Peer-to-Peer Systems and Security IN2194

Chapter 1
Peer-to-Peer Systems
1.3 Structured Networks

Prof. Dr.-Ing. Georg Carle

Dr. Heiko Niedermayer

Overview

- Routing
- Using a Structure for P2P Routing
- Hash Tables / Distributed Hash Tables
- Chord
  - A structured KBR system and DHT
- Layers in Structured P2P Systems / Common API
- Other Structured DHTs / KBRs
  - Pastry, Kademlia, …
- Ordered Indexing

Routing

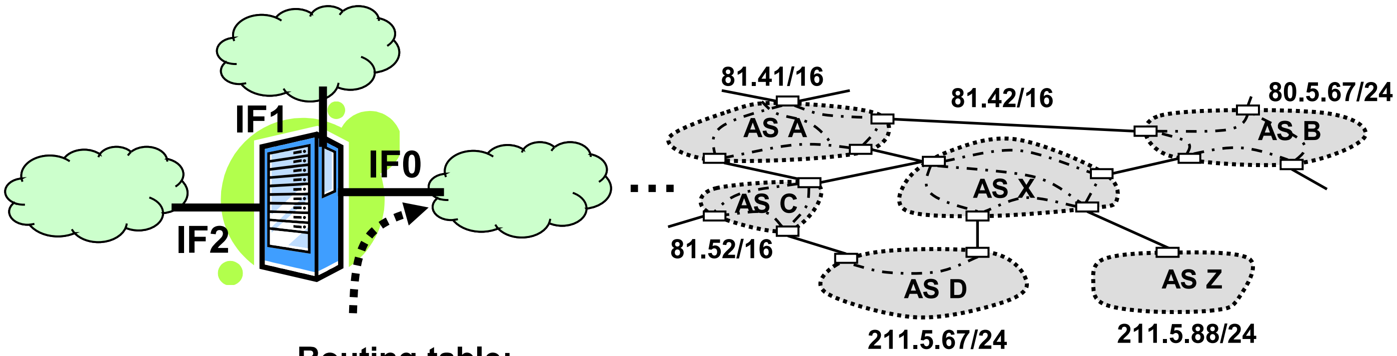
Motivation

- In unstructured networks we have to search all over the network for a node or item as we do not know where it is.
- Wouldn’t it be better to be able to simply say ”Ah, to node A, go this way!” Yes, it would.

Usually, this problem is separated into

- Routing
  - The task to find a way on a network is called routing. The routing table is created using the routing protocol which gathers information about the network and then computes the best paths.
  - Routing protocol => routing table
- Forwarding
  - For a packet that has to be sent or that arrives at an intermediate node (router) the node directly knows the next hop where to send it to. Usually, this is solved using a routing table that stores this information.

Routing on the Internet



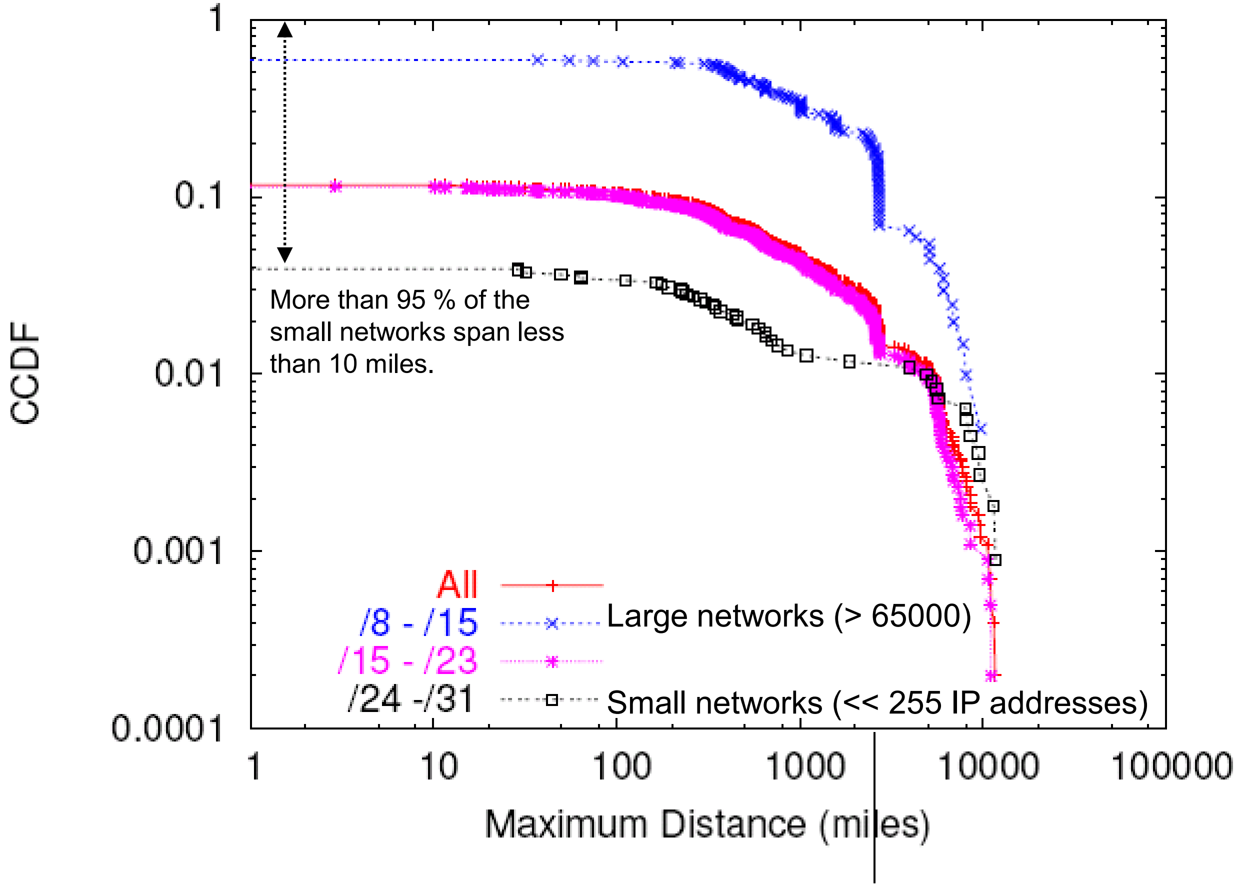
Routing table:
Send all 80/4 => IF0 (all from 80.0.0.0 to 96.255.255.255)
Send all 211/8 => IF0
…..

Scalability of the Internet Routing

- The Internet itself is not structured (no predefined structure!).
- The Internet is divided into Autonomous Systems (AS) that own a certain part of the IP address space.
  - After applying for an adress space (at one of the registries, e.g. RIPE) the addresses are manually set, but the routing is automized using routing protocols.
  - Distant computers only need to know the way to the AS.
  - Geographically close ASes often have close IP ranges.
- Consequences
  - IP addresses are not purely random, but cluster in certain areas. Due to this, we can group together many addresses in only one routing table entry.
  - Thus, routing tables still scale, even in the core network.

Routing on the Internet

- Geographic diversity of IP ranges



More than 95 % of the small networks span less than 10 miles.

Large networks (> 65000)

Small networks (<< 255 IP addresses)

N.Y<->Paris

Freedman et.al (2005)

Routing on the Internet and its relation to P2P

But here is the problem with unstructured overlay networks: If we introduce a routing protocol like in the Internet, we have randomly distributed addresses and cannot group them efficiently. The routing tables would not scale.

➔ We need to find a way to cluster nodes with similar IDs in the Peer-to-Peer network.

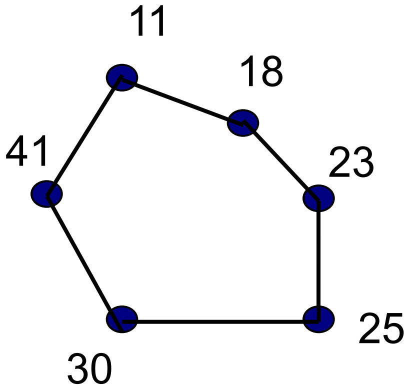
Structured Peer-to-Peer

❑ Structured Peer-to-Peer networks impose the constraint of a structure onto the Peer-to-Peer network.

❑ Each node is either not in the network or in a predefined position based on its node ID and given by the form of the structure.

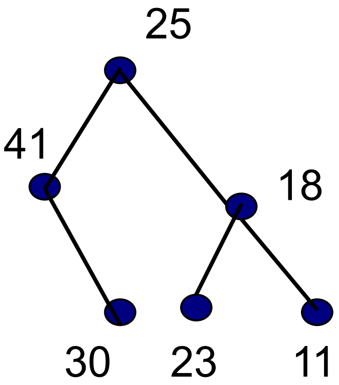
Using a Structure for P2P Routing

Examples for structures



Ring

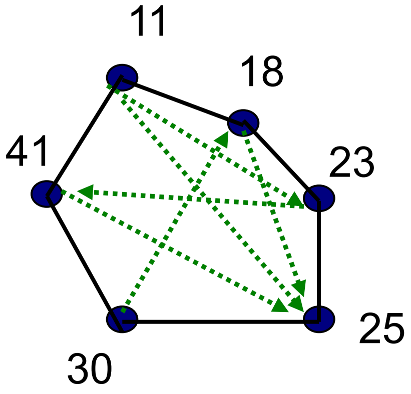
- ❑ Each node knows successor and predecessor.
- ❑ Sending a message
  - ▪ Unless the node is the target, forward the message to the successor



Tree

- ❑ Each node knows its parent and child nodes
- ❑ Sending a message
  - ▪ Up the tree if you are in a wrong subtree
  - ▪ Down the tree if you have the correct subtree as child
- ❑ This is not the way the tree-based DHTs operate!

Making structures efficient

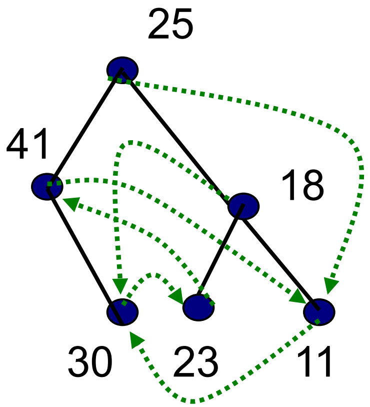


Local connectivity

- Neighbors in the structure ensure basic connectivity and clustering of similar IDs in one region.
- Connecting to more neighbors increases stability as nodes may leave at any time and the structure has to be maintained.

How long does a packet travel?

- In the ring O(n).
- In the tree O(n) unbalanced and O(logn) balanced.



Long-distance links / Shortcuts ⇢

- To reduce the diameter, introduce a set of long-distance links at each node.
- If we use the picture of clusters of nodes that have similar IDs, these links efficiently interconnect these clusters.

