P2P-Networks

David Koll

Advanced Computer Networks

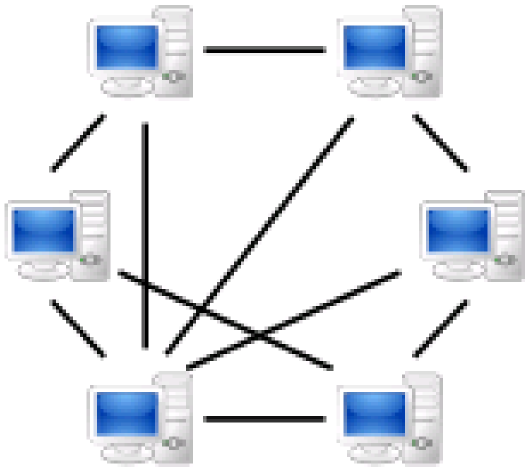
Summer Semester 2013

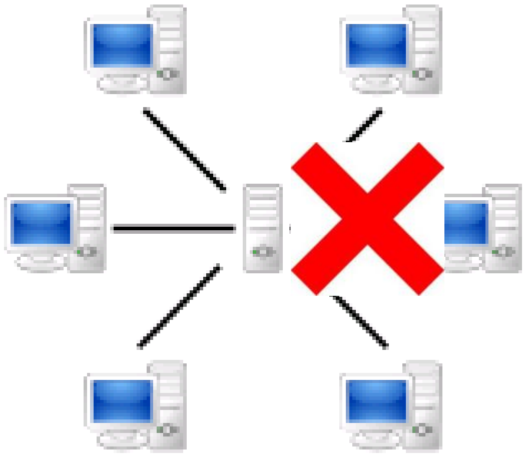
Introduction to P2P Systems

- **What?**
  - What is a P2P system?
- **Why?**
  - Why do we need P2P systems?
- **How?**
  - How are P2P systems working?

What is a P2P System?

- Peer-to-peer (abbreviated to P2P) refers to a computer network in which each computer in the network can act as a client or server for the other computers in the network, allowing shared access to files and peripherals without the need for a central server [Wiki]

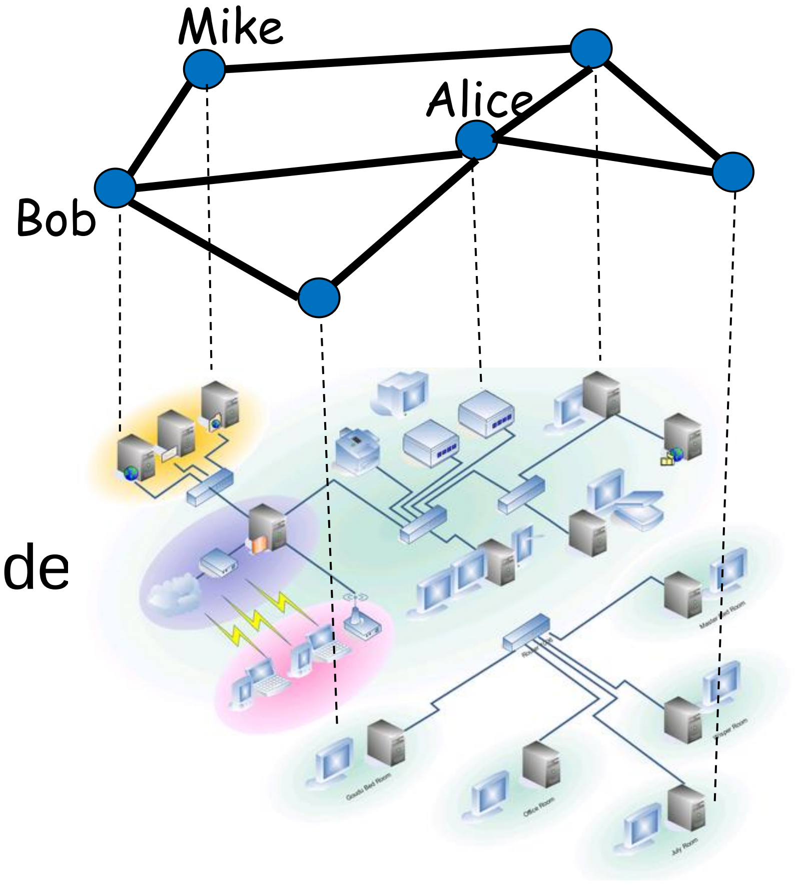




- The sharing of computer resources by direct exchange, rather than requiring the intermediation of a centralized server.

Features

- Decentralized:
  - No central component
- Role: “all peers are equal”
- Self-organized
  - Highly dynamic behavior of node
    - Free to come, free to go



- Overlay Network
  - A network built on the top of physical network
  - Nodes are connected by logical links
  - Flat system architecture

Features (Cont.)

- Large-scale resources
  - Heterogeneous
  - Millions of nodes
- Collaboration
  - Based on voluntary participation
  - Global reach
- Flexible, resilient to attacks, anonymous
- …

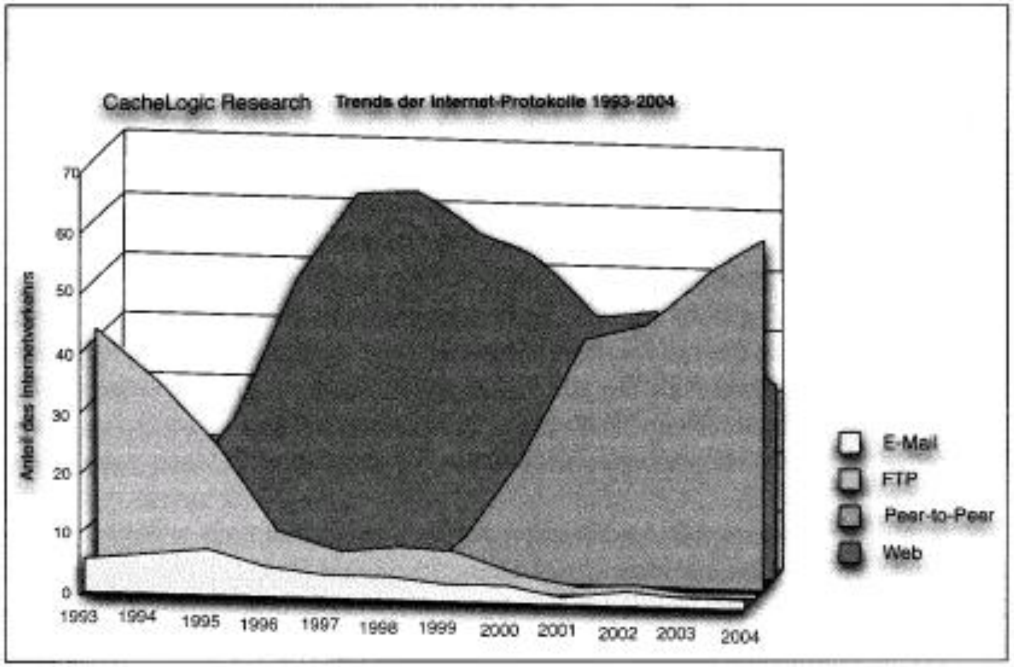
Why P2P?

User Requirement of Resource Sharing

- Text, email: 100 KB
- Picture: 1MB
- Music: 10MB
- Movie: 1GB
- HD Video: 10GB+

User Requirement of Resource Sharing

- 2004: More than half of Internet traffic P2P
- 2010: Still 39%, 2014: ~20%
- Absolute P2P traffic increases (by 100% 2010-2014)



Developing of Network Nandwidth

- Maximum bandwidth of common Internet access technologies [Wiki]

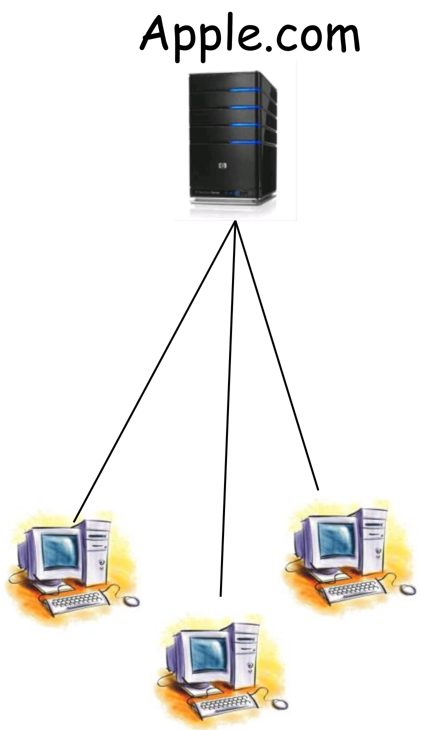
- Current network bandwidth cannot satisfy user demand

- Need a more efficient way to share resources

56 kbit/s	Modem / Dialup
1.5 Mbit/s	ADSL Lite
1.544 Mbit/s	T1/DS1
10 Mbit/s	Ethernet
11 Mbit/s	Wireless 802.11b
44.736 Mbit/s	T3/DS3
54 Mbit/s	Wireless 802.11g
100 Mbit/s	Fast Ethernet
155 Mbit/s	OC3
600 Mbit/s	Wireless 802.11n
622 Mbit/s	OC12
1 Gbit/s	Gigabit Ethernet
2.5 Gbit/s	OC48
9.6 Gbit/s	OC192
10 Gbit/s	10 Gigabit Ethernet
100 Gbit/s	100 Gigabit Ethernet

Client/Server

- The client arrives and requests a service at any given point in time
- The server is dedicated to the service and responds to the client

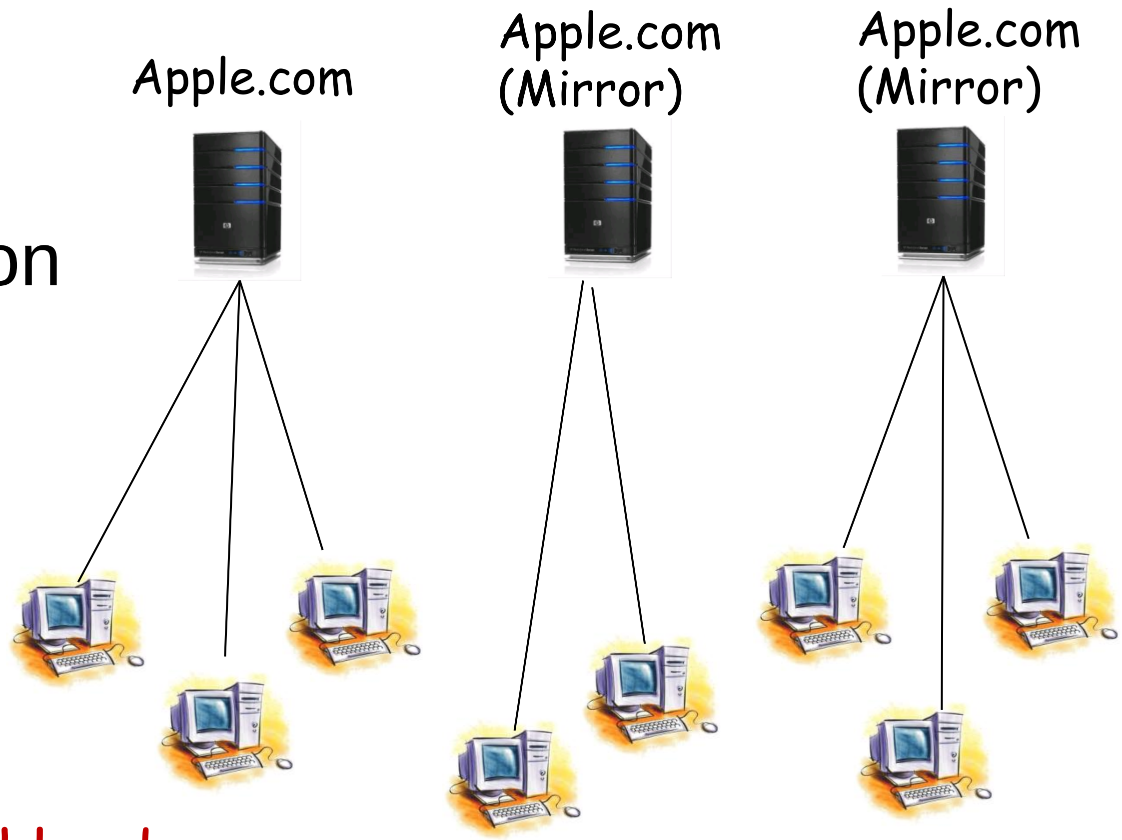


Problems

- Hot spot-uneven workload
- Bottleneck: bandwidth, CPU, …
- Single point of failure
- Scalability
- ~~Maintenance~~

