Telematics
Chapter 9: Peer-to-Peer Networks

Prof. Dr. Mesut Güneş
Computer Systems and Telematics (CST)
Distributed, embedded Systems
Institute of Computer Science
Freie Universität Berlin
http://cst.mi.fu-berlin.de
Contents

● Design Issues
● Basics of Peer-to-Peer systems
● First Generation P2P-Systems
  ● Napster
  ● Gnutella
  ● Super-Peer-Networks
● Distributed Hash Tables
  ● Concept
  ● Case Study I: Chord
  ● Case Study II: Pastry
● Overlay- vs. physical topology
Design Issues

- Two communication principles
  - Client/Server
  - Peer-to-Peer (P2P)
- Originally the Internet has a P2P design
  - All hosts provide services like
    - FTP, Telnet, Email
- With a growing network a more hierarchical network structure was established
  - Client/Server applications
- Since 2000 a new wave of P2P applications emerged

OSI Reference Model

- Application Layer
- Presentation Layer
- Session Layer
- Transport Layer
- Network Layer
- Data Link Layer
- Physical Layer
Peer-to-Peer: Introduction

- Originally the Arpanet/Internet has a P2P design
  - All hosts provide services like (FTP, Telnet, ...)
  - All hosts were equal
- With increasing number of nodes the structure changed to a Client-Server architecture
Peer-to-Peer: Introduction

- “Opposite” of Client/Server
- No central server ⇒ Information are highly distributed
- Each peer is at the same time server and client (servent)
  - Each peer can query, reply to queries, and forward messages at the same time
- Each peer “speaks” directly with other peers
Peer-to-Peer: Concept

- Peer-to-Peer networks are **overlay-networks**
- The focus is on the **application layer**

- Physical topology of the network
Peer-to-Peer: Some Applications

- There are many P2P Applications today
  - P2P Telephony, e.g., Skype
  - P2P Instant Messaging, e.g., ICQ
  - P2P Content Sharing, e.g., Gnutella, Kazaa, Freenet, BitTorrent
  - P2P Resource Sharing, e.g., SETI@home
  - P2P Social Networks, e.g., Krawler
  - P2P Video Streaming, e.g., P2PTV, PDTP
  - ...

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 9: Peer-to-Peer Networks
Peer-to-Peer: Properties and Classification

- Peer-to-Peer Networks have some interesting properties
  - No single point of failure
  - Very flexible and adaptive (scalability)
  - Self-organized
  - Reduce/eliminate the demand for server clusters
  - Improve utilization of resources (memory, CPU)
  - Information are always up-to-date
  - Provide better anonymity

- Classification of Peer-to-Peer Networks
  - Unstructured: Data are placed randomly
  - Structured: Data are placed systematically, e.g., in a Distributed Hash Table (DHT)
Peer-to-Peer: General Problems

- There are also some problems to solve
  - Retrieving of information
  - Discovery of other peers
  - Message routing
  - Keep network traffic low!
Peer-to-Peer: Overlay Architecture

- Overlay architecture
  - Network communications
    - Network characteristics, how is the node connected
  - Overlay node management
    - Discovery of peers, routing
  - Features management
    - Security, reliability, fault resilience
  - Services-specific level
  - Application level
    - Tools on top of the P2P overlay

Main issue: The lookup problem

Where to store, and how to find a certain data item in a distributed system without any centralized control or coordination?
Peer-to-Peer: The Lookup Problem

I have item X. Where to place X?

I want item X. Where can I find X?
Peer-to-Peer: Overlay Interface to Applications

- Application interface for DHT-based overlay networks
First Generation of P2P-Systems
Napster (1)

- File Sharing as main application
- **Not** a pure Peer-to-Peer network
  - Hybrid system
- Each peer sends the list of shared files to a **central database**
- A **peer asks the central database** when it searches for files
- The central database provides a **list of peers** which store the searched files
- The peer downloads the searched file **directly** from the peer with the best transfer rate (P2P)
Napster (2)

1. Send request to the central DB
2. PING to all peers provided from the central DB
Napster (3)

1. Send request to the central DB
2. PING to all peers provided from the central DB
3. (Peer-to-Peer) File Download
Napster (4)

● **Advantages**
  ● Central database has full knowledge about available items
  ● Very fast retrieval of searched information in the central database
  ● Supports simple and complicated (fuzzy) queries

● **Disadvantage**
  ● High traffic load at the central database
  ● Available capacity of other peers are not really used
  ● Scalability problems
  ● Single point of failure!
    ● Technical as well as **juridical**
First Generation of P2P-Systems

Gnutella
Gnutella: Introduction

- The Gnutella-System
  - Very popular P2P system after Napster
  - **Pure** Peer-to-Peer System
  - No central server
  - Mainly used for file sharing
- Very flexible topology
  - Each node can connect to each other
- Very simple protocol
  - Only 5 different messages
- But
  - No routing-“intelligence” (only Loop-Detection)
  - Messages are always broadcast!!