Distributed Hash Tables

Distributed Hash Tables

Hash Table

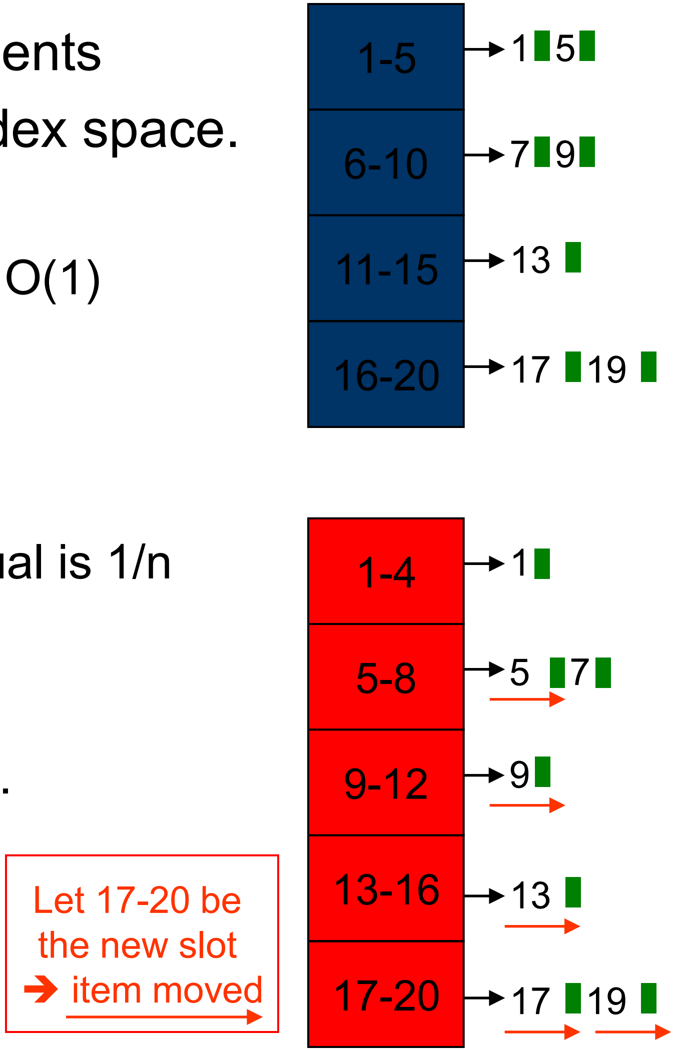
Hash Table

- ❑ n slots (nodes) that are used to store k elements
- ❑ The n slots contain an equal share of the index space.
- ❑ Features
  - ▪ Store elements, efficient amortized lookup in O(1)
  - ▪ h(element) determines slot
- ❑ Hash function h
  - ▪ uniform: all slots are equally likely
  - ▪ universal: propability of two hashes to be equal is 1/n

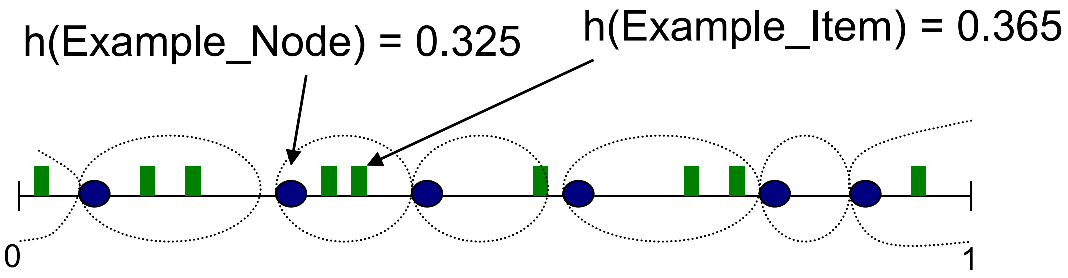
- ❑ What about adding or removing a slot?

  → Need to completely repartition the hash table.

- ❑ Can we avoid repartitioning?
  - ▪ Yes → Consistent Hashing.



Consistent Hashing



Consistent Hashing

- Approach to manage nodes and items.
- Originally developped for organizing distributed webcaches.
- Circular ID space [0,1)
- Map all nodes to unit interval [0,1)
- Map all items to unit interval [0,1)
- Assign items to nodes from their ID to the ID of their successor
  → nodes responsible for [node_ID,successor_ID)
- Allows to add / remove nodes without repositioning of all nodes.

➔ *Nodes* and *data* share same ID space.

Consistent Hashing – Theory – Load Balancing

Theorem „Consistent Hashing“

For any set of N nodes and K items, with high probability:

- ❑ Each nodes is responsible for at most $(1+\varepsilon)\frac{K}{N}$ items (ε = logN for consistent hashing in the way we described it).
- ❑ When an (N+1)st node joins or leaves the network, responsibility for only O(K/N) items changes.

Distributed Hash Tables

Distributed Hash Table (DHT)

- A Distributed Hash Table is a structured Peer-to-Peer system that provides hash table functionality.
  - Nodes and items share a common flat address space.
  - Nodes are responsible for certain parts of the address space.
  - Association of items and nodes may change due to the dynamics of the network.
  - Lookup of an item = routing to responsible node
  - Storage of an item = lookup responsible node and then use a store command to store the item
    - Usually, the DHT stores only reference pointers to sources of an item and not the item itself, e.g. „File XYZ.mpg can be found on 132.3.4.5:12345 and 55.65.3.4:12345."
- The terms „Structured Peer-to-Peer" and „DHT" are often used as synonyms.

DHT Routing / False Negatives

False Negatives

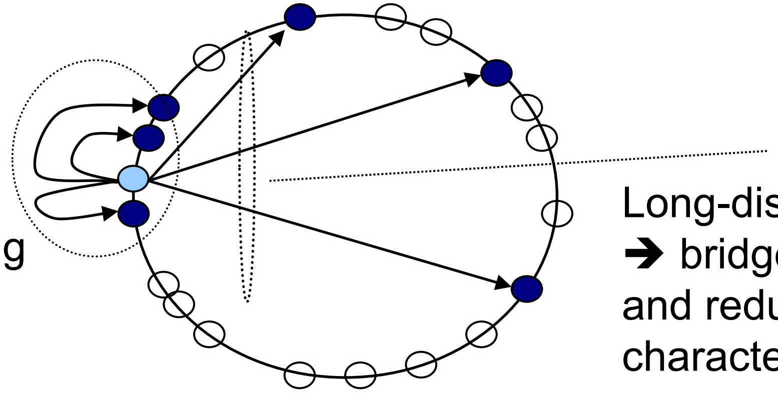
- ❑ Some algorithm does not find something despite it being there.
- ❑ Here: Item not found despite being in Peer-to-Peer system
- ❑ Unstructured Systems
  - ▪ Peer has to search the network, will not ask all peers
  - → False negatives happen by design.
- ❑ DHTs route the message to the responsible node for an item.
  - ▪ This node can tell if it knows the item or not.
  - → Thus, DHTs are free of false negatives.
    - • Exception: Routing errors or maintenance issues may lead to false negatives.

False Positive

- ❑ Here: Non-existing item is found.
- ❑ Not possible if all peers are honest, yet an attacker could invent items.

Chord

P2P Structure: Ring-based



Neighborhood
➔ Keep the ring connected

Long-distance links
➔ bridge long distances and reduce diameter and characteristic path length.

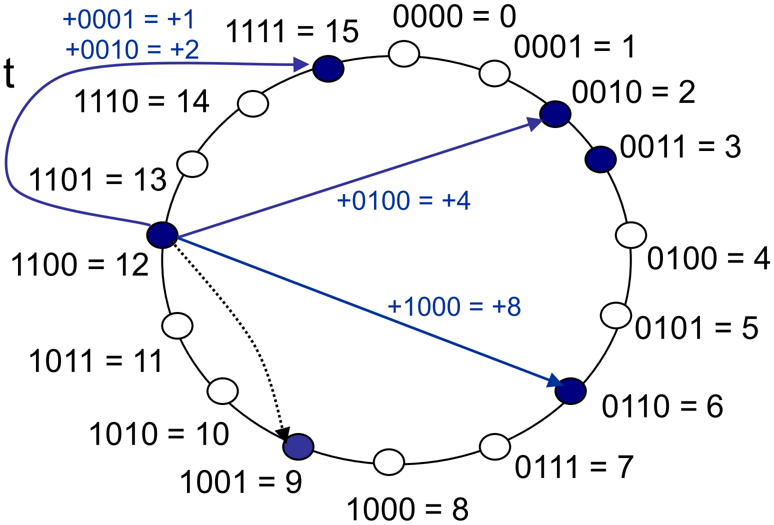
Ring-based Topology

- Nodes organize in a Ring.
- Links
  - To neighbors in the ring for stability and basic connectivity.
  - Long-distance links to achieve efficent routing.
- Examples
  - Chord
  - Symphony (embeds a symmetric small-world topology)

Chord

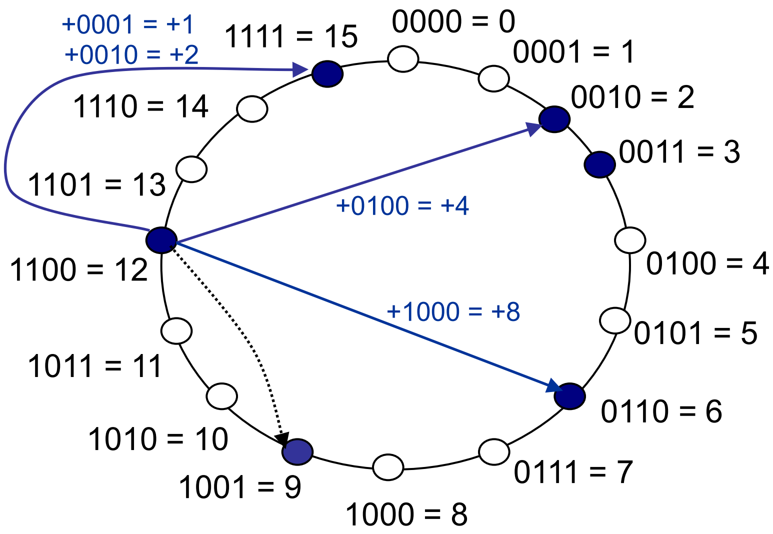
Chord

- Identifier space $[0, 2^m - 1]$, usually m=160.
- Mapping
  - proposed to use hash function sha1(.), e.g. node_ID = sha1(node_IP).
- Management of identifyers
  - A node is responsible for the interval (predecessor_ID,node_ID]
    - Thus, the successor of an ID is responsible for the ID.
- Graph Embedding
  - Neighbor set
    - Successor and predecessor (required, but predecessor only used for maintenance)
    - K successors (optional)
  - **Finger table** (Long distance links)
    - Link to node responsible for **node_ID+2^i** with i=1..160
    - These links are thus in exponential distance over the link index i.



