Distributed Hash Tables (DHTs): Chord & Pastry

Roman Dunaytsev

The Bonch-Bruevich Saint-Petersburg
State University of Telecommunications

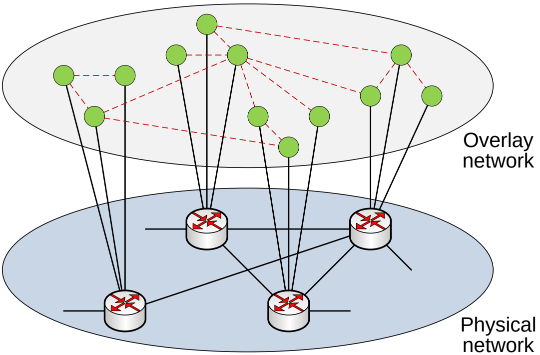
roman.dunaytsev@spbgut.ru

Lecture № 3

Outline

Introduction

- In P2P systems, cooperative peers self-organize themselves into **overlay networks** and relay/store data for each other
- The major challenge is how to achieve **efficient resource search** in these large-scale distributed-storage networks

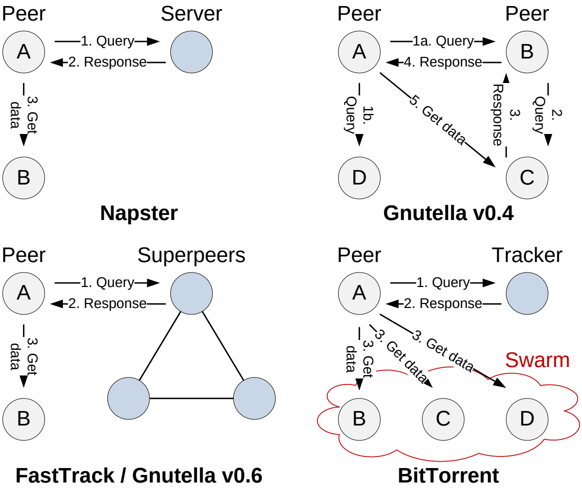


Introduction (cont'd)

- **2 types of overlays**:
  - Unstructured
  - Structured
- **Unstructured systems** – do not impose any structure on the overlay networks or loosely structured
  - E.g., Napster, Gnutella, Freenet, FastTrack, eDonkey2000, BitTorrent
  - Usually resilient to peer dynamics
  - Support complex search based on file metadata
  - Low search efficiency, especially for unpopular files
- **Structured systems** – impose particular structures on the overlay networks
  - E.g., **Distributed Hash Tables (DHTs)**
  - The topology of the peer network is tightly controlled
  - Any file can be located in a small number of overlay hops

Introduction (cont'd)

- Search process in unstructured P2P systems



Introduction (cont'd)

- Basic features of structured P2P overlays:
  - Structure – to accommodate participating nodes and data in the overlay
  - Routing algorithm – to locate nodes in the overlay and insert/retrieve data to/from them
  - Join/leave mechanisms – to enable self-organization and fault tolerance
- **Structures** (aka **geometries**)
  - Structured overlays use a number of different geometries (rings, trees, hypercubes, tori, . . .)
  - The primary goal is to enable the deterministic lookup (i.e., access guarantees with certain time bounds)
  - The performance is directly related to how nodes are arranged and how the overlay structure is maintained when nodes arrive and leave

Introduction (cont'd)

- **Routing algorithms**
  - They define how a target node is located in the overlay network
- DHT-based routing algorithms work as follows:
  1. The node ID space is formed by applying a hashing function to node IDs (e.g., MAC/IP addresses)
  2. A commonly used hashing function is **SHA-1** (Secure Hash Algorithm version 1)
  3. IDs for data items are created by applying the same hashing function to them (e.g., filenames or keywords)
  4. Thus, the node IDs and data IDs fall into the same address space
  5. Data items are typically stored on the closest node with the node ID greater than or equal to the data ID
  6. If the node with the closest ID does not store the data item, then it is not available in the network
  7. Using this approach any existing data item can be found by any node in the overlay