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>Announce availability and probe for other servents</td>
<td>None</td>
</tr>
<tr>
<td>Pong</td>
<td>Response to a Ping</td>
<td>IP address and port number of the responding servant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number and total kB of files shared</td>
</tr>
<tr>
<td>Query</td>
<td>Search request</td>
<td>Minimum network bandwidth of responding servent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Search criteria</td>
</tr>
<tr>
<td>QueryHit</td>
<td>Returned by servents that have the requested file</td>
<td>IP address, port number, and network bandwidth of responding servent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of results and result set</td>
</tr>
<tr>
<td>Push</td>
<td>File download requests for servents behind firewalls</td>
<td>Servent identifier, Index of requested file, IP address and port to send file to</td>
</tr>
</tbody>
</table>
Gnutella: Routing

1. Query 1
2. Reply 2
3. Query 1
4. Reply 4
5. Reply 5
6. Query 1
7. Reply 7
8. Query 1

Reply 6,7
Reply 4,5
Reply 4
### Gnutella: Generated Traffic in Byte (1)

<table>
<thead>
<tr>
<th>( N )</th>
<th>( T=1 )</th>
<th>( T=2 )</th>
<th>( T=3 )</th>
<th>( T=4 )</th>
<th>( T=5 )</th>
<th>( T=6 )</th>
<th>( T=7 )</th>
<th>( T=8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>166</td>
<td>332</td>
<td>498</td>
<td>664</td>
<td>830</td>
<td>996</td>
<td>1,162</td>
<td>1,328</td>
</tr>
<tr>
<td>3</td>
<td>249</td>
<td>747</td>
<td>1,743</td>
<td>3,735</td>
<td>7,719</td>
<td>15,687</td>
<td>31,623</td>
<td>63,495</td>
</tr>
<tr>
<td>4</td>
<td>332</td>
<td>1,328</td>
<td>4,316</td>
<td>13,28</td>
<td>40,172</td>
<td>120,848</td>
<td>362,876</td>
<td>1,088,960</td>
</tr>
<tr>
<td>5</td>
<td>415</td>
<td>2,075</td>
<td>8,715</td>
<td>35,275</td>
<td>141,515</td>
<td>566,475</td>
<td>2,266,315</td>
<td>9,065,675</td>
</tr>
<tr>
<td>6</td>
<td>498</td>
<td>2,988</td>
<td>15,438</td>
<td>77,688</td>
<td>388,938</td>
<td>1,945,188</td>
<td>9,726,438</td>
<td>48,632,688</td>
</tr>
<tr>
<td>7</td>
<td>581</td>
<td>4,067</td>
<td>24,983</td>
<td>150,479</td>
<td>903,455</td>
<td>5,421,311</td>
<td>32,528,447</td>
<td>195,171,263</td>
</tr>
<tr>
<td>8</td>
<td>664</td>
<td>5,312</td>
<td>37,848</td>
<td>265,600</td>
<td>1,859,864</td>
<td>13,019,712</td>
<td>91,138,648</td>
<td>637,971,200</td>
</tr>
</tbody>
</table>

- **Number of users connected to the Gnutella network:** 2000
- **Parameters**
  - \( N = \) Number of connections open
  - \( T = \) Number of hops
- **Query message length:** 83 byte
- **Only query relaying (no Responses !!)**

Source: "Why Gnutella Can't Scale. No, Really.", Jordan Ritter
### Gnutella: Generated Traffic in Byte (2)