Replication

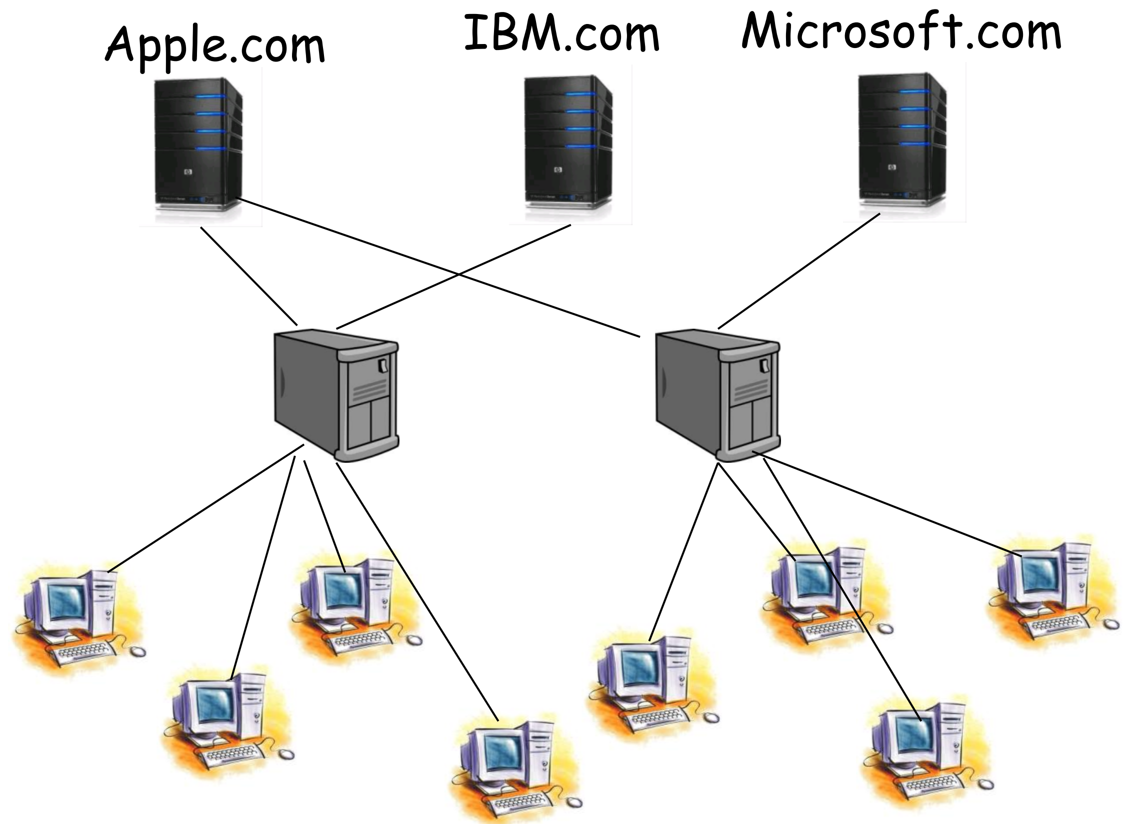
- Server Replication



Problems

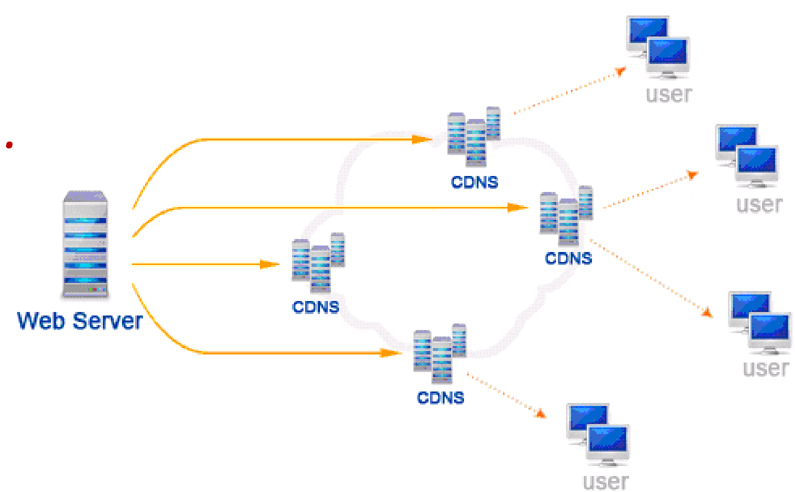
- Hot spot-uneven workload
- Bottleneck: bandwidth, CPU, …
- ~~Single point of failure~~
- ~~Scalability~~
- Maintenance

Proxy, CDN



Problems

- ~~Hot spot uneven workload~~
- Bottleneck: bandwidth, CPU, …
- ~~Single point of failure~~
- ~~Scalability~~
- Maintenance



P2P: Advantages

- Changes the way of network bandwidth use
- Easy to deploy, easy to use
- Dynamic for joining and leaving
- Distributed resource sharing
  - Files, data, storage, computation, …
  - Provide something useful and free
  - Anyone can contribute
- Fault tolerant
- Service ability: large scale
- Service of quality: the more users, the better

P2P: How?

P2P Applications and Systems

- File sharing
  - Napster, Gnutella, BitTorrent
- Multimedia streaming
  - P2P TV: PeerCast, PPlive, PPStream, TVUZattoo, ….
  - P2P based VOD systems
- Communication
  - Skype, MSN, …
- Computation
  - SETI@home: Search for Extra-Terrestrial Intelligence

Current State of P2P

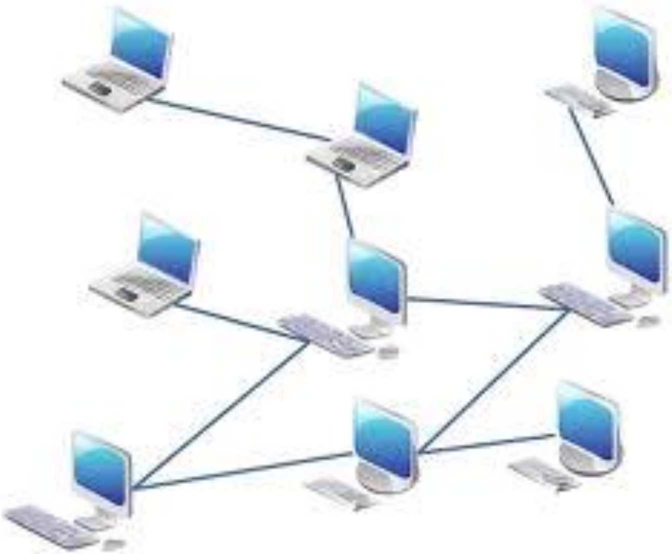
- P2P applications are popular over the world
- P2P networks are mainly used for resource sharing
  - Music, videos, software, …
  - Some are illegal copyrighted materials
- New emerging applications
  - Online media streaming, P2P TV
  - P2P telephone system
  - Software installation and update
  - Decentralized social network applications

Typical Research Topics

- Structure
  - How to search for information
    - Unstructured P2P
    - Structured P2P
- Security and privacy
  - How to protect system security and user privacy?
- Legal issues

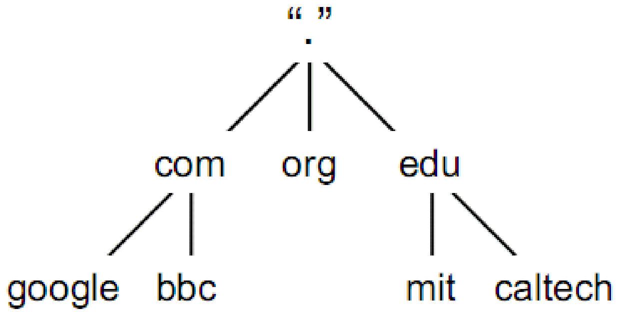
Search in P2P Networks

- How to locate resources in P2P networks?



DNS- Domain Name System

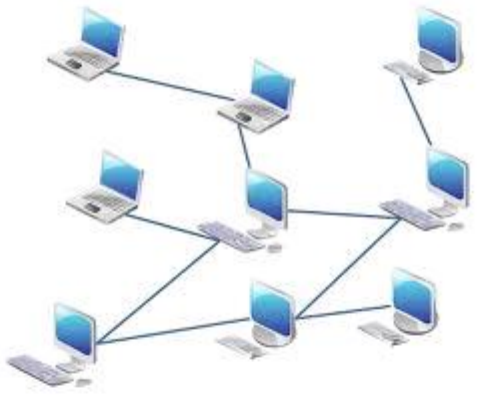
- Translates queries for domain names into IP addresses for the purpose of locating computer services and devices worldwide
- Distributed database organized in hierarchical structure



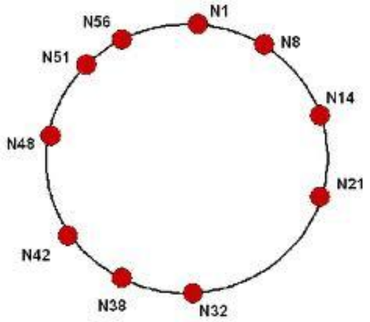
Information Retrieval System

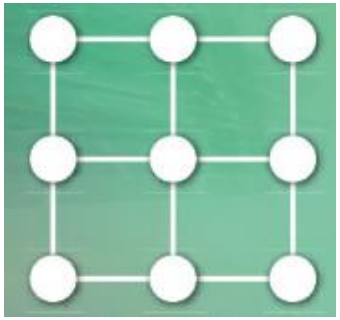
- Keyword-based

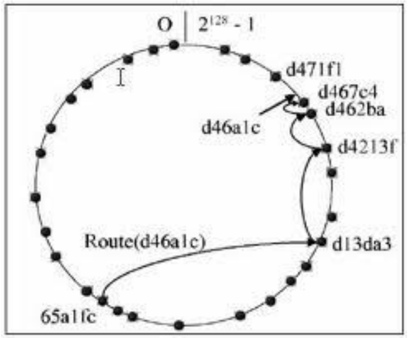
Search in P2P Networks?



- Unstructured P2P
  - Highly flexible, dynamic, easy to maintain
  - Hard to find information
- Structured P2P
  - Hard to maintain its structure
  - Easy to find information







computer NETWORKS

Unstructured P2P Networks

Unstructured P2P Networks

- Example: BitTorrent

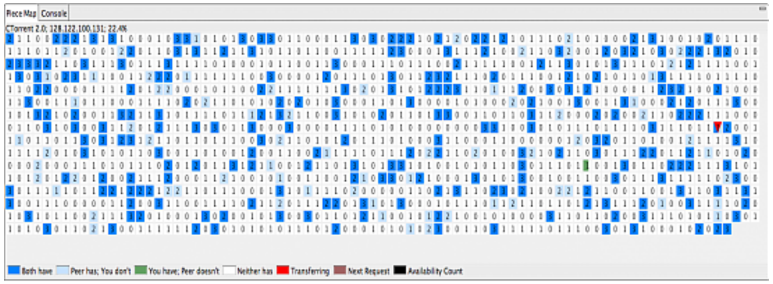
- Successor of Napster, Gnutella, …
  - Napster: Pioneer P2P, shut down in 2001
    - easy to manage and search, but relied on central lookup server (drawbacks?)
    - data transfer directly between peers
  - Gnutella: Answer to Napster weaknesses
    - fully distributed P2P network based on overlay network, no central server
    - Search: flooding the network with request (drawbacks?)