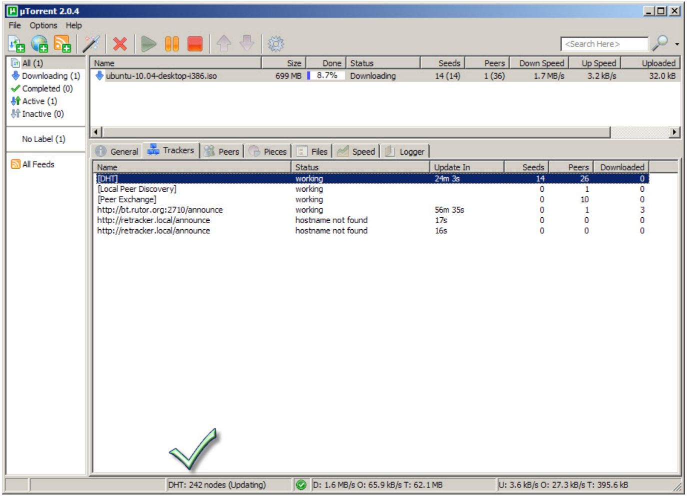
Introduction (cont'd)

- **Join/leave mechanisms**
  - Usually, P2P systems are highly dynamic in nature (aka **node churn**)
  - Hence, some mechanisms are needed to allow nodes to join or leave the system at any time with minimal impact to the functioning of the overlay
- Nodes **join** as follows:
  1. Get a unique ID for the node
  2. Position itself into the overlay structure based on the node ID and the geometry
  3. Update the routing tables (both the joining node and all the affected nodes)
- **Bootstrapping**:
  - As a rule, a new node contacts a bootstrapping server first and gets a partial list of existing nodes
  - Another common approach is to let nascent nodes know in advance an **entry point** into the network (e.g., a list of known nodes of the overlay or a list of non-public bootstrapping servers)

Introduction (cont'd)

- Nodes **leave** as follows:
  1. When a node leaves or becomes unreachable, nodes that point to that node are affected
  2. Their routing table entries will be stale and have to be updated
  3. A **gracefully** departing node notifies its neighbors about its departure
  4. Its neighbors then propagate these changes if needed
  5. However, a node may leave the system **unexpectedly** (e.g., due to network failure or power outage)
  6. Under these circumstances, the node will not notify its neighbors
  7. Hence, the system must have some **failure detection** mechanism
  8. Failure detection is usually handled by keep-alive messages or periodic checking

- In P2P file sharing systems, DHT just helps peers to find each other

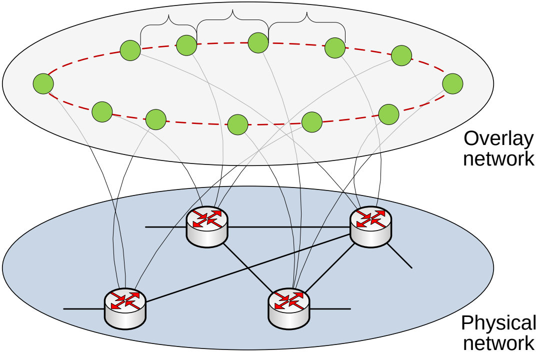


Chord

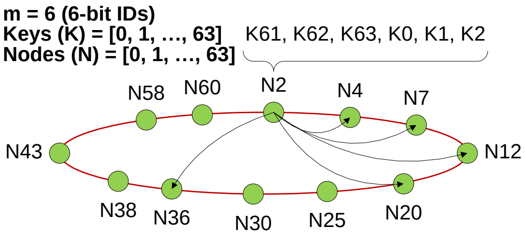
- **Chord** was proposed in 2001 by Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, and was developed at MIT
  - See "Chord: a scalable peer-to-peer lookup service for Internet applications"
- Chord uses consistent hashing and SHA-1 as a hash function to assign each node (by hashing the node's IP address) and each data item an m-bit ID, where m is a predefined system parameter
- These IDs are arranged as a **circle** modulo 2^m, from 0 to $2^m - 1$
- **Modulo arithmetic** is a system of arithmetic for integers, where numbers "wrap around" after they reach a certain value – **the modulus**
  - E.g., $7 + 7 + 7 \equiv 9 \pmod{12} \Rightarrow$ 9:00 PM or 21:00 (mod 24)

Chord (cont'd)

- Data items are mapped to nodes whose ID is greater than or equal to the ID of the data item (aka a **key**)
  - Due to **consistent hashing**, all nodes receive roughly the same number of keys and can join/leave the system with minimal disruption
- Thus, a node in a Chord circle with clockwise increasing IDs is responsible for all keys that precede it counter-clockwise