Chord – Structure

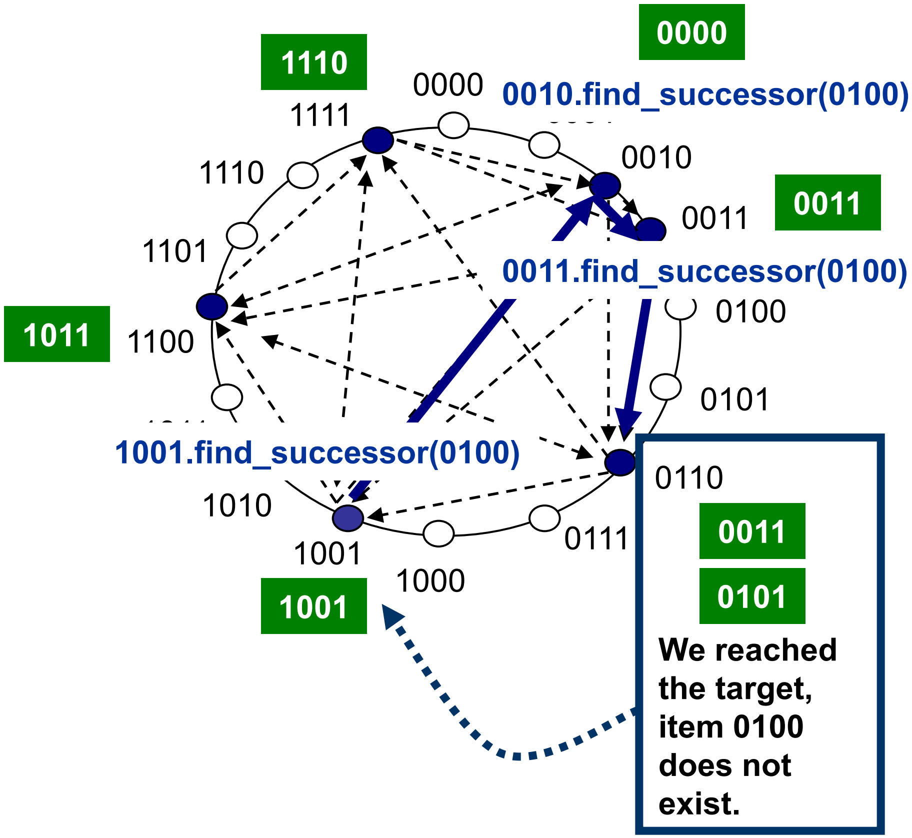
- ❑ Routing strategy
  - Greedy, goal is to find the successor of an ID.
- ❑ Maintenance
  - Join & Stabilization will be discussed on the next slides.



Chord – Lookup

Lookup

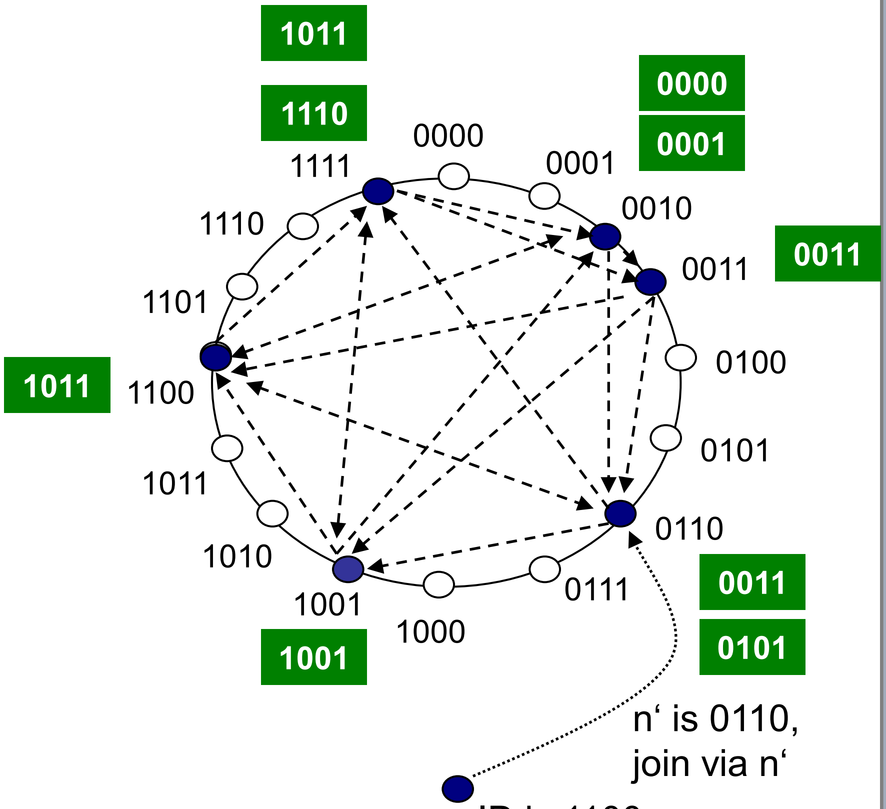
- ❑ Goal: Find successor(ID)
- ❑ Method
  - Node n either initiates or receives the query. If n is successor(ID), then we reached the target.
  - IF successor(n) == successor(ID) THEN
    - Forward to successor(n).
  - ELSE
    - The next hop is the **closest preceeding finger n‘** of ID in the finger table of node n.
    - Call n‘.find_successor(ID)


1110
0000
0000
0010.find_successor(0100)
1111
0010
1110
0011
0011
1101
0011.find_successor(0100)
1011
1100
0100
0101
1001.find_successor(0100)
0110
0011
0101
1010
0111
1001
1000
1001
We reached the target, item 0100 does not exist.

Chord – Join

Join

- Node n wants to join the network
  - n_ID is hash of its IP address (or randomly selected)
- n contacts n‘ with n‘ already in the network
  - n‘ has been found via some bootstrapping mechanism
  - If no node exists, n starts a new empty network.
  - n uses n‘ to build its finger table
    - n‘ does the lookups for successor(finger)
- n contacts its successor s, they divide the interval and n copies the data it is responsible for from s.
- n then contacts its predecessor and the predecessors of IDs that are likely to be required to link to n (→ ID – 2^i).



n‘ is 0110,
join via n‘

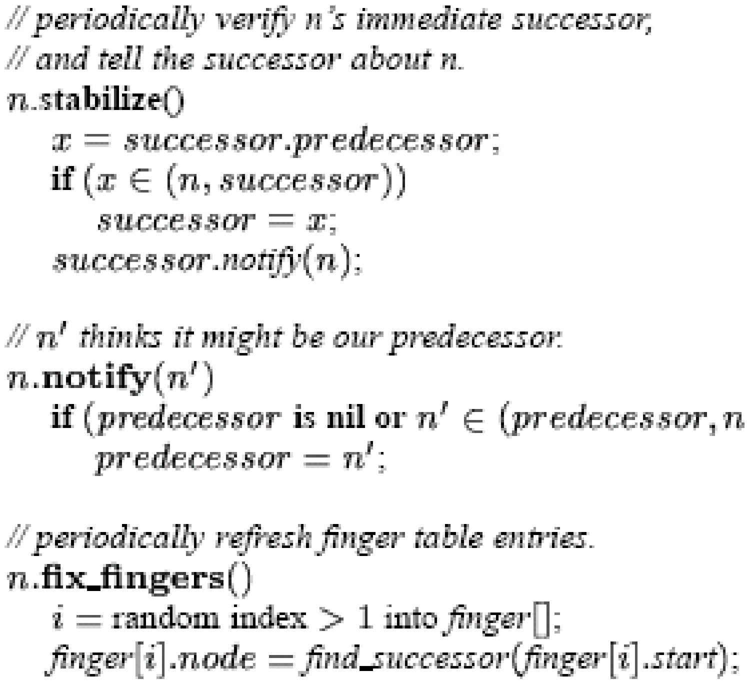
ID is 1100

successor(1100+1) -> 1111
successor(1100+2) -> 1111
successor(1100+4) -> 0010
successor(1100+8) -> 0110
predecessor(1100) -> 1001

Chord – Stabilization

Stabilization

- The stabilization is used to correct and maintain the finger tables. The goal is to converge to the correct fingers despite changes in the network.
- Pseudocode

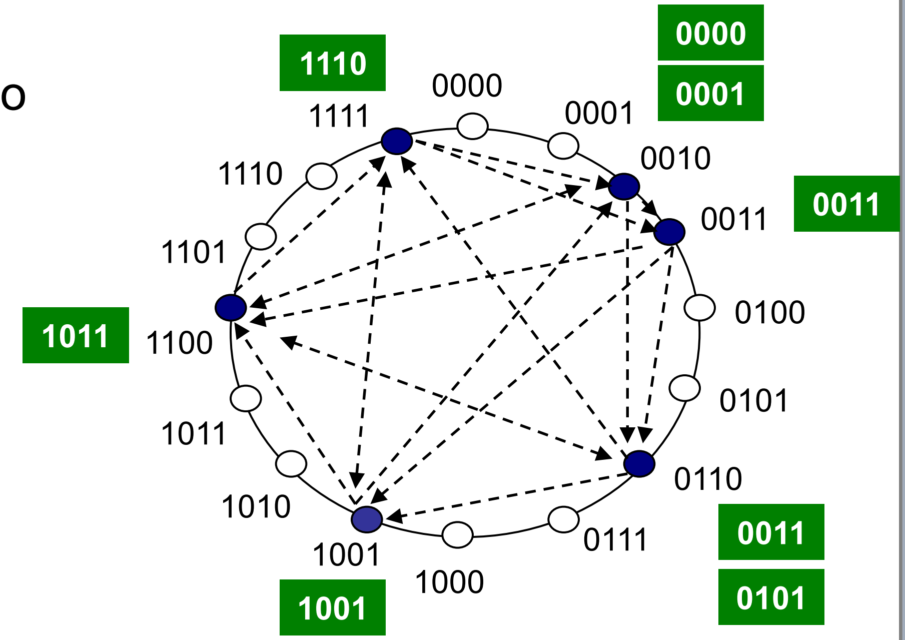


```
// periodically verify n's immediate successor,
// and tell the successor about n.
n.stabilize()
    $x = successor.predecessor$;
    if ($x \in (n, successor)$)
        $successor = x$;
    $successor.notify(n)$;

// $n'$ thinks it might be our predecessor.
n.notify($n'$)
    if ($predecessor$ is nil or $n' \in (predecessor, n)$)
        $predecessor = n'$;

// periodically refresh finger table entries.
n.fix_fingers()
    $i$ = random index > 1 into finger[];
    $finger[i].node = find\_successor(finger[i].start)$;
```

Some definitions for the pseudocode:
finger[i].start = n+2^i
finger[i].node = ID of node in finger table, >= n+2^i



Chord – Failure and Replication

Failure of a node

- ❑ Maintain a successor list with r successors
- ❑ Use successor list to handle the failure / leave of successor
- ❑ For the time from failure until stabilization repaired the network structure, the authors proposed to introduce timeout for messages and use a less-optimal node or other successor as next hop.

Replication

- ❑ Store items also on the r successors
- ❑ Has to be done by higher layer software though as Chord only does the Key-Based Routing part.