BitTorrent

- A new popular approach to sharing large files
  - Origin of 37% of German internet traffic in 2009
- Originally used for distributing legal content
  - Linux distributions, software updates
  - Official movies

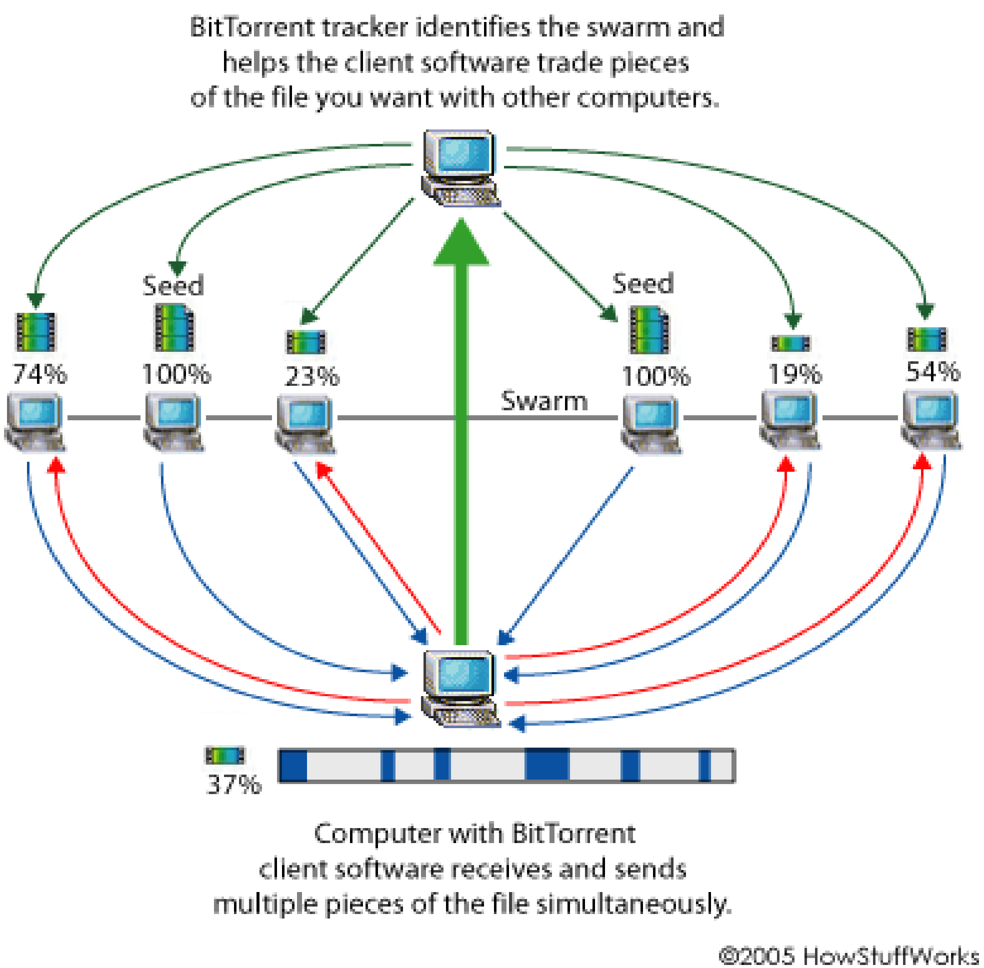
- Goal:
  - Quickly and reliably replicate one file to a large number of clients
- Call it "P2P content distribution"

Basic Idea



- Chunking:
  - Files split into smaller pieces or chunks
  - Chunks can be downloaded in parallel
  - Downloading order does not matter
- Swarming
  - Clients join a crowd of peers uploading and downloading the same content
  - Nodes request chunks from neighbors and download content in parallel
- Use a web server to publish content
- Use a central unit to locate resource

Basic Idea


BitTorrent tracker identifies the swarm and
helps the client software trade pieces
of the file you want with other computers.
Seed
Seed
74%
100%
23%
100%
19%
54%
Swarm
37%
Computer with BitTorrent
client software receives and sends
multiple pieces of the file simultaneously.
©2005 HowStuffWorks

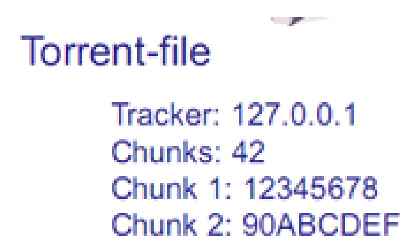
Basic Components

- Web server: for content publication

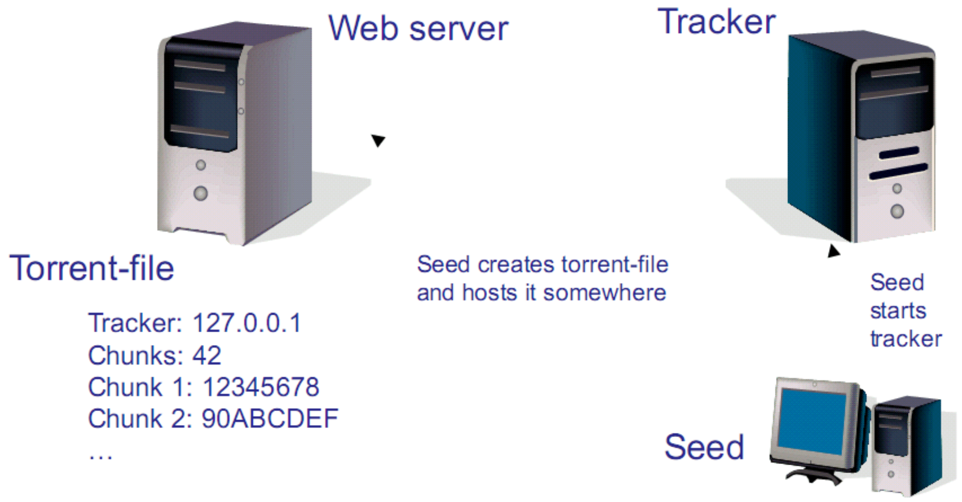
- Tracker: a special central server for running the content distribution system
  - Tracking active peers
  - Mapping from file name to peers

- Peer
  - Seed: a peer with a complete copy of the file
  - Leecher: peer still downloading the file

- ".torrent" file: metadata and description of the file
  - The number of chunks
  - The tracker's IP



Operation



- Sharing a file:
  - (1) Seed generates a ".torrent" file from the file
  - (2) Upload the ".torrent" file to some public web server or sending it to friends by email
- Searching a file:
  - No dedicate search component
  - User can search ".torrent" file from web server
- Downloading a file:
  - (1) Download the ".torrent" file
  - (2) Connect to the tracker to locate the file
  - (3) Choose some fast peers to download chunks in parallel

Tit-for-Tat Policy and Chunk Selection

- Tit-for-Tat policy
  - The more you give, the more you get
  - A peer serves peers that serve it
  - Encourages cooperation, discourage free-riding
- Chunk selection
  - Peers uses rarest first policy when downloading chunks
  - Having a rare chunk makes peer attractive to others
  - The goal is to maximize availability of each chunk

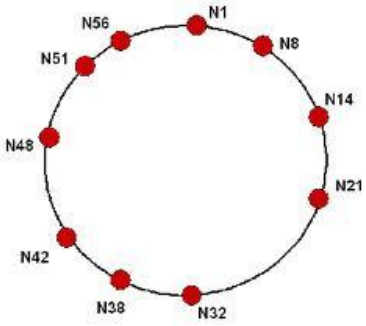
BitTorrent : Pros and Cons

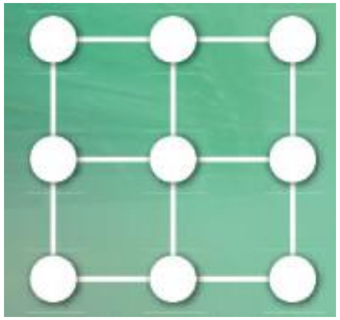
- Strengths
  - Works well for “hot content”, very fast and resilient
  - Proficient in utilizing partially downloaded files
  - Discourages “free-riding”
  - Efficient for distributing large files to a large number of clients
- Weaknesses
  - Assumes all interested peers active at same time
  - Tracker could be single point of failure
  - Lack of search feature

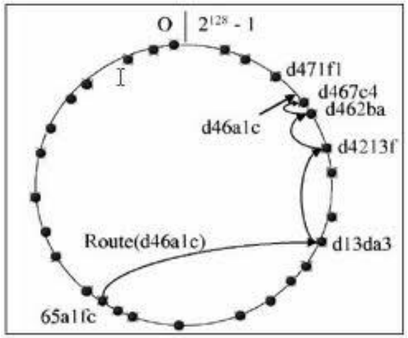
Structured P2P Networks

Structured P2P Networks

- Routing & Lookup
- DHTs







- The following slides are based on a lecture by Prof. Roscoe, ETH Zürich, and provided with his kind permission

Problem Space

- Challenge: spread lookup database among P2P participants
- Goals:
  - Scalable – operates with millions of nodes
  - Self-organized – no central, external control
  - Load-distributing – every member should contribute (at least ideally)
  - Fault tolerant – robust against node leaves or failures
  - Robustness – resiliance against malicious activity

Idea

- Distributed Hash Tables
  - Hash content identifiers to machines
  - Hash IP addresses
  - Store content (or content locator) at machine with closest hash value
- Originally 4 papers submitted to SIGCOMM 2001:
  - CAN, Chord, Pastry, Tapestry
- Widely used in practice (e.g., BitTorrent uses Kademlia DHT)

Background: Hash Functions

- Hash function maps arbitrary input sequence to fixed length output:
  - H(m) = x, x of fixed length
- Crypto-Hashes:
  - Small input changes result in large output changes (Avalanche criterium)
  - If H(m1) = x is known, it is hard to find another m2 giving H(m2) (collision resistant)
- Inheritly hash functions span whole 2^k space (k bits hash length)

computer NETWORKS

MD5 / SHA-1

- Message Digest Algorithm 5
  - 128 bit hash values
  - Weak collisions found
- SHA-1 (similar to MD4)
  - 160 bit hash values
  - Stronger than MD5, but „under researcher‘s attack“: find collisions in 2^{69}
- But: Both algorithms efficiently map input homogeniously to 2^k space

DHTs

- Index data by hash value
- Assign each node in the network a portion of the hash address space
- DHT provides the lookup function

Example: Chord

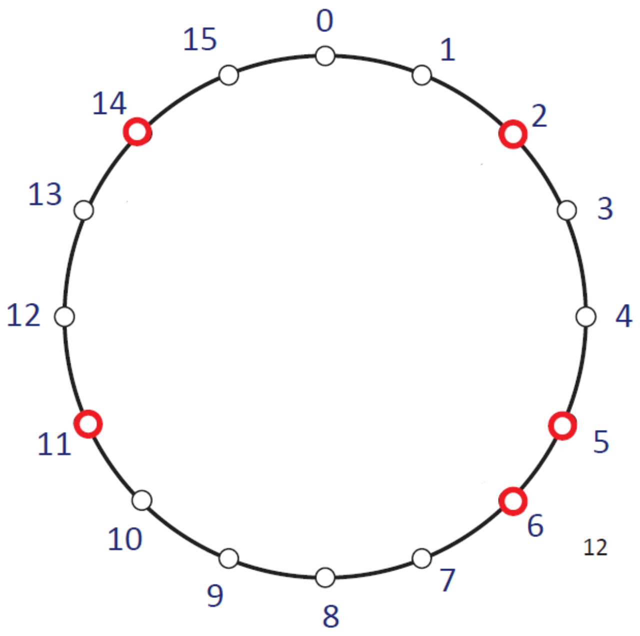
- Published 2001 at SIGCOMM by Stoica et al. „Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications"

