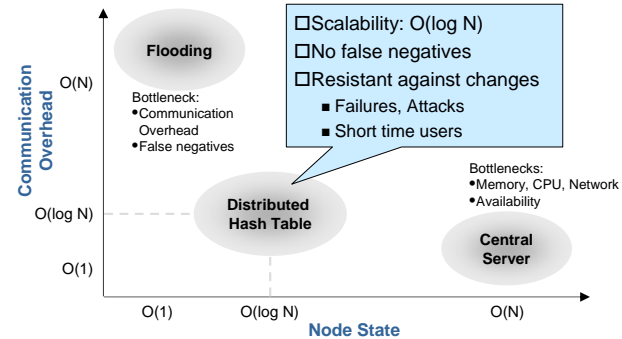
## Motivation Distributed Indexing (1)

### Communication overhead vs. node state



## Motivation Distributed Indexing (2)

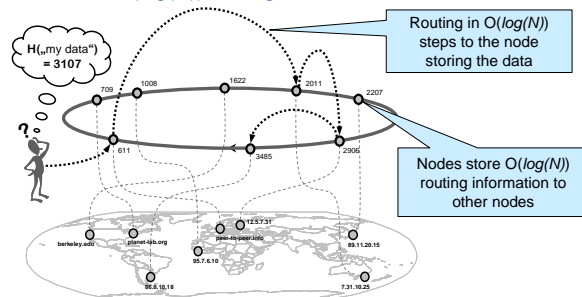
### Communication overhead vs. node state



## Distributed Indexing

### Goal is scalable complexity for

- Communication effort:  $O(\log(N))$  hops
- Node state:  $O(\log(N))$  routing entries



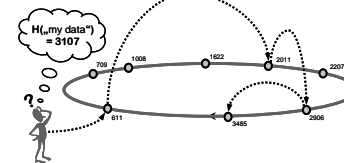
## Distributed Indexing

### Approach of distributed indexing schemes

- Data and nodes are mapped into same address space
- Intermediate nodes maintain routing information to target nodes
  - Efficient forwarding to „destination“ (content – not location)
  - Definitive statement of existence of content

### Problems

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

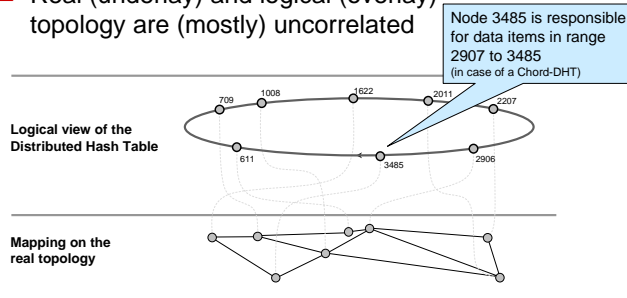


## Comparison of Lookup Concepts

System	Per Node State	Communication Overhead	Fuzzy Queries	No false negatives	Robustness
Central Server	$O(N)$	$O(1)$	✓	✓	✗
Flooding Search	$O(1)$	$O(N^2)$	✓	✗	✓
Distributed Hash Tables	$O(\log N)$	$O(\log N)$	✗	✓	✓

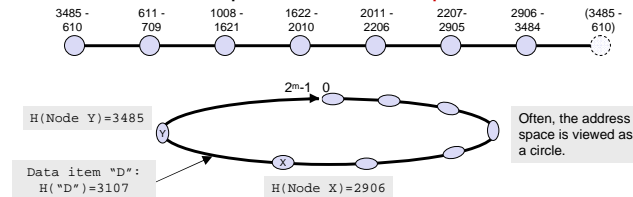
## Association Address Space – Nodes

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



## Addressing in Distributed Hash Tables

- Step 1: Mapping of content/nodes into linear space
  - Usually:  $0, \dots, 2^m - 1 \gg$  number of objects to be stored
  - Mapping of data and nodes into an address space (with hash function)
    - E.g.,  $\text{Hash}(\text{String}) \bmod 2^m: H(\text{„my data“}) \rightarrow 2313$
  - Association of parts of address space to DHT nodes