- Each node has a **successor** and a **predecessor**
- Since nodes may disappear from the network, each node records several nodes preceding it and following it
- Each node also maintains information about (at most) m other neighbors, called **fingers**, in a **finger table**
- The i-th entry, $i = 1, 2, \ldots, m$, in the finger table of node N points to the node whose ID is the smallest value bigger than or equal to $N + 2^{i-1} \pmod{2^m}$ in the clock wise direction



Finger table of node 2 (N2)

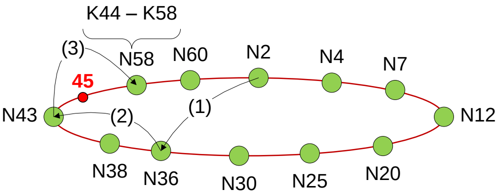
i	Target	Successor
1	$2 + 2^0 = 3$	N4
2	$2 + 2^1 = 4$	N4
3	$2 + 2^2 = 6$	N7
4	$2 + 2^3 = 10$	N12
5	$2 + 2^4 = 18$	N20
6	$2 + 2^5 = 34$	N36

Chord (cont'd)

- **Chord routing algorithm**:
- The primary goal of the routing algorithm is to quickly locate the node responsible for a particular key
- Chord routing works as follows:
  1. A key lookup query is routed along the ID circle
  2. Upon receiving a lookup query, the node first checks if the lookup key ID falls between this node ID + 1 and its successor ID
  3. If it does, then the node returns the successor ID as the destination node and terminates the lookup service
  4. Otherwise, the node relays the lookup query to the node in its finger table with ID closest to, but preceding, the lookup key ID
  5. The relaying process proceeds iteratively until the destination node is found

Chord (cont'd)

- $m = 6$ (i.e., modulo $2^m = 64$); 12 nodes; node 2 looks up key 45
  - (1) N36 is the closest to key 45; (2) N43 immediately precedes key 45; (3) N58 is the first successor of key 45 on the circle



(1) Finger table of N2

Target	Suc.
2 + 1 = 3	N4
2 + 2 = 4	N4
2 + 4 = 6	N7
2 + 8 = 10	N12
2 + 16 = 18	N20
2 + 32 = 34	**N36**

(2) Finger table of N36

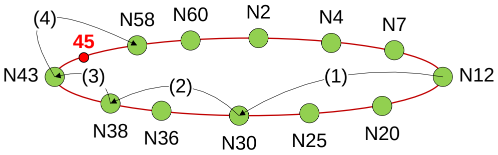
Target	Suc.
36 + 1 = 37	N38
36 + 2 = 38	N38
36 + 4 = 40	**N43**
36 + 8 = 44	N58
36 + 16 = 52	N58
36 + 32 ≡ 4	N4

(3) Finger table of N43

Target	Suc.
43 + 1 = 44	**N58**
43 + 2 = 45	N58
43 + 4 = 47	N58
43 + 8 = 51	N58
43 + 16 = 59	N60
43 + 32 ≡ 11	N12

Chord (cont'd)

- $m = 6$ (i.e., modulo $2^m = 64$); 12 nodes; node 12 looks up key 45
  - (1) N30 immediately precedes key 45; (2) N38 immediately precedes key 45; (3) N43 immediately precedes key 45; (4) N58 is the first successor of key 45 on the circle



(1) Table of N12	
Target	**Suc.**
12 + 1 = 13	N20
12 + 2 = 14	N20
12 + 4 = 16	N20
12 + 8 = 20	N20
12 + 16 = 28	**N30**
12 + 32 = 44	N58

(2) Table of N30	
Target	**Suc.**
30 + 1 = 31	N36
30 + 2 = 32	N36
30 + 4 = 34	N36
30 + 8 = 38	**N38**
30 + 16 = 46	N58
30 + 32 = 62	N2

(3) Table of N38	
Target	**Suc.**
38 + 1 = 39	N43
38 + 2 = 40	N43
38 + 4 = 42	**N43**
38 + 8 = 46	N58
38 + 16 = 54	N58
38 + 32 ≡ 6	N7

(4) Table of N43	
Target	**Suc.**
43 + 1 = 44	**N58**
43 + 2 = 45	N58
43 + 4 = 47	N58
43 + 8 = 51	N58
43 + 16 = 59	N60
43 + 32 ≡ 11	N12