<table>
<thead>
<tr>
<th>T</th>
<th>N=3</th>
<th>T=2</th>
<th>T=3</th>
<th>T=4</th>
<th>T=5</th>
<th>T=6</th>
<th>T=7</th>
<th>T=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>283.68</td>
<td>1,418.4</td>
<td>4,822.56</td>
<td>13,900.3</td>
<td>36,594.7</td>
<td>91,061.3</td>
<td>218,15</td>
<td>508,638</td>
</tr>
<tr>
<td>2</td>
<td>378.24</td>
<td>2,647.68</td>
<td>12,860.2</td>
<td>53,710.1</td>
<td>206,897</td>
<td>758,371</td>
<td>2,688,530</td>
<td>9,306,220</td>
</tr>
<tr>
<td>3</td>
<td>472.8</td>
<td>4,255.2</td>
<td>26,949.6</td>
<td>147,986</td>
<td>753,17</td>
<td>3,658,050</td>
<td>17,214,200</td>
<td>79,185,000</td>
</tr>
<tr>
<td>4</td>
<td>567.36</td>
<td>6,240.96</td>
<td>48,793</td>
<td>332,473</td>
<td>2,105,470</td>
<td>12,743,500</td>
<td>74,798,500</td>
<td>429,398,000</td>
</tr>
<tr>
<td>5</td>
<td>661.92</td>
<td>8,604.96</td>
<td>80,092.3</td>
<td>651,991</td>
<td>4,941,123</td>
<td>35,823,800</td>
<td>252,002,000</td>
<td>1,734,360,000</td>
</tr>
<tr>
<td>6</td>
<td>756.48</td>
<td>11,347.2</td>
<td>122,55</td>
<td>1,160,440</td>
<td>10,242,000</td>
<td>86,526,900</td>
<td>709,521,000</td>
<td>5,693,470,000</td>
</tr>
</tbody>
</table>

- Mean percentage of users, which share content: **30%**
- Mean percentage of users, which can respond to replies: **40%**
- Mean number of responses per responding user: **10**
- Mean length of responses: **60**

- "**Standard client settings yield a whopping 17MB generated in response to a [...] search query**"
Gnutella: Graphs ...

Partial Map of Gnutella Network - 7/27/00

Clip2 Distributed Search Services
http://dss.clip2.com
©2000 Clip2.com, Inc.
Gnutella: Pros & Cons

● **Advantages**
  ● Simple protocol
    ➡ easy to implement
  ● Low maintenance traffic
    ➡ Only periodic PING/PONGs
  ● No restriction on overlay topology

● **Disadvantages**
  ● Message broadcasts lead to network flooding
    ➡ high network traffic
    ➡ poor scalability
First Generation of P2P-Systems
Super-Peers
Super-Peers

- A possible solution of the scalability problem: Super-Peers
  - Concept used in Gnutella 0.6, Kazaa
  - Idea: Nodes with high bandwidth, CPU power, etc. become Super-Peers
    - Super-Peers
    - Normal Peers
  - Only Super-Peers are involved in the **routing process**
  - A “normal” (client) peer connects to a Super-Peer
  - A Super-Peers knows all the contents of its client peers
  - Client peers query their super-peers
  - A Super-Peer may forward a query to other Super-Peers
Super-Peers: Pros & Cons

- **Advantages**
  - Reduction of the search process
  - Fast retrieval of data

- **Disadvantages**
  - High traffic at the Super-Peers
  - How to select Super-Peers?
  - What happens when a Super-Peer breaks down?
  - Content of client peers not always possible to index
    - Database contents
Distributed Hash Tables (DHT)
Distributed Hash Tables (DHT): General

- Efficient Look-Up Services required
  - DHTs guarantee an upper bound for key look-ups
  - DHTs are overlay networks
  - Each node and each object possesses a unique ID (hash key)
  - An object is stored on the node which is responsible for the object ID
  - A node is responsible for an ID if its own ID is the nearest to the object ID
Distributed Hash Tables (DHT): General

- Idea: Overlay topology is not randomly
  - Each node stores in its routing table only the nodes according particular requirements
- All DHTs provide two basic routing operations
  1. lookup(key) → node
  2. put(object, key) → node

- Popular DHTs:
  - Content Addressable Network (CAN)
  - Chord
  - Pastry
  - Tapestry
Distributed Hash Tables (DHT): General