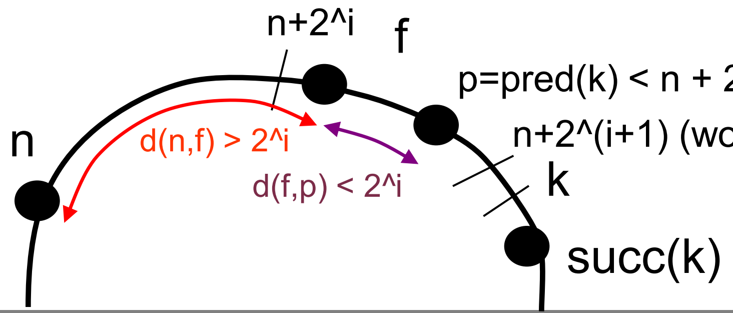
Chord – Theory – Lookup

Theorem (Chord Lookup)

With high probability, the number of nodes that has to be contacted to find a successor of an ID k in an N node Chord network is O(logN).

Proof

- Suppose, node n wishes to resolve query for successor of k. Let p be the node that immediately precedes k.
- If n != p then
  - n forwards the query to the closest predecessor of k
  - Suppose that p is in the i-th finger interval of n and that this finger is f.
  - d(n,f) > 2^i and that 2^i>d(f,p) ➔ d(n,p) = d(n,f)+d(f,p) > 2 d(f,p)
  - Thus, the distance is at least halved in each step.



p=pred(k) < n + 2^(i+1) otherwise n would have used finger i+1

n+2^(i+1) (worst case assumption that k is larger)

continues ⇨

Chord – Theory – Lookup

- Now, we consider node n and item k to be random. We show that w.h.p. the number of forwardings will be O(logN).
  - After logN forwardings distance was at least halved logN times. Thus, the distance will be reduced to at most

$$2^m \left(\frac{1}{2}\right)^{\log N} = \frac{2^m}{N}$$

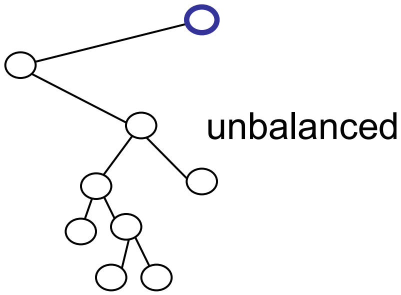
  - The exptected number of nodes in that area is 1 and the number of nodes is O(logN) w.h.p.
    - The latter follows from using the Chernoff bound on N Bernoulli experiments if node is in or not in the interval (hit with probability 1/N).
    - With high probability means that the probability that the assumption is wrong converges to 0 with 1 / (a polynomial) or alternatively that there is a constant c so that the error probability is bounded by $\frac{1}{N^C}$.
  - Thus, w.h.p. we need at most O(logN) more steps. ■
- Thus, w.h.p. we the end up with O(logN) nodes contacted.

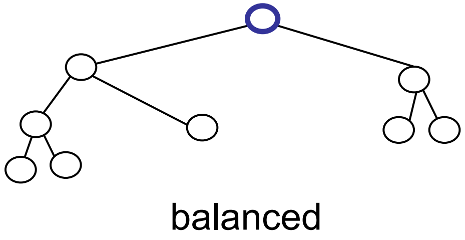
Chord / All structured P2P – Theory – Lookup

- Chord is assumed to half the distance per step

Generalization

- Any structured Peer-to-Peer system that halves the distance per O(1) steps achieves O(logn) lookup.
- The basis of the logarithm depends on the fraction of search space that is left after one step.
- Please note: To achieve this „tree-like“ search performance , the graph of P2P system should not be degenerated (e.g. all peers cluster in one area) and thus the virtual search tree with your node as root should be balanced, atleast with high probabilty.





Chord – Theory – Join

Theorem (Chord Join)

With high probabilty, each node joining or leaving an N node Chord network will use $O(\log^2 N)$ messages to re-establish the Chord routing invariants and finger tables.

Proof sketch

The basic idea is to show the following.

Once the new node has contact to its rendenzvous-peer its finger table has to be created. This consists w.h.p. of O(logN) entries that need O(logN) lookup each, thus ending up at $O(\log^2 N)$.

Similar assumptions hold for the links to the new node that have to be modified.

The data transfer of items is not part of the theorem. Only one node (the predecessor) has to be contacted for the transfer. ■

Layers in Structured P2P Systems / Common API

Key-Based Routing

DHT and Routing

- A hash table is an application where one can store and retreive data.
- DHTs need Key-Based Routing for their operations.

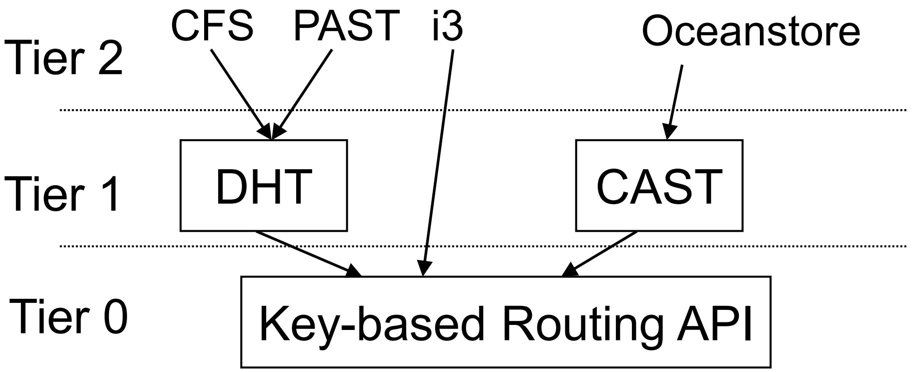
Key-Based Routing (KBR)

- Route and deliver messages to a key.
- The key is represented by the node that is responsible for the key.
- KBRs are usually structured networks.
- Most DHT proposals describe a Key-Based Routing system.

Identifier-Based Routing

- Like KBR, but to node with given ID.
  - Delivery fails if node does not exist.

Common API – Abstractions and APIs



Common API

- ❑ Idea to represent a standard set of functions with well-defined semantics to ease the development of Peer-to-Peer applications.
  - No widespread use though, except for Freepastry.
- ❑ Differentiate between different layers, e.g. DHT and KBR.
- ❑ Example APIs
  - Key-based Routing API
    - Functionality to route and deliver messages to keys.
  - DHT API
    - Hash table functionality

Key-based Routing API – Data Types

Data Types

- ❑ key
  - ▪ 160 bit string
- ❑ nodehandle
  - ▪ Encapsulates the transport address and nodeID of a node
  - ▪ *NodeID* is of type key.
- ❑ msg
  - ▪ Messages contain data of arbitrary length.

Conventions

- ❑ For read-only parameters: → p
- ❑ For read-write parameters: ⇔ p
- ❑ Ordered set p of type T: T[ ] p
- ❑ Root of a key = responsible node of key

Key-based Routing API – Routing Messages

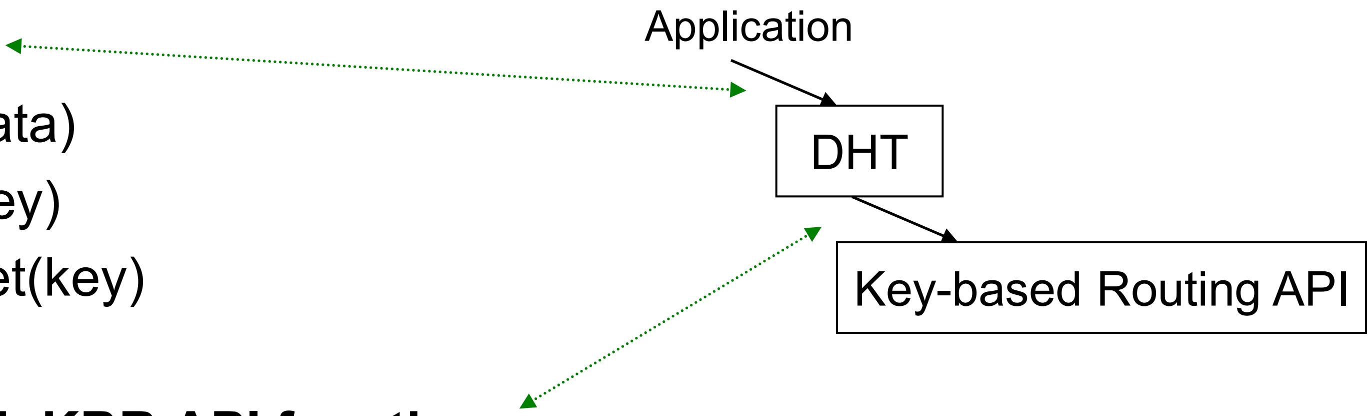
Routing messages

- void route(key → K, msg → M, nodehandle → hint)
  - Send msg to root of key K.
  - hint is an optional proposal for the next hop
  - Either K of hint may be null
- void forward(key ⇔ K, msg ⇔ M, nodehandle ⇔ nextHopNode)
  - Upcall to the application
  - Initiated before forwarding M
  - Application may modify the message and parameters.
- void deliver(key → K, msg → M)
  - Upcall to the application.
  - Invoked on the node that is root for K when M arrives.

Other definitions in KBR API

- Not in the lecture: API for route table maintenance (route state access), etc.