Chord (cont'd)

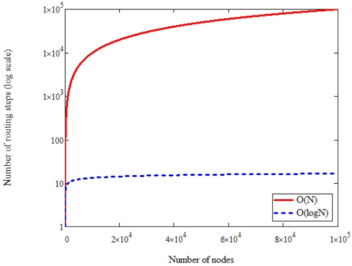
- As a finger table stores at most m entries, its size is independent of the number of keys or nodes in the network
- The Chord routing algorithm exploits the information stored in the finger table of each node
  - A node forwards queries for a key K to the closest predecessor of K on the ID circle according to its finger table
  - For distant keys K, queries are routed over large distances on the ID circle in a single hop
  - The closer the query gets to K, the more accurate the routing information of the intermediate nodes on the location of K becomes

Chord (cont'd)

- It has been shown that the number of routing steps in Chord is at the order of $O(\log N)$, where N is the total number of nodes
  - According to the Change of Base Theorem, when we talk about logarithmic growth, the base of the logarithm is not important:

$$\log_a N = C * \log_b N, \qquad C = \log_a b, \quad a, b > 0, \quad a, b \neq 1$$

- Queries on an unstructured P2P network tend to have lookup complexity of the order of $O(N)$



Chord (cont'd)

- **Chord join/leave mechanisms**:
- Nodes join as follows:
  1. The newly arrived node first uses consistent hashing to generate its ID
  2. It then contacts the **bootstrapping** server to lookup the successor ID
  3. This successor node becomes new node's successor node
  4. The joining node is inserted into the overlay and takes on part of the successor node's load
  5. The new node uses a stabilization protocol to verify its finger table
- To validate and update successor pointers as nodes join and leave the system, the **stabilization protocol** is executed periodically at the background of individual nodes
- When a node detects a failure of a finger during a lookup, it chooses the next best preceding node from its finger table

Pastry

- **Pastry** was proposed in 2001 by Antony Rowstron and Peter Druschel, and was developed at Microsoft Research, Ltd., Rice University, Purdue University, and University of Washington
  - See "Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems"
  - The Pastry project: www.freepastry.org
- Similar to Chord, its main goal is to create a completely decentralized, structured P2P overlay in which objects can be efficiently located and lookup queries efficiently routed

Pastry (cont'd)

- In Pastry, data items and nodes have unique 128-bit IDs, ranging from 0 to $2^{128} - 1$
  - For the purposes of routing, these IDs are treated of as sequences of digits in base 2^b
  - Typically, $b = 4$, so these digits are hexadecimal (HEX)
- These IDs are arranged as a **circle** modulo 2^{128}
- The node IDs are randomly generated at node join, and uniformly distributed in the ID space
- Instead of organizing the ID space as a Chord-like circle, the Pastry routing is based on **numeric closeness** of IDs
  - When forwarding a message to a destination key K, a node will choose the node in its routing table with the longest prefix match

Pastry (cont'd)

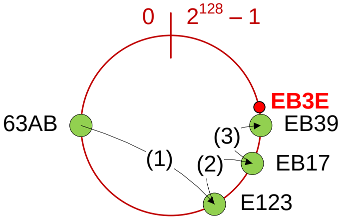
- Each node in Pastry maintains 3 tables:
  - Routing table
  - Leaf set
  - Neighborhood set
- **Routing table** contains $\lceil \log_{2^b} N \rceil$ rows with 2^b columns, where N is the total number of Pastry nodes
  - The entries in row j refer to a node whose ID shares the present node ID only in the first j digits
  - Similar to **Chord's finger table**, it stores links into the ID space
- **Leaf set** is a set of l nodes with numerically closest IDs ($1/2$ larger and $1/2$ smaller than the ID of the current node)
  - Like **Chord's successor list**
- **Neighborhood set** maintains information about nodes that are close together in terms of **network locality**
  - E.g., number of IP hops, Round-Trip Time (RTT) values

Pastry (cont'd)

- **Pastry routing algorithm**:
- The primary goal of the routing algorithm is to quickly locate the node responsible for a particular key
- Pastry routing works as follows:
  1. Given a message with its key, the node first checks its leaf set
  2. If there is a node whose ID is closest to the key, the message is forwarded directly to the node
  3. If the key is not covered by the leaf set, then the node checks the routing table and the message is forwarded to a node that shares a common prefix with the key by at least one more digit
  4. This way, with $\log_{2^b} N$ steps, the message can reach its destination node
- Thus, the number of routing steps in Pastry is at the order of $O(\log N)$