- Each node manages a small number of references to other nodes.
  - $O(\log N)$ references, $N$ number of nodes
- By mapping nodes and data items into a common address space, routing to a node leads to the data items for which a certain node is responsible.
- Queries are routed via a small number of nodes to the target node.
  - An item can be located by routing via $O(\log N)$ hops.
- Identifiers are distributed on the nodes nearly equally.
  - Load for retrieving items may be balanced among the nodes.
- No node plays a particular role.
  - Formation of hot spots or bottlenecks can be avoided.
  - Departure of a node does not have considerable effects on the DHT.
  - DHTs are considered robust against random failures and attacks.
- Distributed index provides a definitive answer about results.
  - If a data item is stored in the system, the DHT guarantees that the data is found.
Distributed Hash Tables (DHT): General

I want $D$! $H(\cdot D\cdot)=3107$
Distributed Hash Tables (DHT): General

Logical view of the Distributed Hash Table

Mapping on the real topology
Distributed Hash Tables (DHT): General

(a) Direct Storage

(b) Indirect Storage
Distributed Hash Tables (DHT)

Chord
Case Study I: Chord (1)

- The IDs are arranged in a **virtual** ring from 0 – (2\(m\) -1)
- Consistent hashing is used
  - The addition and removal of slots into the hash does not significantly change the mapping of the keys
- Consistent Hashing (e.g. SHA-1) assigns each **node** (e.g. peer) and each **object** a \(m\)-bit ID in the same overlay ID ring
  - For nodes the IP address is hashed
  - For data the data key (e.g. file name) is hashed
- New nodes take their position in the ring according their ID
- The key \(k\) is assigned to the first node with ID \(\geq k\)
  - \(\text{successor}(k)\)
- Chord provides an upper bound of \(O(\log N)\) for the key look-up
  - \(N\) is the number of peers in the system
Case Study I: Chord (2)

successor(0) = 0
successor(1) = 1
successor(2) = 3
successor(6) = 6

successor(14) = 15
successor(10) = 13
successor(9) = 9
successor(3) = 4
successor(5) = 1
Case Study I: Chord (3)

- Each node $n$ has a routing table with max. $m$ entries
  - Finger table
- The $i$-th entry = first node $s$ with an ID at least $2^{i-1}$ larger than $n$’s ID
  - $s = \text{successor}(\text{node.ID} + 2^{i-1})$
- Peer $s$ is the i-th finger of peer node
Case Study I: Chord (4)
Case Study I: Chord (5)
Case Study I: Chord – Routing (1)

- **Routing:** if node $n$ searches for the key $k$
  - If successor($k$) unknown, forward query to next node $n'$ in the finger table
  - If $n'$ does not know the successor($k$), $n'$ forwards the query to next node in its finger table
  - This is done until the successor($k$) is found

- **Idea:** in each step, the distance to the successor($k$) is halved
  - $O(\log N)$ many (overlay-) hops
  - Chord finds a key in $O(\log N)$
Case Study I: Chord – Routing (2)

<table>
<thead>
<tr>
<th>i</th>
<th>succ</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>succ</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Case Study I: Chord – Stabilize

- Each node $n$ performs a stabilize-procedure to correct possible network anomalies
  - Node $n$ asks its successor for its predecessor $p$
  - Node $n$ tests, whether $p$ has to be its successor, i.e., $p$ is between $n$ its successor
- Node $n$ updates periodically one finger $f$ which is selected randomly.
  - For this node $n$ searches for successor(node.ID + $2^{x-1}$) and updates it with the new successor(node.ID + $2^{x-1}$)
Chord-based Applications

- On top of Chord many applications can be realized
  - Cooperative File System (CFS)
    - Multiple providers of content cooperate to store data
    - The total load is spread over all participants
  - Chord-based DNS
    - Distributed DNS lookup service
    - Host names are keys and IP address are values
Distributed Hash Tables (DHT)
Pastry
Case Study II: Pastry – Introduction

- Similar concept as Chord
  - Nodes are organized in a virtual ID ring from 0 to $2^{128} - 1$
  - Each node and object is assigned a 128 bit key (hash key)
  - Each ID is sequence of numbers in the base $2^b$ ($b=1$ binary, $b=4$ hex)
- But:
  - The routing is based on prefix matching
  - Considers the topological distance
  - Each node maintains
    - a routing table
    - a leaf set
Case Study II: Pastry – Routing Table (1)

- Routing table
  - $\log_2 N$ rows
  - Each row has $2^b - 1$ entries
  - Each entry in row $n$ has the same $n$ digit prefix as the current node
  - Each entry in row $n$ differs on the $(n+1)$-th position from the current node
  - Each of these entries has one of the $2^b - 1$ digits on the $(n+1)$-th position
Case Study II: Pastry – Routing-Table (2)