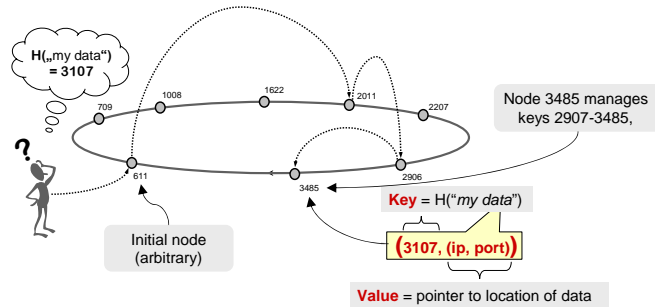
## Step 2: Routing to a Data Item

- Step 2:
  - Locating the data (content-based routing)
- Goal: Small and scalable effort
  - $O(1)$  with centralized hash table
    - But: Management of a centralized hash table is very 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 ( $N = \#$  nodes)

## Step 2: Routing to a Data Item

### Routing to a K/V-pair

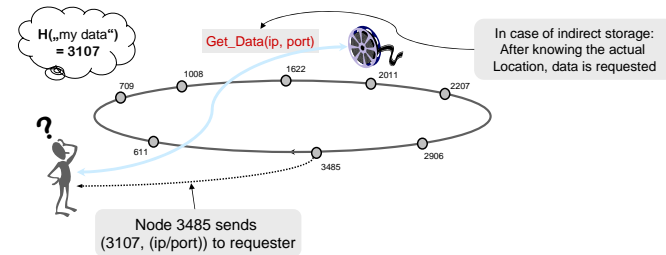
- Start lookup at arbitrary node of DHT
- Routing to requested data item (key)



## Step 2: Routing to a Data Item

### Getting the content

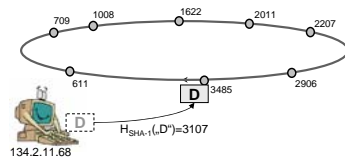
- K/V-pair is delivered to requester
- Requester analyzes K/V-tuple (and downloads data from actual location – in case of indirect storage)



## Direct Storage

### Content is stored in responsible node ("my data")

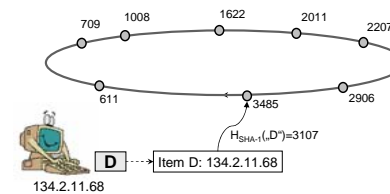
- Inflexible for large content – o.k., if small amount data (<1KB)
- Anonymity and censorship?



## Indirect Storage

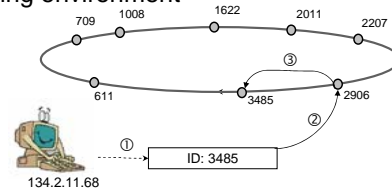
### Indirect storage

- Nodes in a DHT store tuples like (key,value)
  - Key =  $\text{Hash}(\text{"my data"}) \rightarrow 2313$
  - Value is often real storage address of content: (IP, Port) = (134.2.11.140, 4711)
- More flexible, but one step more to reach content



## Node Arrival

- Joining 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 K/V-pairs of hash range
  5. Binding into routing environment

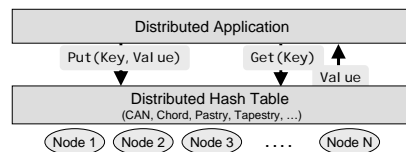


## Node Failure / Departure

- Failure of a node
  - Use of redundant K/V 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 K/V pairs to corresponding nodes
  - Unbinding from routing environment

## DHT Interfaces

- Generic interface of distributed hash tables
  - Provisioning of information `Publish(key,value)`
  - Requesting of information (search for content) `Lookup(key)`
  - Reply `value`
- DHT approaches are interchangeable (with respect to interface)



## Comparison: DHT vs. DNS

### Domain Name System

- Mapping: Symbolic name → IP address
- Is built on a hierarchical structure with root servers
- Names refer to administrative domains
- Specialized to search for computer names and services

### Distributed Hash Table

- Mapping: key → value can easily realize DNS
- Does not need a special server
- Does not require special name space
- Can find data that are independently located of computers

## Discussion

---

- Use of routing information for efficient search for content
- Keys are evenly distributed across nodes of DHT
  - No bottlenecks
  - A continuous increase in number of stored keys is admissible
  - Failure of nodes can be tolerated
  - Survival of attacks possible
- Self-organizing system
- Simple and efficient realization

## Summary

---

- P2P systems are supposed to solve the problems of centralized systems
- In reality, most applications are file-sharing
- However, other applications seem reasonable and possible (e.g., GRID)
- Basic approaches are structured and unstructured