- Keys are SHA-1 hashes – 160 bit identifiers
- Key: Identifier of a data item
- Value: Identifier of a node

- Host (key,value) pair at node with ID larger or equal to key – successor(key)

Identifier Space

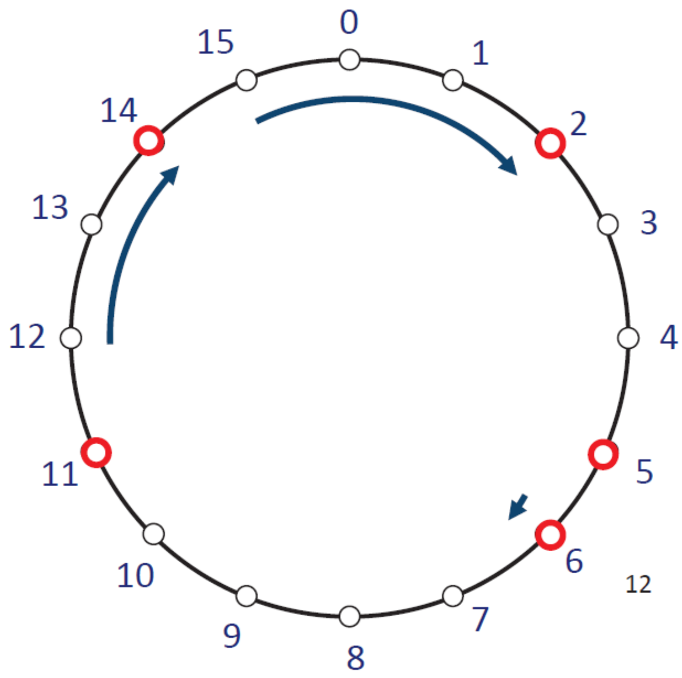
- Identifier in 2^4 space
  - Space from 0..15
  - Nodes pick IDs:
    - 2,5,6,11,14 covered by nodes
    - Remaining values are not directly covered by a node



Successor

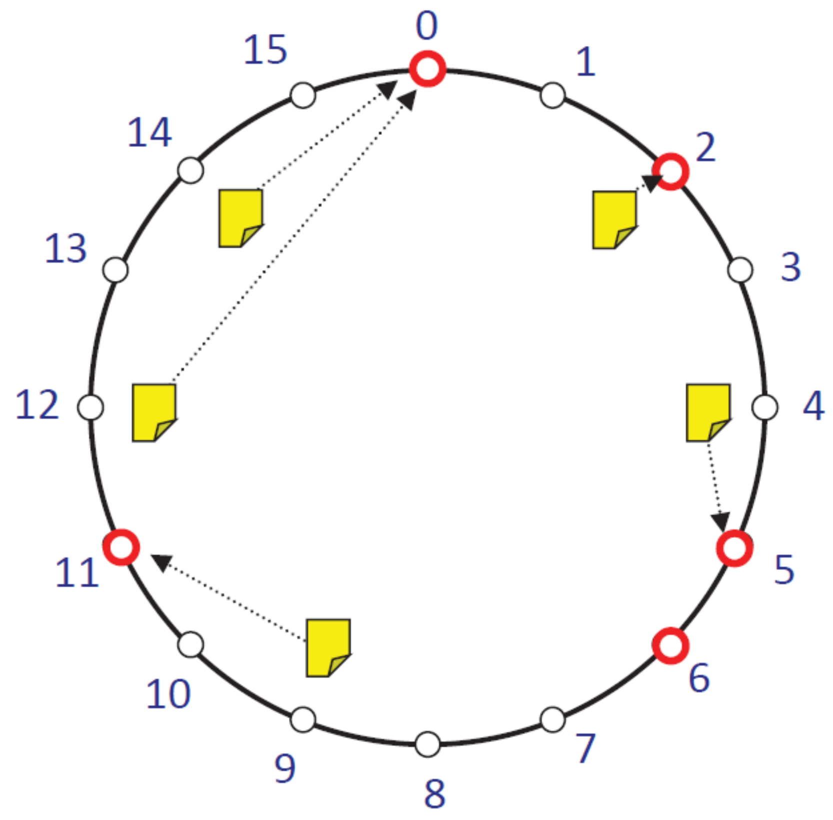
- First node in clockwise direction with ID larger or equal the key

- Examples:
  - succ(6) = 6
  - succ(12) = 14
  - succ(15) = 2



How to store and locate data?

- Each (key,value) pair is assigned the identifier H(key)
- Each item is stored at its succ(H(key))

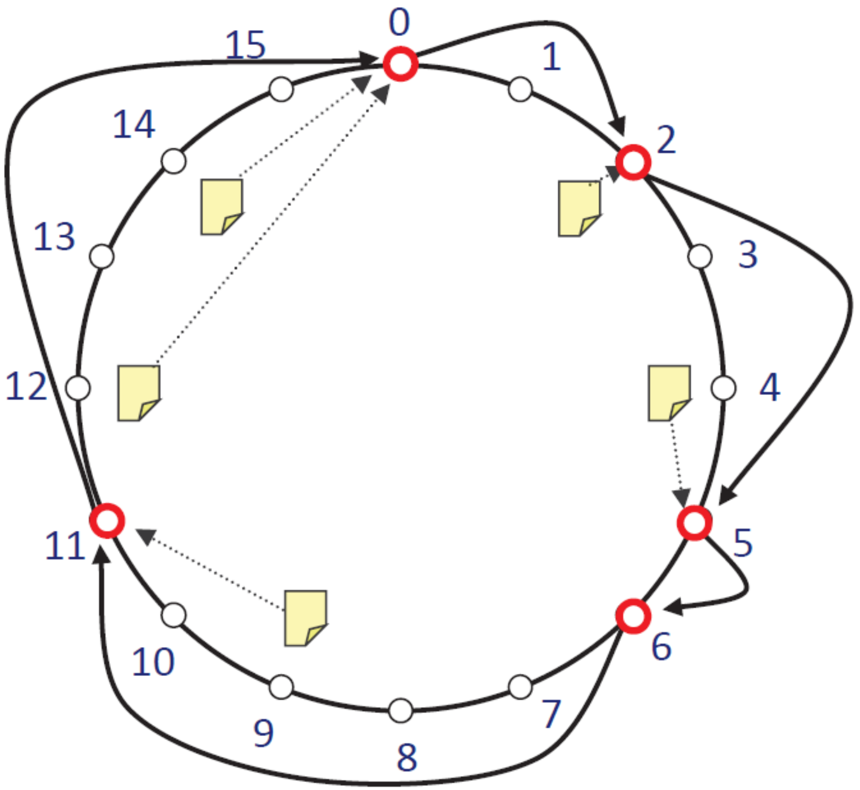


Drink	Location	H(Drink)
Beer	Göttingen	12
Wine	France	2
Whisky	Scotland	9
Wodka	Russia	14

Successor Pointer

- Each node points to its successor
  - Known as node‘s succ pointer

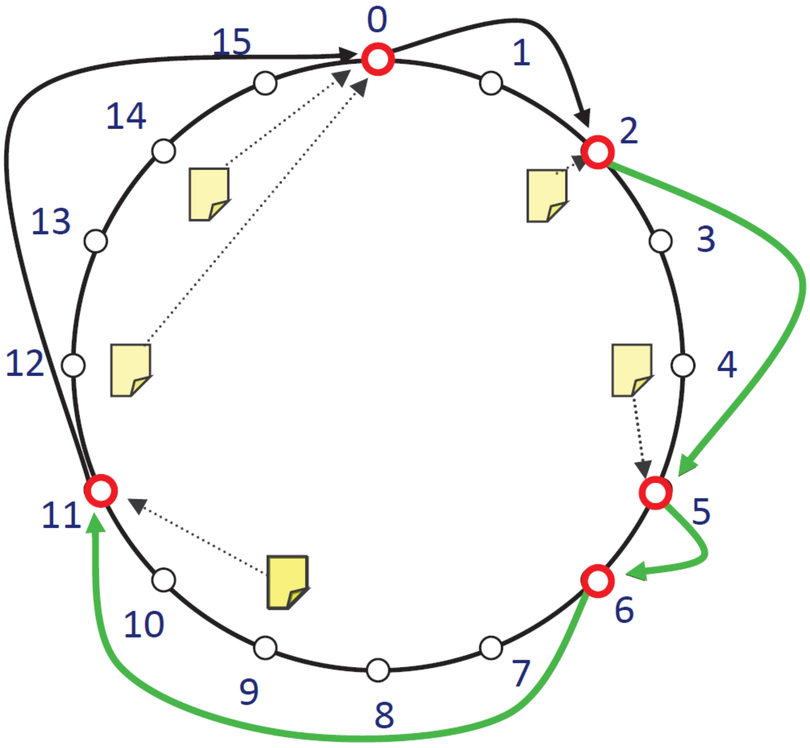
- Example:
  - 0‘s succ = 2
  - 2‘s succ = 5
  - ...



Basic Lookup of Data

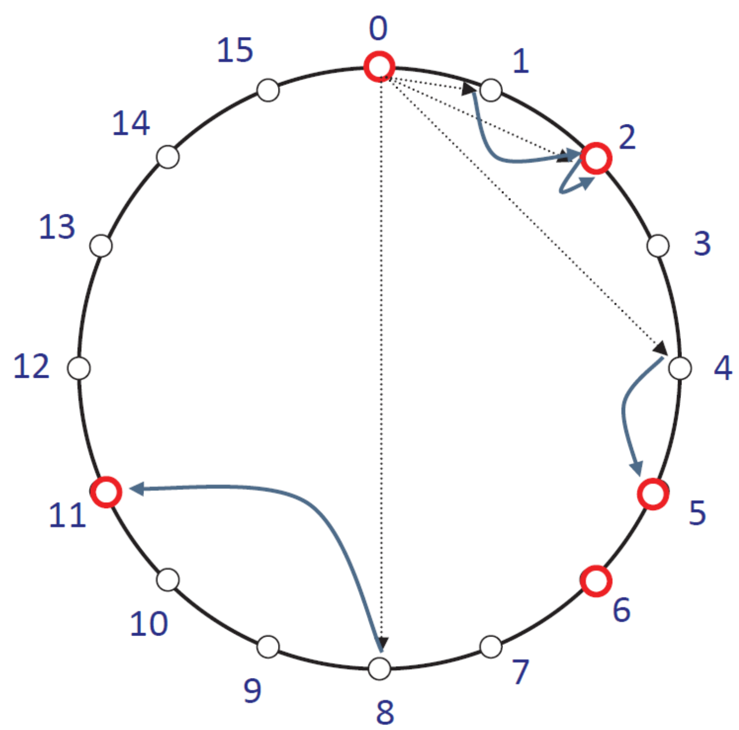
- Lookup key:
  - Calculate H(key)
  - Follow succ pointers until key is found
  - Lookup time: O(n)

- Example:
  - „Where can I drink Whisky?“
  - Calculate H(Whisky) = 9
  - Traverse nodes:
    - 2,5,6,11
  - Return „Scotland“



Scalable Lookup

- Each node maintains finger table (max k entries)
- for i in 0..k-1: finger[i] = succ($n+2^{i-1}$)
  - Point to succ(n+1)
  - Point to succ(n+2)
  - Point to succ(n+4)
  - ...
  - Point to succ($n+2^{i-1}$)