Pastry (cont'd)

- $b = 4$; base $2^b = 16$; $N = 10,000$ nodes; $\lceil \log_{16} 10,000 \rceil = 4$ rows; node 63AB looks up key EB3E
  - From its routing table, node 63AB gets node E123, which shares 1-digit common prefix with the key
  - Node E123 checks its routing table and gets node EB17, which shares 2-digit common prefix with the key
  - Node EB17 then checks its routing table and gets node EB39, which shares 3-digit common prefix with the key
  - Finally, node EB39 checks its leaf set and forwards the message directly to node EB3E



Pastry (cont'd)

- 63AB → E123 → EB17 → EB39 → EB3E
  - "…" represents arbitrary suffixes in base 16
  - IP address and port number associated with each entry are not shown

(1) Routing table of node 63AB

0...	1...	2...	3...	4...	5...		7...	8...	9...	A...	B...	C...	D...	**E...**	F...
60...	61...	62...		64...	65...	66...	67...	68...	69...	6A...	6B...	6C...	6D...	6E...	6F...
630...	631...	632...	633...	634...	635...	636...	637...	638...	639...		63B..	63C..	63D..	63E..	63F..

(2) Routing table of node **E…** (e.g., E123)

0...	1...	2...	3...	4...	5...	6...	7...	8...	9...	A...	B...	C...	D...		F...
E0...		E2...	E3...	E4...	E5...	E6...	E7...	E8...	E9...	EA...	**EB...**	EC...	ED...	EE...	EF...
E10..	E11..		E13..	E14..	E15..	E16..	E17..	E18..	E19..	E1A..	E1B..	E1C..	E1D..	E1E..	E1F..

(3) Routing table of node **EB…** (e.g., EB17)

0...	1...	2...	3...	4...	5...	6...	7...	8...	9...	A...	B...	C...	D...		F...
E0...	E1...	E2...	E3...	E4...	E5...	E6...	E7...	E8...	E9...	EA...		EC...	ED...	EE...	EF...
EB0..		EB2..	**EB3..**	EB4..	EB5..	EB6..	EB7..	EB8..	EB9..	EBA..	EBB..	EBC..	EBD..	EBE..	EBF..

(4) Leaf set of node **EB3…** (e.g., EB39)

EB30	EB31	EB32	EB33	EB34	EB35	EB36	EB37	EB38		EB3A	EB3B	EB3C	EB3D	**EB3E**	EB3F

Pastry (cont'd)

- **Pastry join/leave mechanisms**:
- Nodes join as follows:
  1. The joining node must know of at least another node already in the system
  2. It generates an ID for itself, and sends a join request to the **known node**
  3. The request will be routed to the node whose ID is numerically closest to the new node ID
  4. All the nodes encountered on route to the destination will send their state tables (routing table, leaf set, and neighborhood set) to the new node
  5. The new node will initialize its own state tables, and it will inform appropriate nodes of its presence

Pastry (cont'd)

- Nodes leave/failure as follows:
  1. Nodes in Pastry may fail or depart without any notice
  2. Routing table maintenance is handled by periodically exchanging **keep-alive** messages among neighboring nodes
  3. If a node is unresponsive for a certain period, it is presumed failed
  4. All members of the failed node's leaf set are then notified and they update their leaf sets
- With concurrent node failures, eventual message delivery is guaranteed unless $l/2$ or more nodes with adjacent IDs fail simultaneously
  - Parameter l is an even integer with typical value of 16

Summary

- Structured overlays use the concept of consistent hashing and are able to locate objects with a cost that is at the order of O(log N), where N is the total number of nodes
- The number of DHT algorithms is huge and continues to grow
- Most variants of DHT-based systems try to optimize:
  - Data lookup cost
  - Routing table size
  - Maintenance cost
  - Fault tolerance

	Chord	**Pastry**
Structure	Circle	Hybrid: circle + tree (similar to the Plaxton's algorithm)
Routing algorithm	Matching key and node ID	Matching key and prefix in node ID
Routing performance	O(log N), where N is the number of nodes	O(log N), where N is the number of nodes

Learning Outcomes

- Things to know:
  - Fundamentals of DHT algorithms
  - How Chord works
  - How Pastry works