DHT API



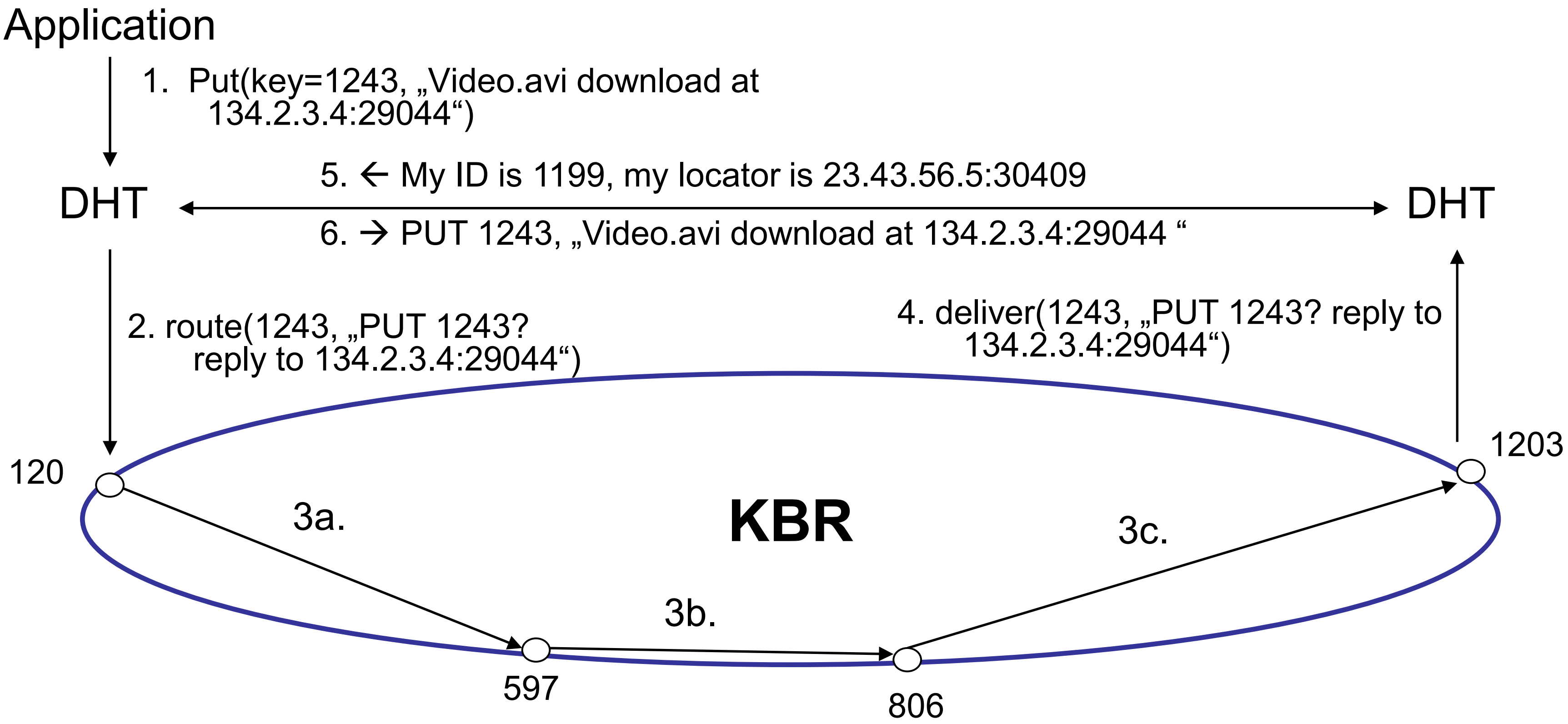
DHT API

- ❑ put(key,data)
- ❑ remove(key)
- ❑ value = get(key)

DHT API with KBR API functions

- ❑ PUT
  - Send a message with the put information to the root of the key.
  - No need for a hint (== next hop)
  - route(key,[PUT,value,S],null)
- ❑ GET
  - The requesting node uses route to find the root, the root returns the value directly using the hint option.
  - route(key,[GET,S],null)
  - route(null,[value,R],S)

Typical usage and interaction of DHT and KBR



Application

1. Put(key=1243, „Video.avi download at 134.2.3.4:29044“)

DHT

5. ← My ID is 1199, my locator is 23.43.56.5:30409

6. → PUT 1243, „Video.avi download at 134.2.3.4:29044 “

DHT

2. route(1243, „PUT 1243? reply to 134.2.3.4:29044“)

4. deliver(1243, „PUT 1243? reply to 134.2.3.4:29044“)

Designing a Key-Based Routing System (KBR)

Designing a Key-Based Routing System (KBR)

Structured Key-based Routing

Designing a structured KBR

1. Cluster nodes with similar IDs
   - → IDs
   - → Metric for IDs
   - → Connect neighbors and neighboring clusters
2. Speed-up
   - → Connect distant clusters
   - → Know more nodes or ask more nodes
3. Robustness
   - → Know more nodes
   - → Maintenance

KBRs differ in

- ❑ Topology
- ❑ Maintenance
- ❑ Lookup strategy / Message Forwarding

Topology and Maintenance

Topology

- Structure of the graph and embedding of IDs
  - Major issues in the next slides.

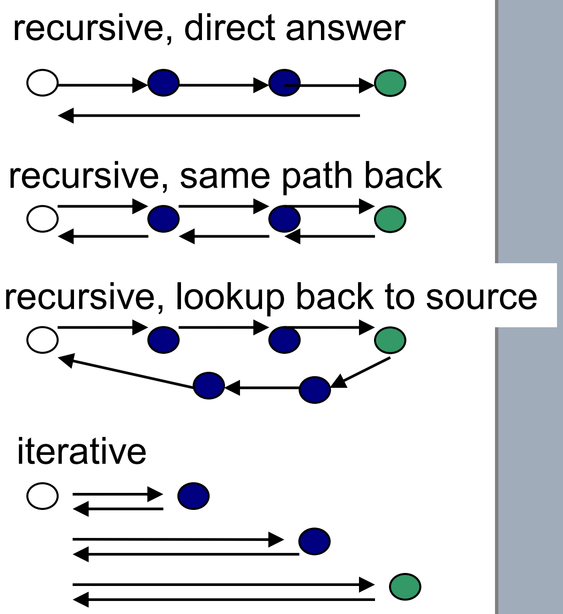
Maintenance

- Check if other nodes still exist.
  - Heartbeat messages, etc.
  - If not, repair the network.
- Check if structure is still correct.
  - If not, repair the network.
- Multiple nodes per direction / buckets
  - Know multiple nodes, so that failures can be circumvented once a packet needs to travel in this direction.
- …

Lookup / Message Forwarding

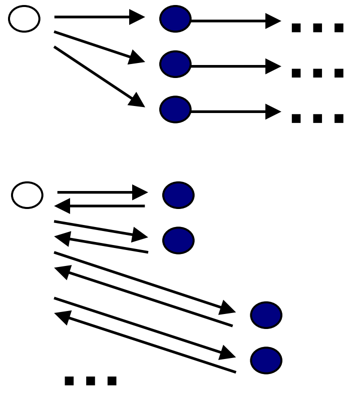
Lookup strategy

- Recursive
  - Lookup is forwarded though the network.
  - Answer may be sent back directly to source (standard) or through a path in the network.
  - Pro: uses existing connections
  - Con: message loss / failures harder to detect
- Iterative
  - Nodes are either the target or reply with next hop list.
  - The source does the lookup itself.
  - Pro: source has to work, source detects failures
  - Con: more messages, always connection setups



Lookup robustness and speed-up

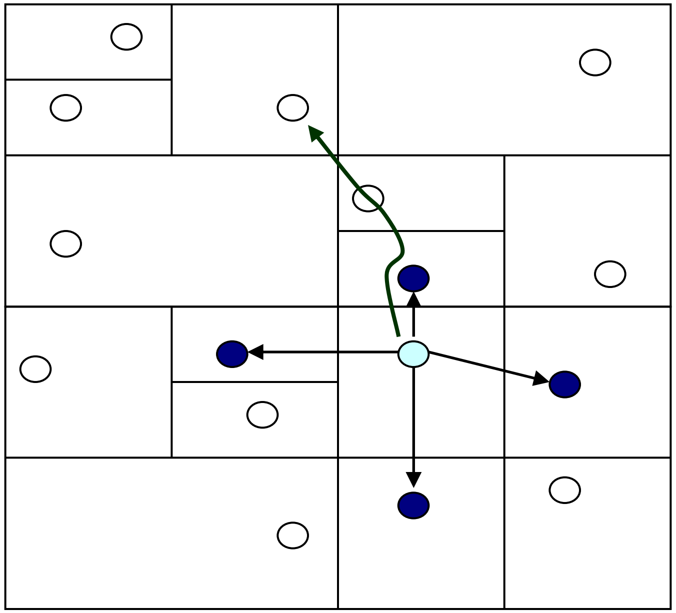
- Concurrent lookup
  - Ask k neighbors to forward or answer the query.
  - If less than k-1 nodes/paths fail or are slow, still one will answer in time.
- Caching of short-cuts and content
  - Cache target or intermediate nodes for future lookups.



Other Structured DHTs / KBRs

Other structured DHTs / KBRs

The Content-Adressable Network (CAN)



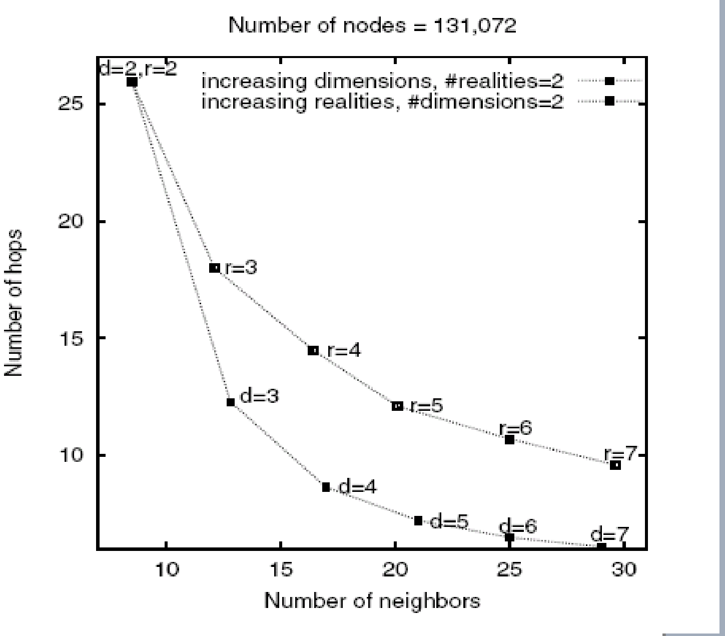
The Content-Adressable Network (CAN)

- Proposed by Ratnasamy et al in 2001
- Identifyer space: d-dimensional torus
- Management of identifyers
  - A node owns a zone of which its identifyer is a member.
- Graph Embedding
  - For each 2d directions, link to the owner of a neighbor zone in that direction.
- Routing Strategy
  - Select as next hop the neighbor closest to the target (Euklidean distance).
- Maintenance
  - Join
    - A nodes selects a random point and routes to that point.
    - The zone is then split into two equal parts.

The Content-Adressable Network (CAN)

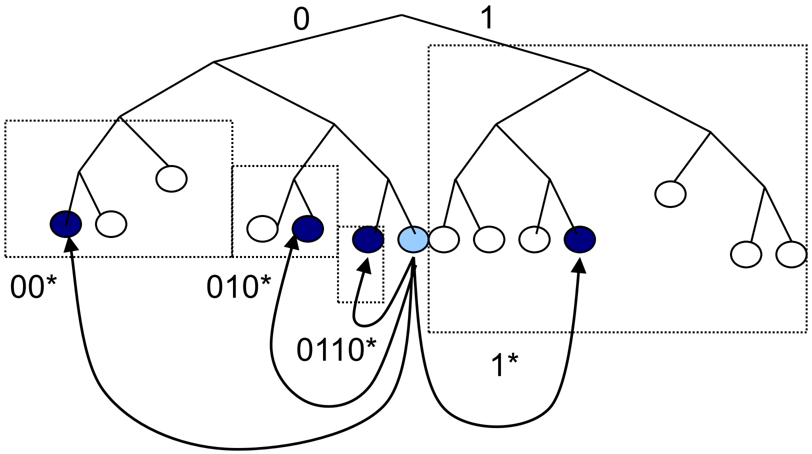
The Content-Adressable Network (CAN) – Results

- State per node: O(d)
- Average path length: $O(dn^{\frac{1}{d}})$
- Dimensions d
  - Increase in d decreases path length and increases fault taulerance.
- Realities r
  - Idea: Run r CANs (with different hash functions for the mapping of nodes and items) in parallel with same nodes and data.
  - Results: With respect to path lengths, increase in dimension is better.
- Idea: Allow p peers per zone
  - Avg. path length reduced by factor O(p).
  - Per-hop latency can be reduced as links with lowest latency can be selected in each direction.
  - Increase in p, increase in fault tolerance.