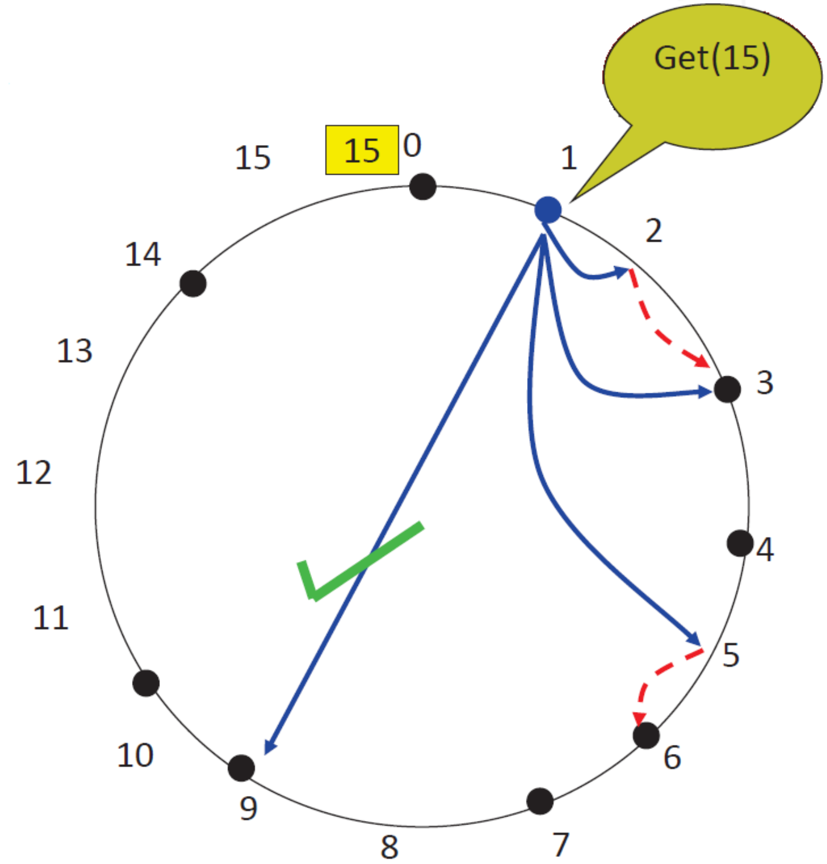


- Makes lookup time logarithmic!
  - O(log n)

Routing

- Determines the next hop
- Each node n knows $succ(n+2^{i-1})$ for all i=1..k
- Forward queries for key to then highest predecessor of key

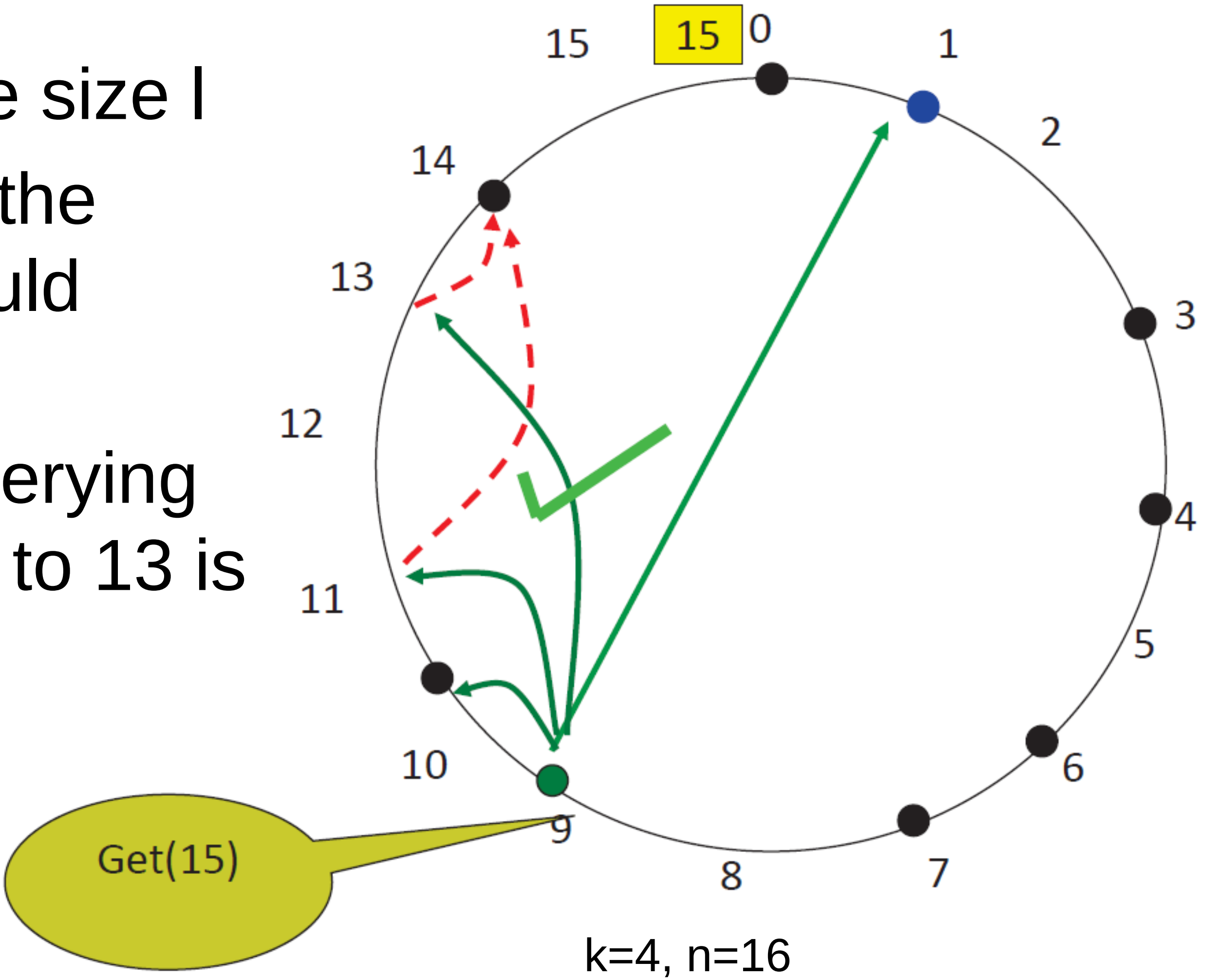
- Routing entries = $\log_2(n)$



k=4, n=16

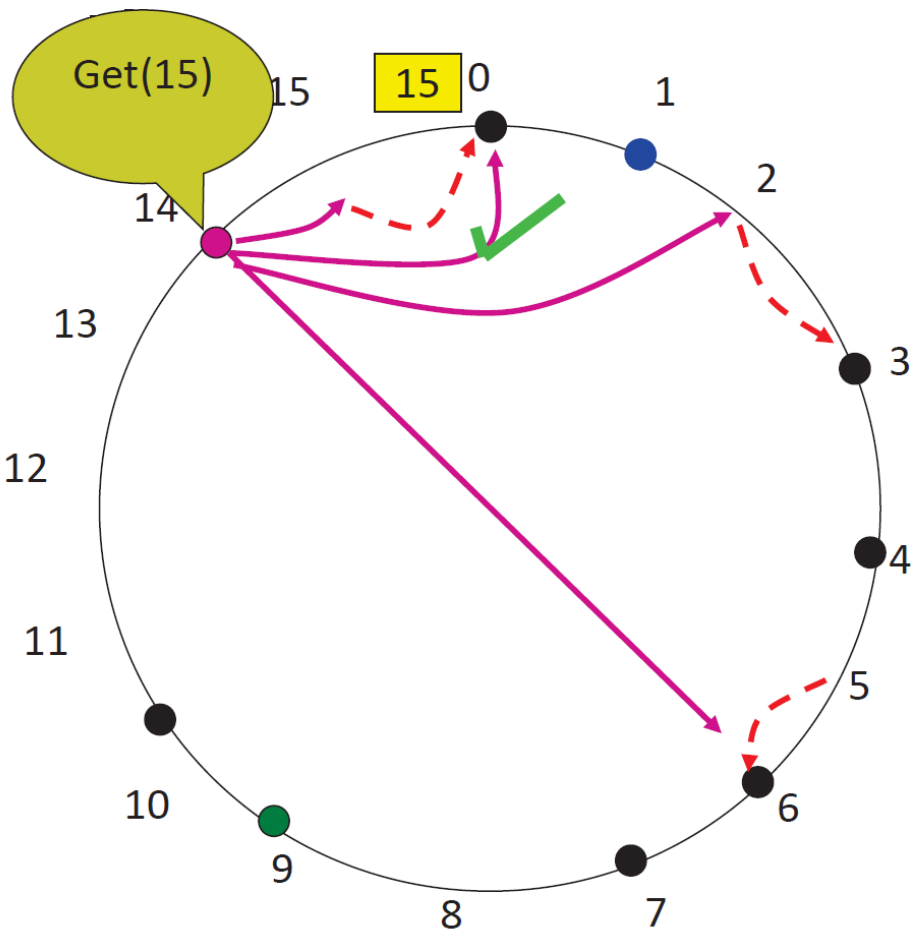
Routing cont'd

- Routing table size l
- Node 9 was the highest 1 could reach
- Node 9 is querying again, finger to 13 is best



Routing cont'd

- 13 is handled by 14
- 14 completes the route:
  - 15 is found at 0



Routing cont'd

- From node 1, 3 hops to node 0 where item 15 is stored
- k=4 equals an ID space of 16, therefore the maximum number of hops is:
  - $Log_2(16) = 4$

- Average complexity is ½ log(n)

Routing cont‘d

- Such routing algorithms solve the lookup problem
- General concept:
  - Each node has only a limited view on the network
  - A node that receives a message containing a destination ID that is not managed by that node, it just forwards the request to the closest hop
- Here, algorithm is based on numeric closeness

- In Gummadi et al., „*The Impact of DHT Routing Geometry on Resilience and Proximity*“, SIGCOMM 2003, implications are discussed

Recursive vs. Iterative Lookup

- Recursive: Each node forwards the request (as shown) to the next hop
  - Fast, efficient
  - Each node can optimize forwarding

- Iterative: The requesting client queries the next hop iteratively from the nodes
  - Allows the lookup client to keep in control
  - Lookup client detects and localizes failures

Achieved goals

- The DHT is scalable, as operations are performed in log(n)
- It is self-organized as each node directly knows its position (thanks to the hash function) and learns about the next hops
- On average load-distributing

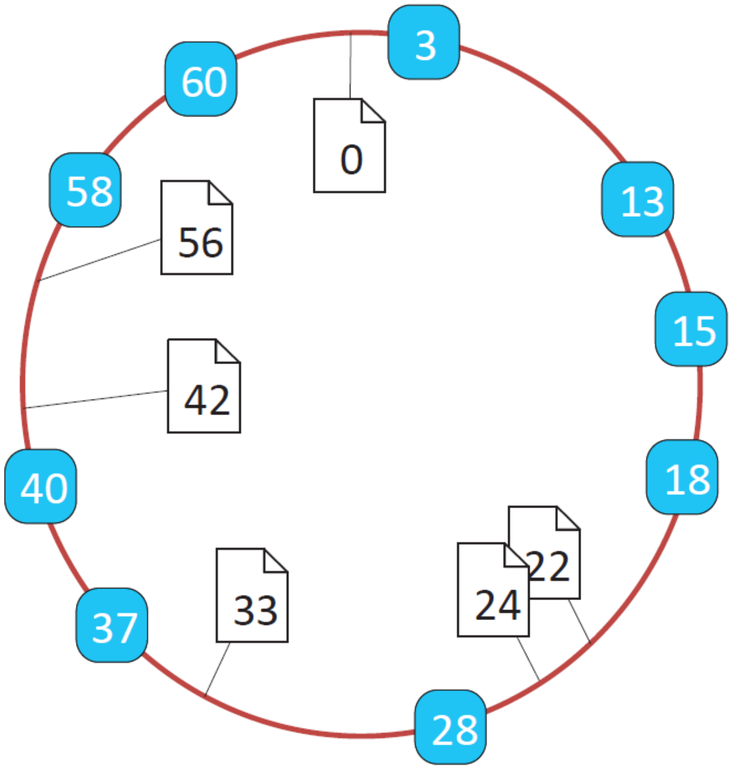
- What about joins and especially leaves?