Ratnasamy et al (2001)

Tree-based Topology



- Also called Prefix-based routing or Plaxton‘s Mesh
- Idea
  - Maintain at least one link to each area with a prefix that is a shortest string not prefix to the node_ID
  - Example
    - Node_ID = 011010
    - Links to 1*, 00*, 010*,0111*,01100*,011011
- Examples
  - Pastry
  - Tapestry

Pastry

Pastry

- Identifyer space $[0, 2^m - 1]$, usually m=128.
- Management of identifyers
  - The numerically closest node is responsible for a key.
- Graph Embedding
  - Routing Table R
    - Let **b** bits be a **character**. The ID is then represented as a string of b-bit characters.
    - Idea: for each shared prefix length, have a link to one node in each interval with a common prefix of that length and a different next character.
    - Example: Node ID = 1023

	0	1	2	…	$f = 2^b - 1$
0	-0-231	------------	-2-333		-f-023
1	-----------	1-1-23	1-2-30		1-f-01
2	10-0-1	10-1-1	-------------		10-f-3
…	…	…	…		…
m/b					

Next character after common prefix

Shared Prefix length

This is the table of the IDs that we use to select the next hop ID, for each ID there is also the information IP:Port

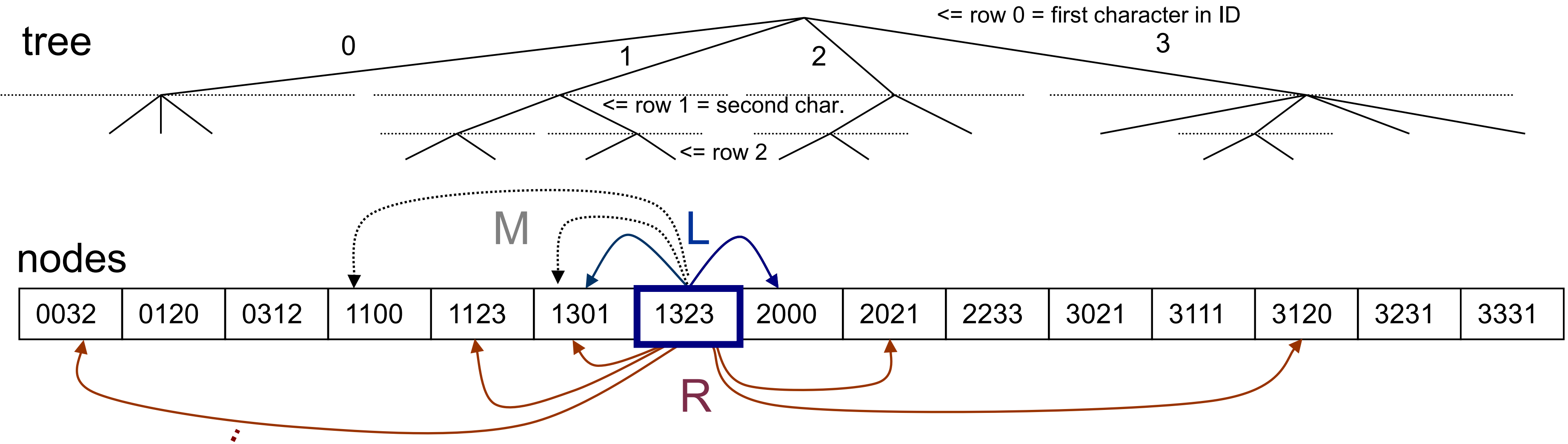
Pastry

- Leaf set L
  - Link to L/2 closest numerically smaller nodeIDs and to L/2 closest numerically larger nodeIDs
  - L usually 2^b or 2^(b+1)
- Neighbor set M
  - Maintain a set of nodes that are the closest known nodes according to some numeric proximity measure (IP hops or RTT)
  - [M| usually 2^b or 2^(b+1)

- ❑ Routing strategy
  - Let ID be the target and the current node n is not responsible
  - IF ID is within the leaf set THEN
    - Forward to closest node in leaf set
  - ELSE
    - Use routing table and forward message to a node that shares a longer common prefix with ID than n
    - If that is not possible use a node from L, R or M that does not share a longer prefix with ID, but is numerically closer to ID than n

Pastry - Routing

- Let b=2 (2-bit char.) and m=8 (4 characters), |L| = 2,|M|=2, node n=1323



For a link to an interval any node in the interval can be selected. So, simply select one you know. Instead of 0032, we could also have chosen 0120 or 0312.

	0	1	2	3	
0	0032	-	2021	3120	← to other subtrees of root
1	n/a	1100	n/a	-	← to other subtrees of own subtree
2	1301	n/a	-	n/a	← etc.
3	n/a	n/a	n/a	-	

with

n/a no node in interval

- prefix of node n, so no link

Pastry – Join

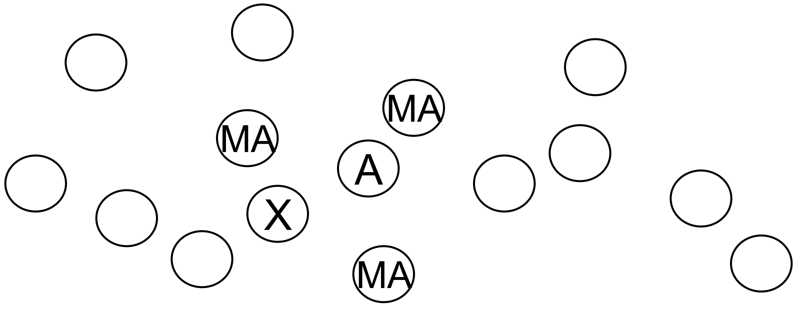
Join

- Say, node X wants to join.
- X knows a near Pastry node A according to the proximity metric
    - X uses some external information (say WWW) to find such a node
- X first selects a node ID, we call it X_ID.
- X sends a join message to A containing X_ID.
- A forwards the join message towards X_ID.
    - All nodes on the path including A reply to X with their state tables (R,M,L)
    - Z is the node responsible for X_ID
    - X may ask additional nodes
- X informs all nodes that need to know of X‘s arrival.

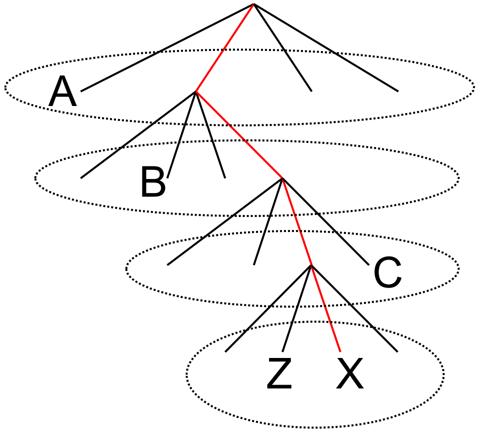
continues ⇨

Pastry – Join

- X uses the aquired information to build its state tables
  - Neighbor List M_X = M_A
    - as X and A are close in the underlay
  - Leaf set L_X = L_Z
    - as X and Z are direct neighbors
  - Routing table
    - Row 0: 0_X = 0_A
    - Row i: i_X = i_Intermediate-Hop_i
- X sends a copy of its state tables to all nodes in R,M, and L.
  - These nodes will then update their table according to this information.
    - E.g. A should add X to it neighbor list M_A.



X neighbor of A in underlay (real world)



use A and row 0 of A to fill row 0

use B and row 1 of B to fill row 1

use C and row 2 of C to fill row 2

Leaf set

Pastry – Locality

Locality

- ❑ Unlike most DHTs Pastry directly addresses the problem of locality, i.e. to prefer to have local links than links that cross the planet.
- ❑ Locality is measured by proximity metrics, e.g. IP hops.
- ❑ The neighbor set of a node holds a knowledge of close nodes according to such a proximity metric.
- ❑ Locality through the join process
  - ▪ Pastry assumes that a node n that joins the networks, joins via a geographically nearby node A.
  - ▪ As this node A already prefers routing table entries with good proximity, the state information of this node A and the other nodes is filled with nodes that are likely to be good nodes according to the proximity measure.

  ➔ Consequence, n is likely to fill its routing table with nodes that are close.

Pastry – Results

- Experimental evaluation

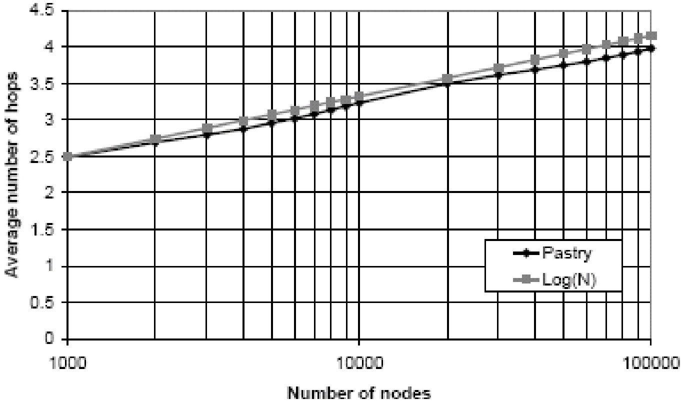


Fig. 4. Average number of routing hops versus number of Pastry nodes, $b = 4$, $|L| = 16$, $|M| = 32$ and 200,000 lookups.

Rowstron & Druschel (2001)

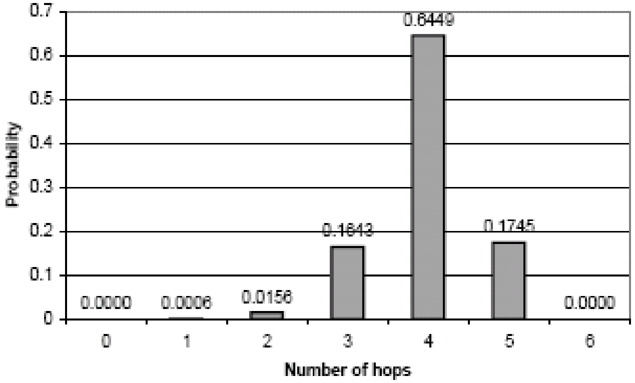


Fig. 5. Probability versus number of routing hops, $b = 4$, $|L| = 16$, $|M| = 32$, $N = 100,000$ and 200,000 lookups.

Rowstron & Druschel (2001)

Pastry

Discussion

- Routing hops

$$O(\log_{2^b} n)$$

- Node state

$$O((2^b - 1)\log_{2^b} n)$$

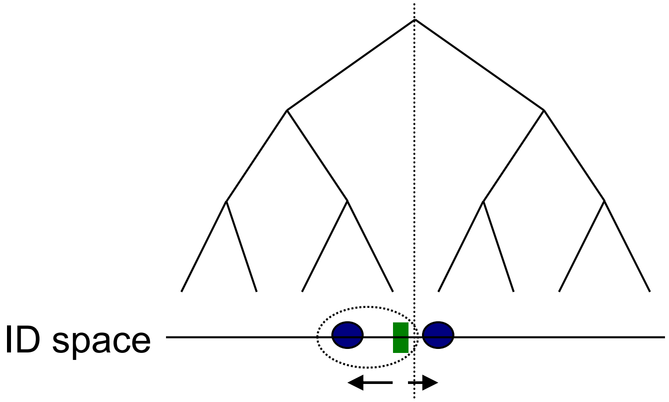
- Low overhead for join and leave.
  - Join

$$O(\log_{2^b} n)$$

  - Leave

$$O(\log_{2^b} n)$$

XOR topology



Item 🟩 is in left subtree.
Closest node is in right subtree.
Responsible node is in wrong subtree (→ Leaf set across subtrees).
→ With the XOR metric, however, the item is closer to any node in its subtree than to nodes in other subtrees.

XOR topology

- ❑ Closely related to the tree topology
- ❑ XOR as distance metric

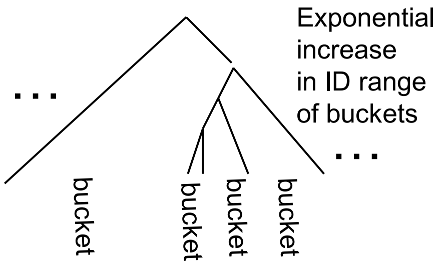
$$d(x,y) = x \oplus y$$

- ❑ Advantage of XOR metric
  - ▪ Unidirectional, that means that for a given x and D there is only one point y that satisfies d(x,y) = D. ➔ Lookups for same key converge to the same path, and thus caching items along this path is good to avoid hotspots.
- ❑ Example
  - ▪ Kademlia

Kademlia

Kademlia

- Identifyer space $[0, 2^m - 1]$, usually m=160.
- Mapping
  - proposed to use hash function sha1(.), e.g. node_ID = sha1(node_IP).
- Management of identifyers
  - The resposible node is the closest node to the ID according to XOR metric.
- Graph Embedding
  - k-Buckets
    - For any 0 ≤ i < 160, there is a k-bucket with up to k nodes with $d_{XOR}(ID, node\_ID) \in [2^i, 2^{i+1})$

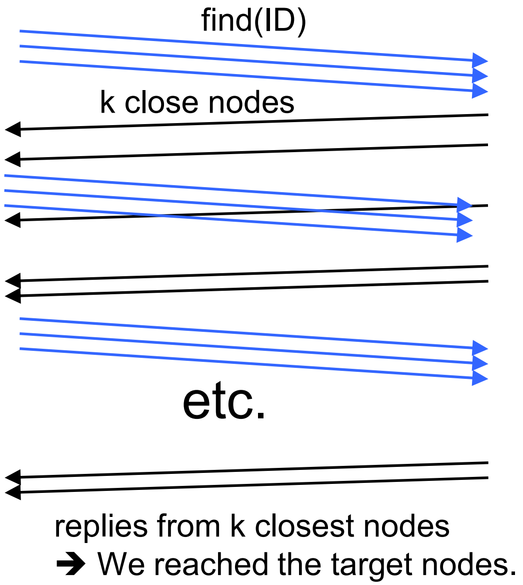


Exponential increase in ID range of buckets

-
  -
    - A k-bucket contains up to k nodes with their (IP, UDP port, ID).
    - If a k-bucket is full and new node found, the least-recently seen node r is pinged
      - It responds ➔ node r is moved to tail and new node is discarded
      - It does not respond ➔ add the new node to bucket, remove old node r.
    - The strategy motivated by the fact that in many networks nodes that have been in the network for a long time are more likely to stay than young nodes.

Kademlia

- Routing
  - Greedy according to XOR metric
- Lookup
  - Node
    - The initiator of a lookup asks the alpha closest entries from the bucket for the ID.
    - They return the k-bucket or alternatively the k closest nodes for the query in their buckets.
    - This is repeated, from the nodes received the alpha closest yet unknown nodes are also queried.
    - The lookup terminates when the initiator has replys from the k closest nodes it has seen.
  - Value
    - Analog, but anyone who knows the value does not reply with k nodes, but with the value.



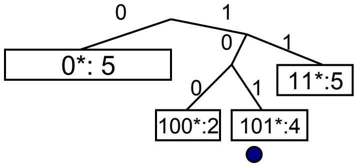
Kademlia

- Storage & Caching
  - To store a value, locate the k closest nodes to the ID via Lookup and then store the value at these nodes.
  - Values are considered softstate and need refreshing.
  - Values are cached at the first node on a path that did not know it.
- Join
  - Node u joins via an existing node w and they add each other to their k-buckets.
  - u performs lookup to its ID
  - u refreshs all k-buckets further away than its closest neighbor.
- Maintenance
  - Refresh k-buckets for which there was no contact within a certain time, e.g. an hour
  - Refresh means lookup of random ID in bucket.

Kademlia

Management of k-buckets

- Kademlia does not use all 160 buckets if they are not full.
- A node starts with one bucket from 00000… to 11111….
- A bucket is split if it contains the node and the node knows more than k nodes in the bucket.

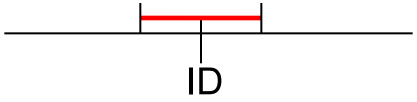


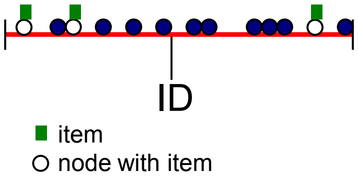
Example: buckets and size k = 5, node 101* interval

Tolerance Zone in KAD (a Kademlia derivative used in filesharing)

- KAD does not route to one exact ID.
- The tolerance zone is a zone around the ID.
  - e.g. first 16 bits in common with ID
- Items are stored on r nodes in its tolerance zone.
  - With r as the number of replicas (with one of them as responsible host).
- Lookup needs to find one node in the tolerance zone that knows the item.
  - Problem that this may not be the node closest to the item ID.

    → Searching necessary in tolerance zone.





Example: Kad network in Ed2k / aMule

The Kad network

- ❑ Based on Kademlia and used in clients like aMule, etc.
- ❑ „Buddy" function
  - ▪ Firewalled or NATed clients can ask other peers to support them as „buddy", only one buddy allowed and client waits 5min after firewall check before requesting a buddy.
- ❑ 2-layer publishing
  - ▪ Meta data (file name, file size, file type, file format, etc.)
    - • Keywords are extracted from file name, reference to sourceID stored at keywords (e.g. „P2P Vorlesung" → keywords „P2P" and „Vorlesung")
  - ▪ Sources
    - • Source published at sourceID = MD4_hash(compete file)
  - ▪ Replication
    - • Root for an item are nodes in a zone with a given prefix, e.g of 8 bit. For each write, there is a replication to 11 nodes in the zone.
- ❑ Keyword search
  - ▪ Lookup for first keyword in search string, rest of the key words are used to filter results

  → No fuzzy queries, range queries, …

Kademlia

Discussion

- The approach can be extended to work on a base of 2^b.
- Routing hops $O(\log_b n)$
- Node state $O(b \log_b n)$
- Low overhead for join and leave.
  - Join $O(\log_b n)$
  - Leave $O(\log_b n)$
- Kademlia is used in modern Peer-to-Peer systems like BitTorrent and in the Edonkey/Overnet/Kad Network family.
- Resistance against Denial-of-Service attacks
  - Buckets can not be filled with new bogus nodes as long as old nodes in a bucket are still alive.
  - Iterative and parallel lookup makes it hard for an attacker to block queries.

Constant-Degree KBRs

With m=O(log(n)) state and L=O(log(n)) DHTs do not achieve the performance of random graphs/small-world graphs. Lets recap the

$$L_{random} \sim \frac{\log n}{\log(m/n)} \underset{\frac{m}{n}=const}{\sim} \log n$$

- ➔ Random graphs achieve L=O(log(n)) with constant degree. This is an average and the O(logn) we give for the KBRs is a maximum with high probability.
- ➔ Can we build structured networks with constant degree and O(logn) hops?

We can, even with degree 2, e.g. binary trees, Viceroy (KBR based on butterfly graph), de Bruijn graphs, Kautz graphs, Distance-Halving.

- ❑ However, short distances are not for free, constant-degree graphs have longer average paths because they have significantly less links!

Example: De Bruijn graphs

Operations

- Let Σ be a set of symbols, say $\Sigma = \{0,1\}$.
- Shuffle S $(s_1, s_2, s_3, ..., s_k) \rightarrow (s_2, ..., s_k, s_1)$
- Shuffle-Exchange SE $(s_1, s_2, s_3, ..., s_k) \rightarrow (s_2, ..., s_k, \Sigma \setminus s_1)$

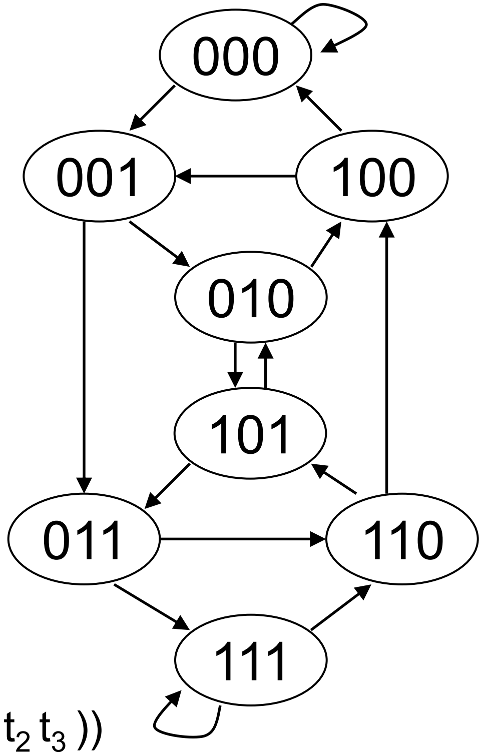
De Buijn graphs

- A node identifyer is then a fixed-length string of these symbols.
- Each node with node_ID has links to nodes that are either S(node_ID) or SE(node_ID).
- Formally:

$$V = \{(s_1 s_2 ... s_k) \mid s_i \in \Sigma\}$$

$$E = \{((s_1 s_2 ... s_k), (t_1 t_2 ... t_k)) \mid t_1 = s_2, t_2 = s_3, ..., t_{k-1} = s_k\}$$

- Routing
  - From s=(s_1 s_2 s_3) to t=(t_1 t_2 t_3) use the links ((s_1 s_2 s_3), (s_2 s_3 t_1)) then ((s_2 s_3 t_1), (s_3 t_1 t_2)) then ((s_3 t_1 t_2), (t_1 t_2 t_3))