Node Join and Leave

- Node join:
  1. Bootstrap: a new node contacts a known node in the DHT
  2. The new node gets a partion of the address space
  3. Routing information is updated
  4. The new node retrieves all tuples for which it is responsible
- Node departure:
  - Replication and load balancing
- Node failure:
  - Reactive or proactive recovery
  - Maintenance, load balancing, redistribution of tuples
  - Data is lost if not replicated!

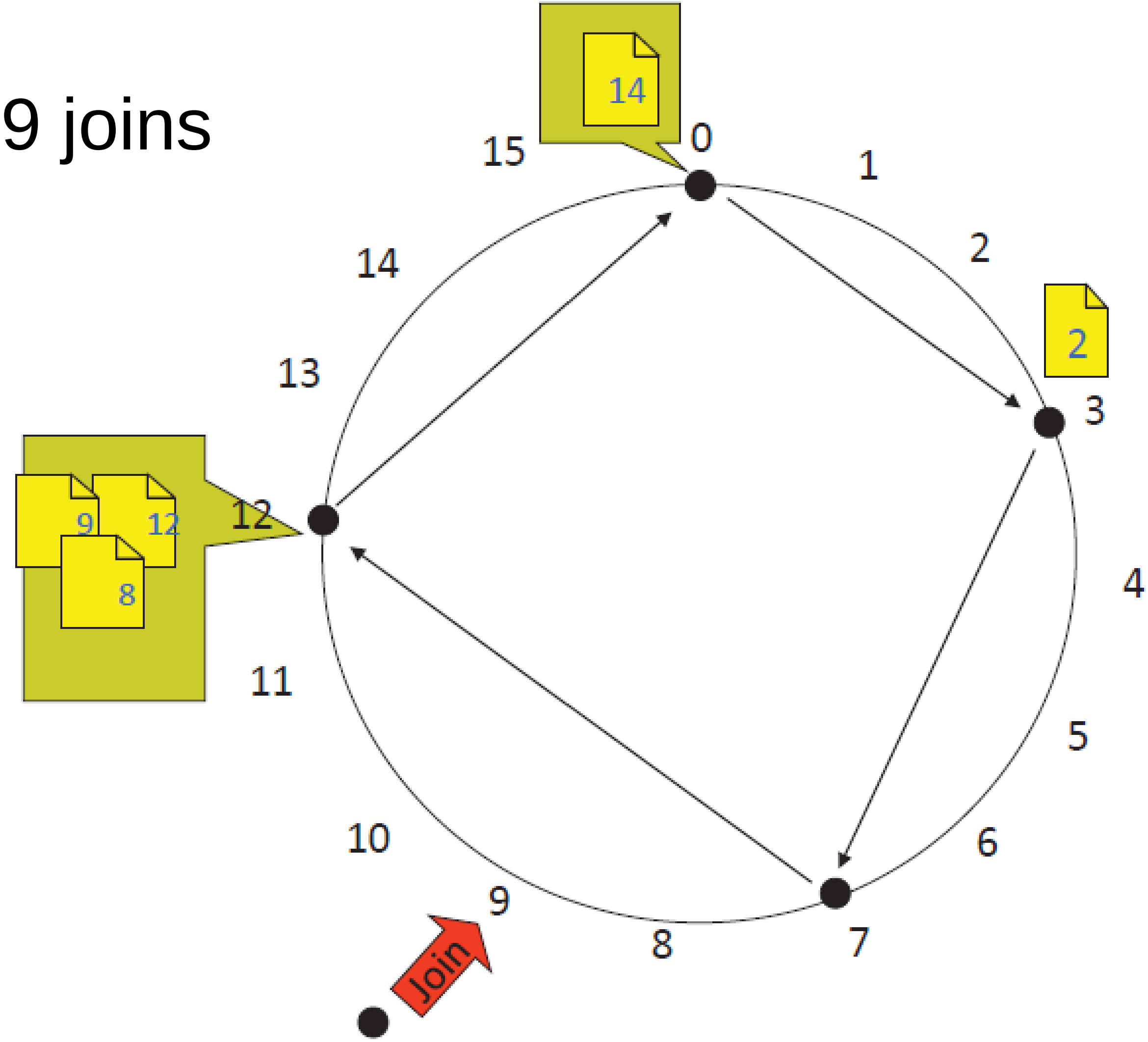
Node Join and Leave

- Join:
  - Lookup of own ID's successor
  - Contact that to get successors and predecessor
- Leaves:
  - Ping successors regularly
  - Always ensure x live nodes in successor set
- Thereby, failures are treated as „normal"



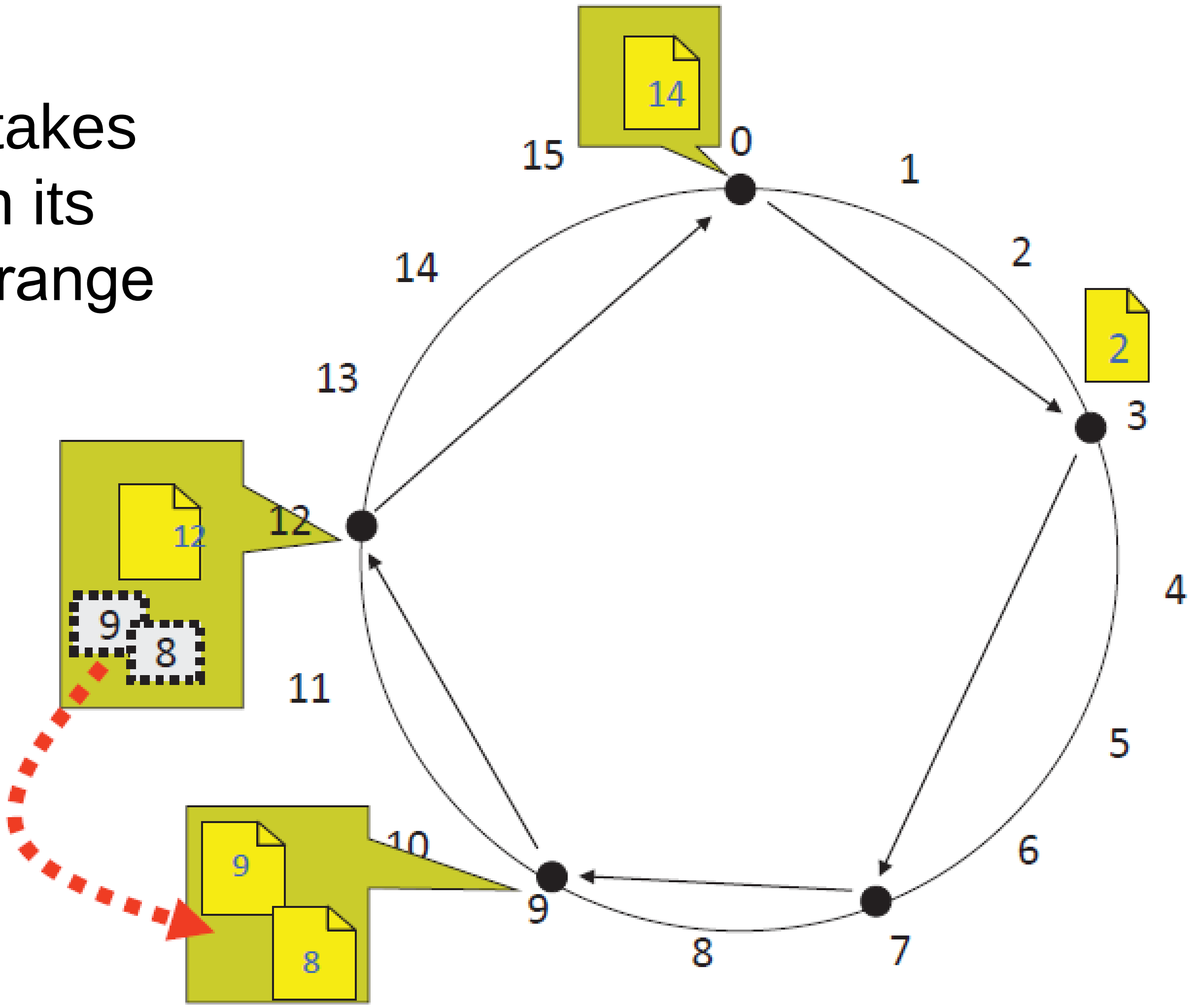
Node Join Example

- Assume node 9 joins



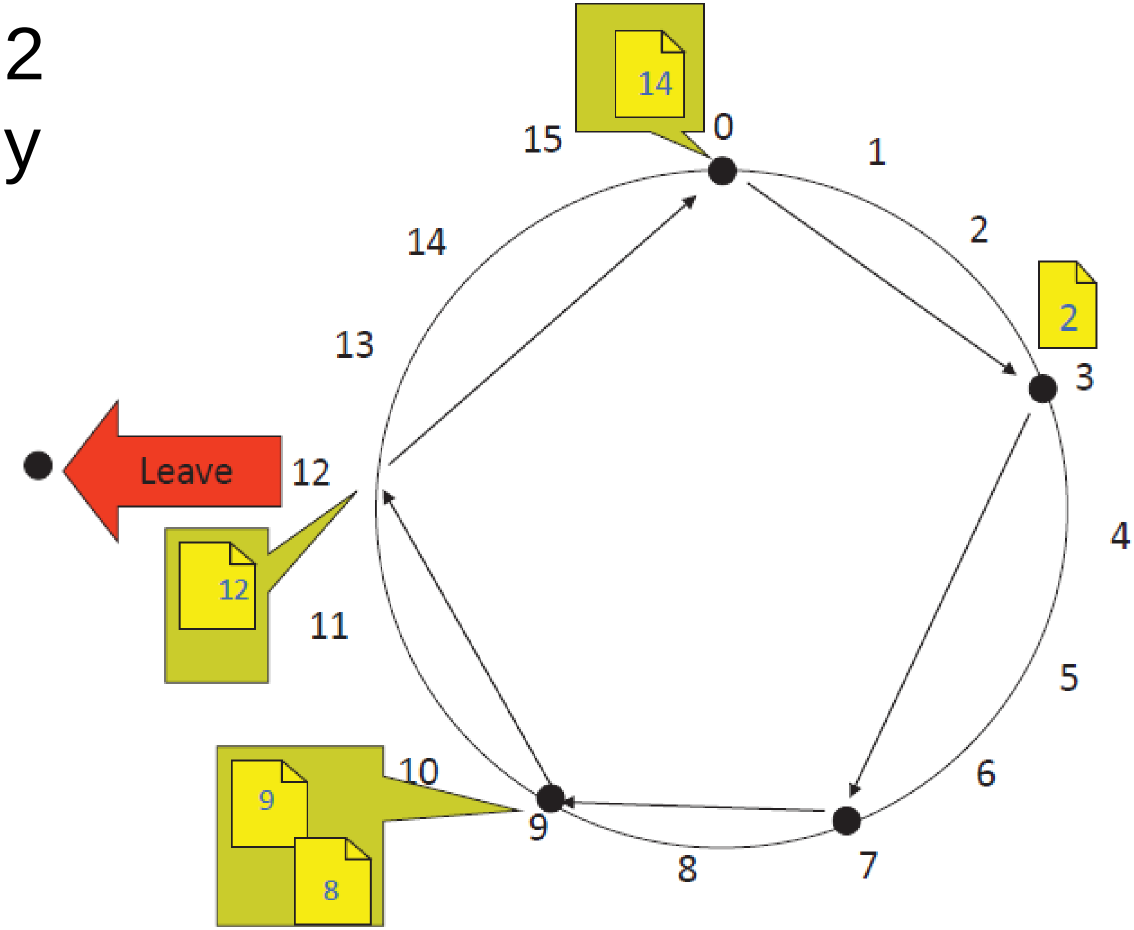
Node Join Example cont'd

- The new node takes over the docs in its "responsibility" range
- Docs 9,8 from its successor