Constant-Distance KBRs

One-Hop-DHT

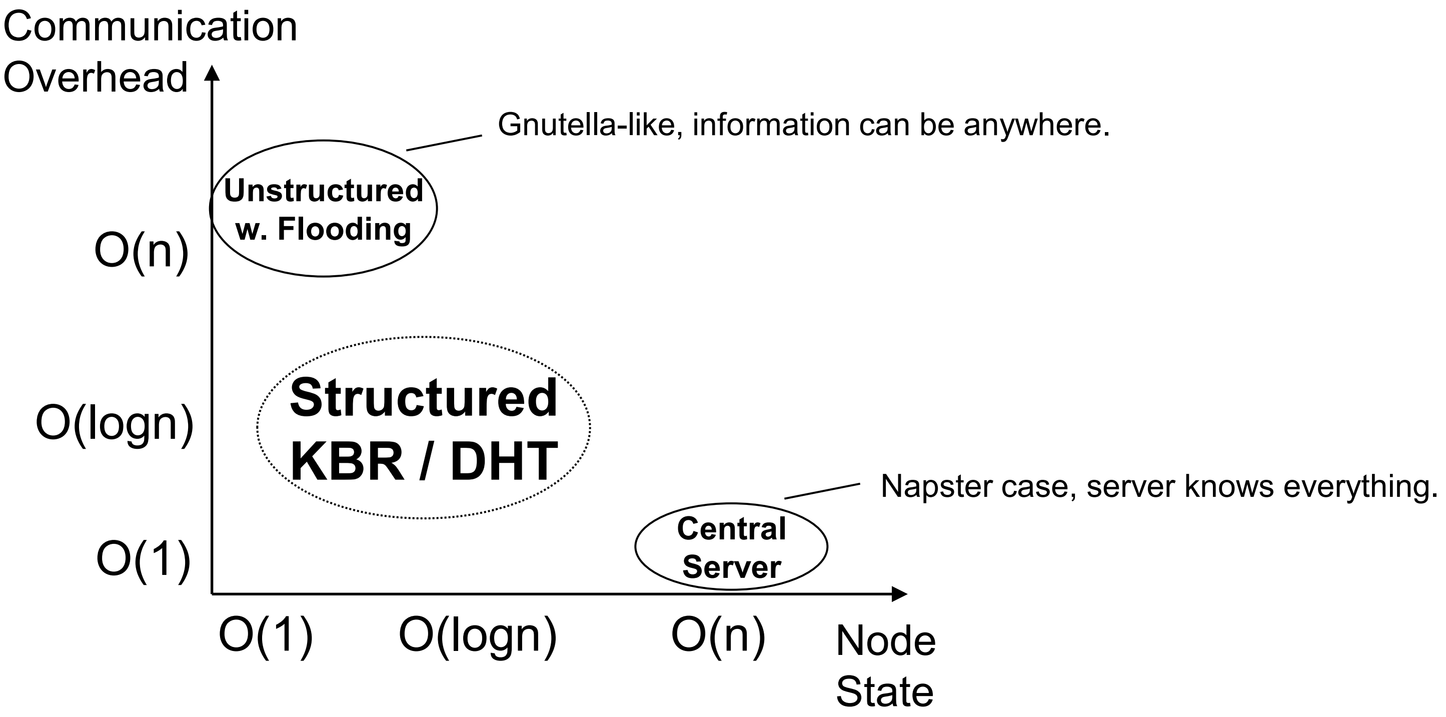
- Structure: Full Mesh / Clique
  - All nodes know each other.
- Limited scalability due to O(n) state per node and O(n) operations per change
  - Hard to maintain for large networks.
  - Authors claim that routing tables with millions of nodes are no problem with current RAM.

Constant-Distance

- 2, 3, … hops instead of 1 hop
- Construction
  - Reduce number of links compared to One-Hop-DHT as long as diameter is still below threshold
  - Alternative: add links until diameter is below threshold

Structured vs Unstructured vs Server

- Comparing DHTs with unstructured networks and central servers



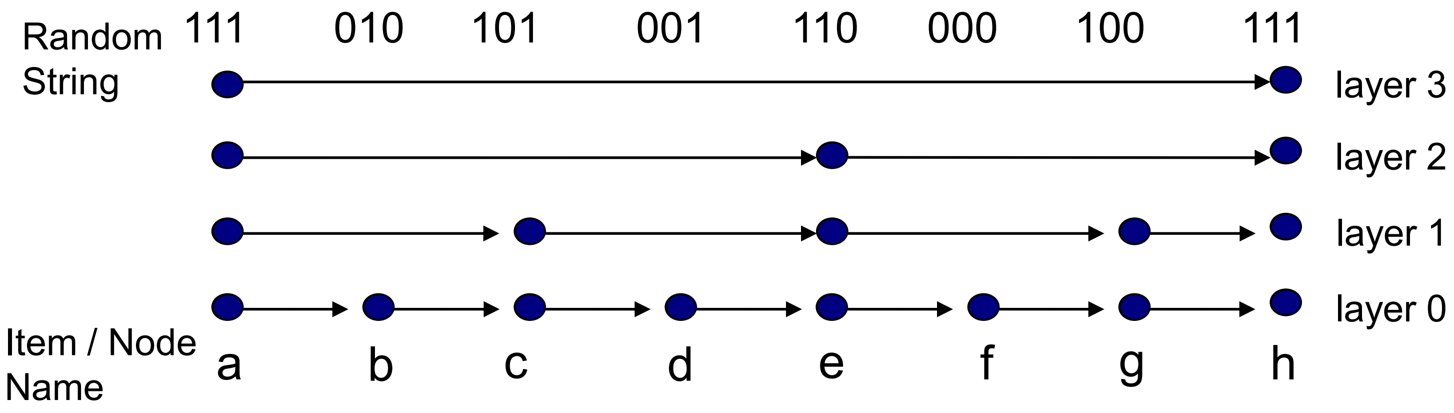
Ordered Indexing

Ordered Indexing

DHT vs Ordered Indexing

- Common DHT use case
  - ID = hash item name
  - Why the hash function?
    - Fixed bitlength
    - Balance the items over ID space
  - Problem: Find all words that start with „Peer“ is not effient in DHTs
  - Other options instead of hash function
    - Use DHT without hashing, but with load balancing (→ next chapter)
    - Ordered Indexing
- Ordered Indexing
  - Build an efficient structure without hashing
- Trie
  - reTRIEval tree

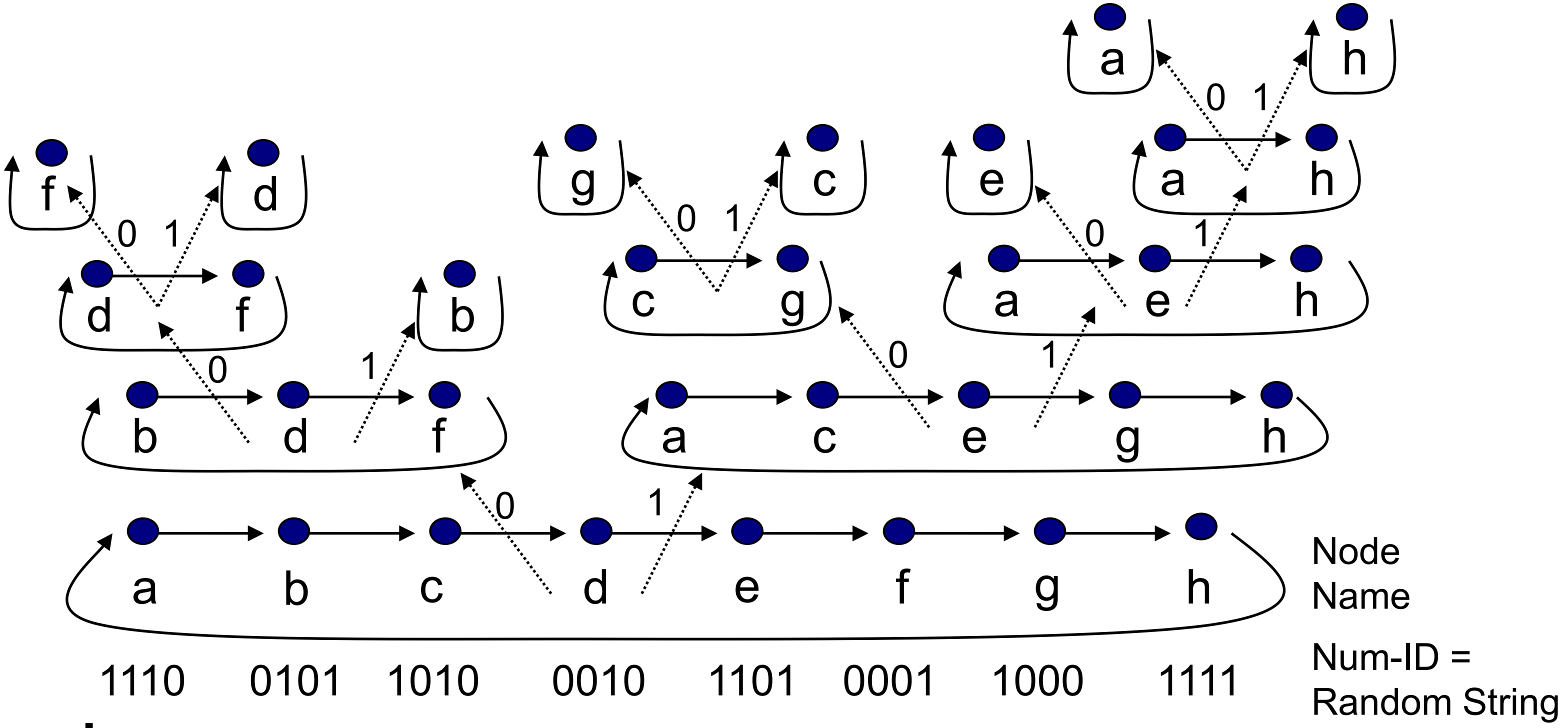
Skip List



Skip List

- ❑ A linear list is inefficient → add „long-distance“ links
- ❑ Idea
  - ▪ Add layers as long as there are more than 2 nodes per layer.
  - ▪ A n-layer consists of a randomly selected subset of the (n-1)-layer.
    - The random string for a peer corresponds to this random selction process.
      - 1 means „part of this layer“
      - 0 means „not part of any further layer“
- ❑ Achieves O(log n) hops with O(log n) in- and out-degree.

Skip Graph



Skip Graph

- Adapts the skip list idea to Peer-to-Peer networks.
- Idea
    - Layer 0 is a circle with all nodes.
    - Recursion: Split (n-1)-layer nodes into two random sets according to the bit of random string at position n. Each set forms again a circle.

Skip Graph

Node Identities

- Name: arbitrary name of node (item), e.g. tum.i8.heiko
- Num-ID: random number for each node

Search for Name

- Next hop selection
  - Start with the highest layer.
  - IF the next hop is closer to the name and still before the name in the order of the names (e.g. alphabetical)
  - ELSE Check lower layer for next hop. ENDIF

Example (graph on last slide):
A looks for F
→ next hop: E
E looks for F
→ next hop: F

Search for Num-ID

- Start search on lowest layer for node with correct next bit, then go to next higher layer.

Example:
A looks for 0000
→ next hop: B (0101)
B looks for 0000
→ next hop: D (0010)
D looks for 0000
→ next hop: F (0001)
(closest match)

Results

- Both search operations take O(log n).
- Skip Graphs support range queries (e.g. all names from c to e).