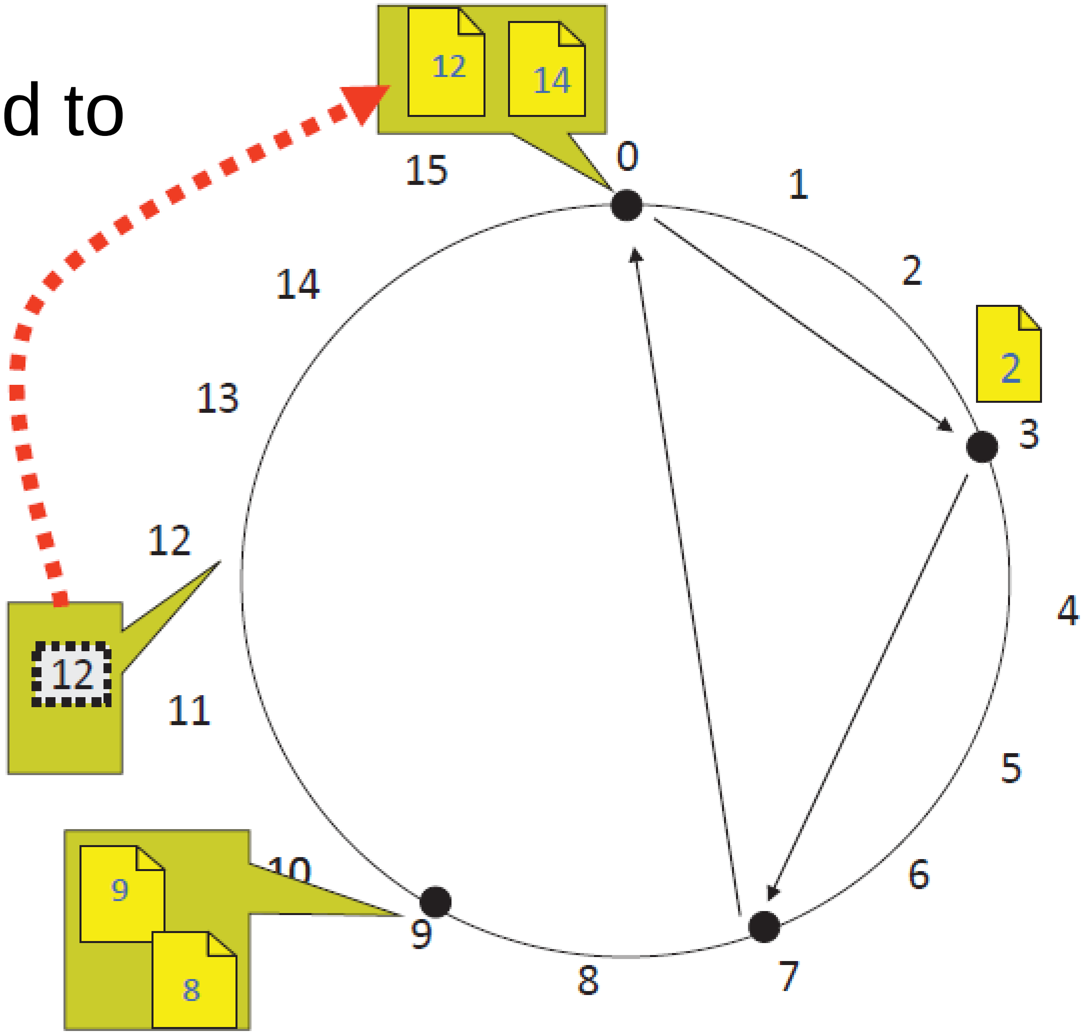
Node Leave

- Assume node 12 leaves gracefully



Node Leave cont'd

- Data is transferred to succ(12) = 14
- Node 12 informs predecessor and successor, who update their finger tables



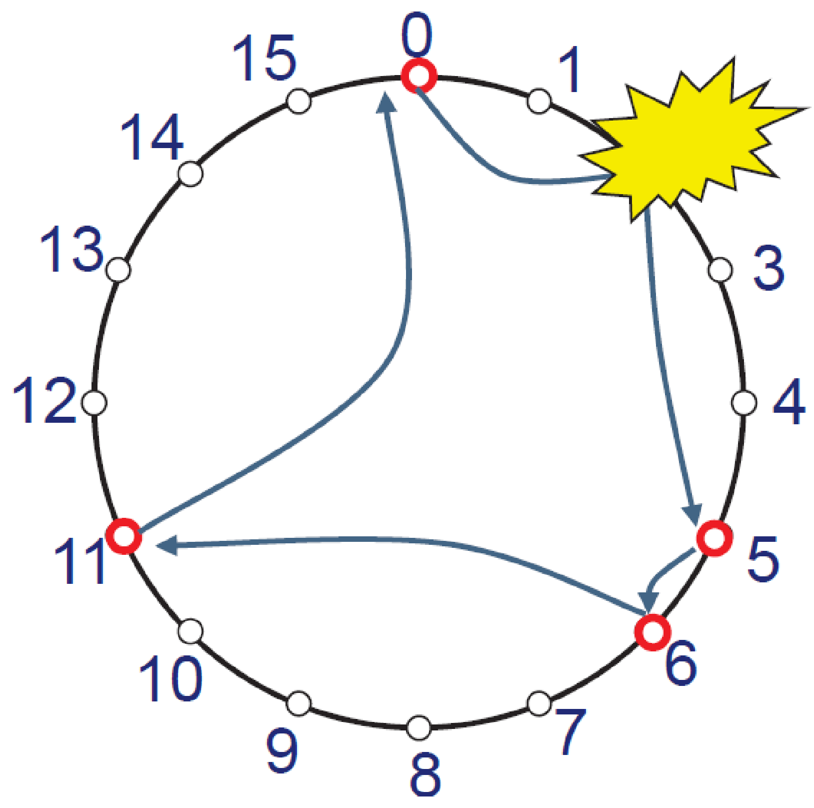
Direct vs. Indirect Storage

- Direct storage:
  - Actual data is stored at the node responsible for it
  - The data is copied towards the responsible node upon node join
  - The node that contributed the data can leave without loss of its data
  - But: High storage and communication overhead!

- Indirect storage:
  - Instead of data, the references to the data are stored
  - The inserting node keeps the data
  - Lower load on the DHT

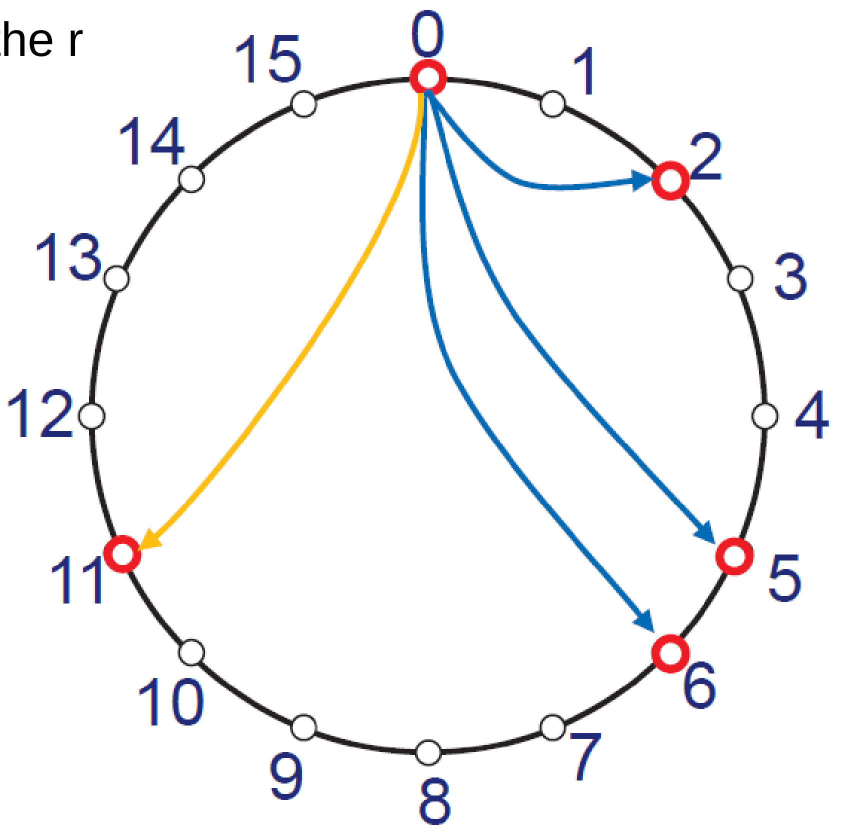
The Fragile Ring

- Problem: Everything is organized in a fragile ring structure
  - Failure of a node breaks the ring and data is lost
  - No way to recover as previous predecessor and successor don't know about each other!



Successor Sets

- As a solution, each node keeps:
  - A Successor set with pointers to the r closest successors
  - Predecessor pointer
- If successor fails, replace with closest alive successor
- If predecessor fails, set pointer to nil
- Replicate objects throughout the successor set



Further Challenges

- How does a node learn its:
  - Predecessors?
  - Fingers?
- What if “better” fingers come along later?
  - How would a node find out?
- How does a node react to failing or leaving fingers?

- All basically the same problem

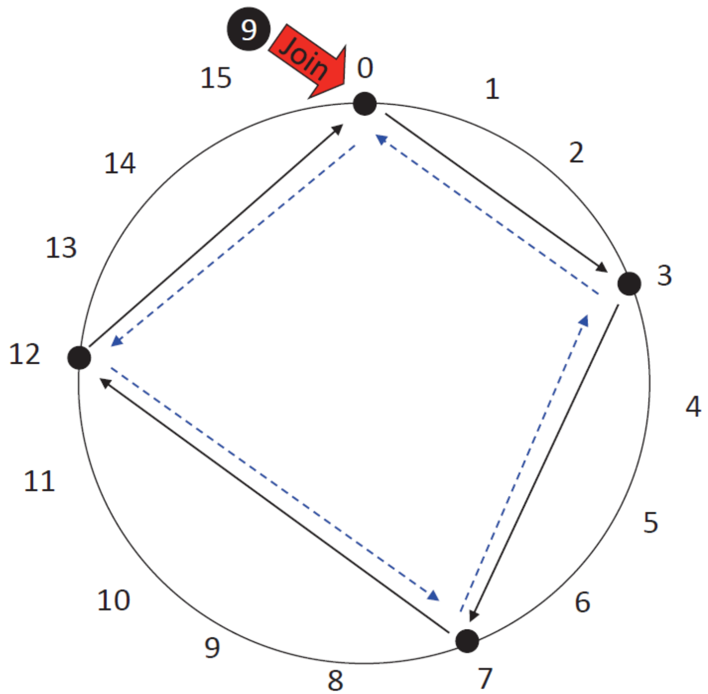
Periodic Stabilization

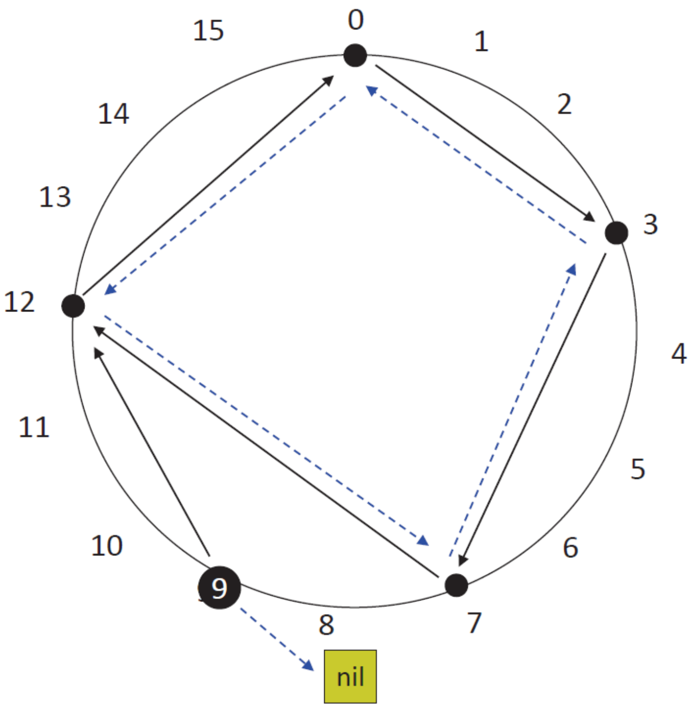
- Used to make pointers eventually correct
- Requires an additional predecessor pointer
  - First node met in anti-clockwise direction starting at n-1

- A node n joins the DHT through a node o:
  - Find n's successor by lookup(n)
  - n sets its successor to the found successor
  - Stabilization fixes the rest
    - stabilize() function is run peridically by each node
  - The new node does not determine its predecessor: its predecessor detects and fixes inconsistencies

Periodic Stabilization Example

1. 9 joins through node 0
2. 9 sets its predecessor to nil
3. 9 asks 0 for succ(9). Receives "12"
4. 9 sets its succ to 12

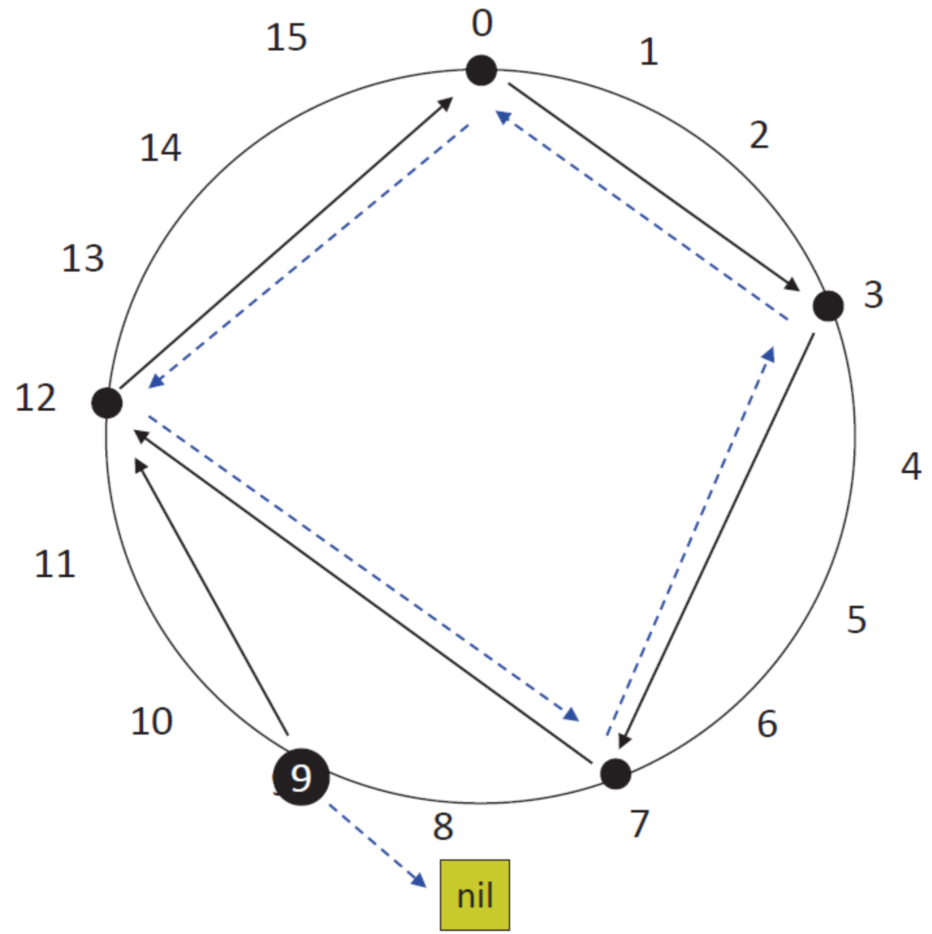




Periodic Stabilization Example

- 9 runs stabilize()

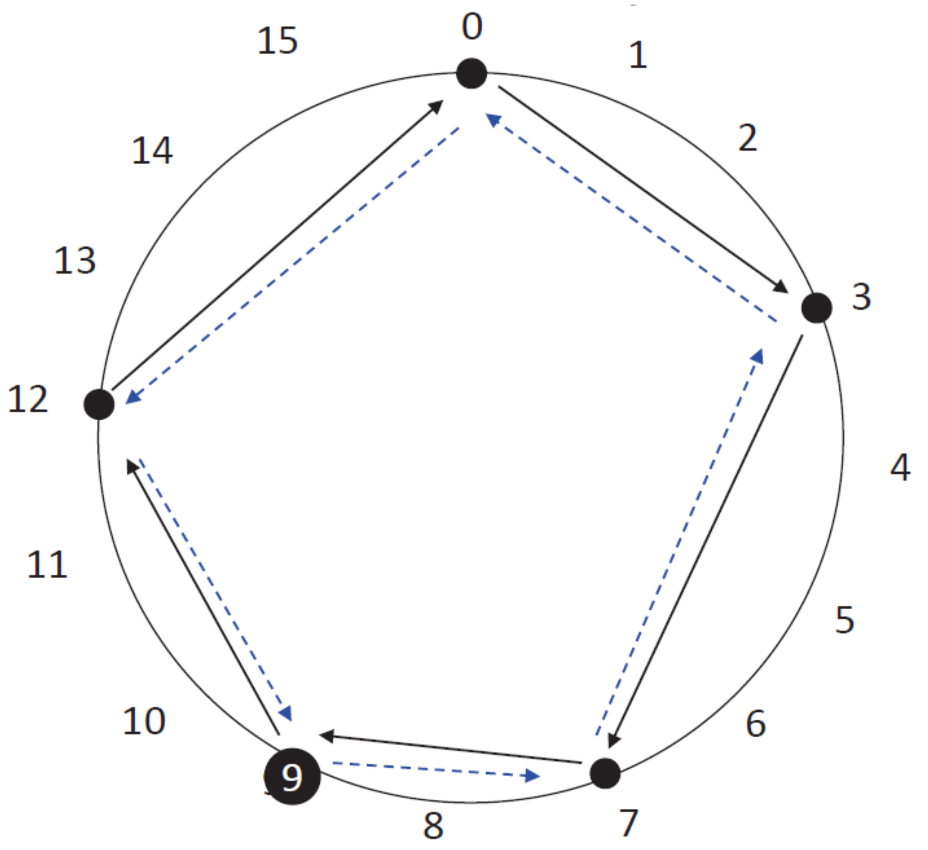
1. 9 asks 12 for its predecessor
2. 12 replies with “7”
3. 9 notifies 12 that 9 is now its predecessor



Periodic Stabilization Example

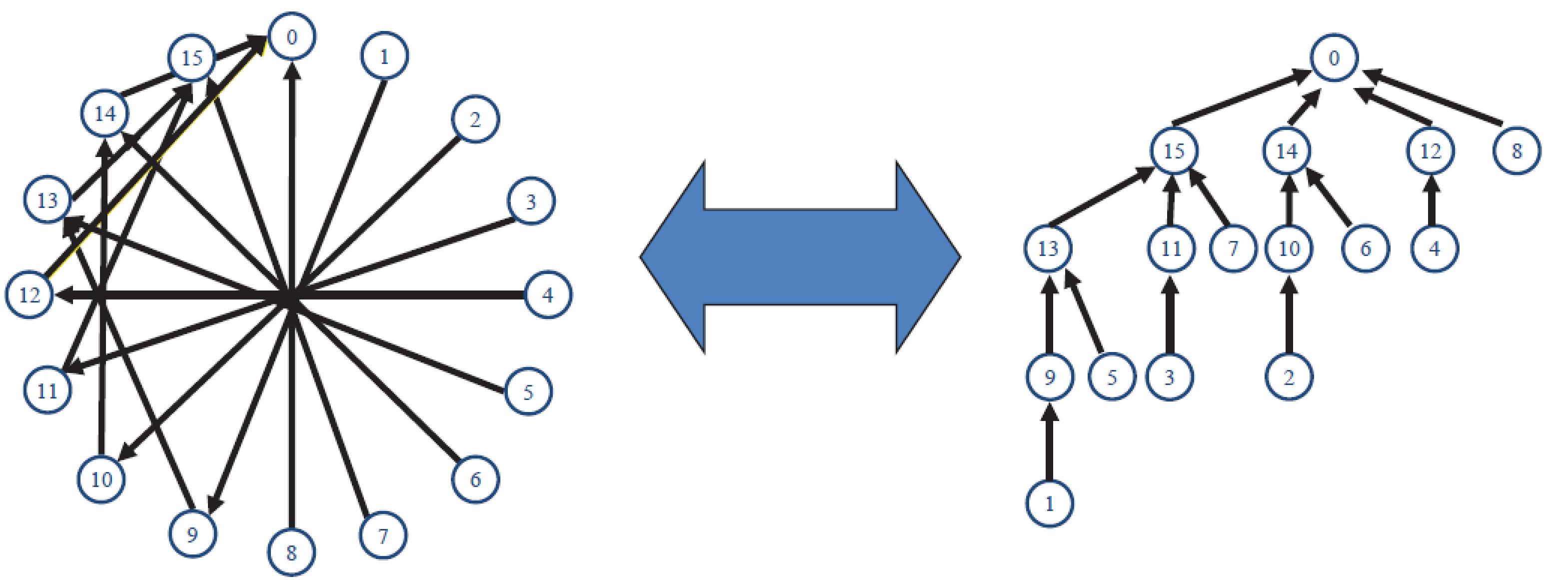
- 7 runs stabilize()

1. 7 discovers from 12 that pred(12) is now 9
2. 7 sets successor to 9
3. 7 notifies 9
4. 9 sets pred(9) to 7



Chord in a “Tree View”

- Finger tables are Chord’s core
  - Providing O(log n) hop routing by at least halving the distance to the target by each hop
  - Forest of binomial trees rooted at each key



Chord - Conclusion

- Lookup time: O(log n)
- Drawbacks:
  - Rigidity
    - Complicates recovery from failed nodes and routing table
    - Precludes proximity-based routing
  - Unidirectional routing
  - Incoming traffic is not used to re-enforce routing tables

- Fault-tolerant, but not very robust.

Other DHTs

- Kademlia (used in BitTorrent)
  - Lookup also done in O(log n) – as with most DHTs
  - Uses distance between two nodes: XOR of both nodes' IDs
  - Nodes still responsible for a part of ID space
  - Location of content basically the same as in Chord
    - Node closest to searched ID
    - O(log n) since XOR can halve distance at each hop
    - Note: This distance is ***not*** geographical

  - For details: Maymounkov and Mazières, Kademlia: "A Peer-to-peer Information System Based on the XOR metric", 2002

computer NETWORKS

References

- [1] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. 36(4), 335-371. 2004.
- [2] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. Comm. ACM 46,2(Feb.), 43–48. 2003.
- [4] Pouwelse, Johan; et al. "The Bittorrent P2P File-Sharing System: Measurements and Analysis". Peer-to-Peer Systems IV. Berlin: Springer. pp. 205–216. 2005.
- [5] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun, The Akamai Network: A Platform for High-Performance Internet Applications, ACM SIGOPS Operating Systems Review, Vol. 44, No.3, July 2010.