- Routing table
- NodeID **34035761**, \( b = 3 \)

<table>
<thead>
<tr>
<th>row</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03761261</td>
<td>13541241</td>
<td>27235106</td>
<td></td>
<td>45521251</td>
<td>53610176</td>
<td>65547772</td>
<td>71154601</td>
</tr>
<tr>
<td>1</td>
<td>30771255</td>
<td>31521225</td>
<td>32236556</td>
<td>33400741</td>
<td></td>
<td>35716236</td>
<td>36124163</td>
<td>37715026</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>34102724</td>
<td>34203162</td>
<td>34303735</td>
<td>34404216</td>
<td>34576555</td>
<td>34653437</td>
<td>34734152</td>
</tr>
<tr>
<td>3</td>
<td>34006624</td>
<td>34012773</td>
<td>34025354</td>
<td></td>
<td>34045631</td>
<td>34050032</td>
<td>34064645</td>
<td>34076214</td>
</tr>
<tr>
<td>4</td>
<td>34030342</td>
<td>34031643</td>
<td>34032346</td>
<td>34033652</td>
<td>34034040</td>
<td></td>
<td>34036123</td>
<td>34037437</td>
</tr>
<tr>
<td>5</td>
<td>34035073</td>
<td>34035152</td>
<td>34035031</td>
<td></td>
<td>34035070</td>
<td></td>
<td>34035667</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>34035701</td>
<td>34035717</td>
<td></td>
<td></td>
<td>34035740</td>
<td></td>
<td></td>
<td>34035774</td>
</tr>
</tbody>
</table>
Case Study II: Pastry – Neighborhood Set & Leaf Set

- Leaf set
  - Contains the $L$ closest nodes
  - $L/2$ smaller nodes
  - $L/2$ larger nodes
- Typical values for $|L|$ are $2^b$ or $2 \times 2^b$
### Case Study II: Pastry – Node State

**Routing Table**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03761261</td>
<td>13541241</td>
<td>27235106</td>
<td>45521251</td>
<td>53610176</td>
<td>65547772</td>
<td>71154601</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>30771255</td>
<td>31521225</td>
<td>32236556</td>
<td>33400741</td>
<td>35716236</td>
<td>36124163</td>
<td>37715026</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>34102724</td>
<td>34203162</td>
<td>34303735</td>
<td>34404216</td>
<td>34576555</td>
<td>34653437</td>
<td>34734152</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>34006624</td>
<td>34012773</td>
<td>34025354</td>
<td>34045631</td>
<td>34050032</td>
<td>34064645</td>
<td>34076214</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>34030342</td>
<td>34031643</td>
<td>34032346</td>
<td>34033652</td>
<td>34034040</td>
<td>34036123</td>
<td>34037437</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34035073</td>
<td>34035152</td>
<td>34035031</td>
<td>34035070</td>
<td>34035667</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>34035701</td>
<td>34035717</td>
<td></td>
<td>34035740</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Local Overlay ID: 34035761**

**Leaf Set**

<table>
<thead>
<tr>
<th>smaller</th>
<th>larger</th>
</tr>
</thead>
<tbody>
<tr>
<td>34035667</td>
<td>34035774</td>
</tr>
<tr>
<td>34035701</td>
<td>34036012</td>
</tr>
<tr>
<td>34035717</td>
<td>34036042</td>
</tr>
<tr>
<td>34035740</td>
<td>34036076</td>
</tr>
</tbody>
</table>
Case Study II: Pastry – Routing (1)

- **Routing in Pastry**
  - Node $n$ searches the leaf set for nodes which are responsible for key $k$
  - If the leaf node $l$ is responsible for key $k$, the query is addressed directly to node $l$
  - Otherwise the routing table is searched for a node with larger prefix matching (at least one more position) as node $n$
  - If such a node does not exist the leaf set is searched for a node which is closer to $k$

- This process is repeated in each node until the query arrives at the node which is responsible for key $k$
  - Intuitive: with each routing step the matching prefix grows for one position
    - $O(\log_2 b N)$ (Overlay-) routing effort
Case Study II: Pastry – Routing (2)
Overlay- vs. Physical Topology
Overlay vs Physical Topology (1)

- DHTs can route very efficient in the overlay network
- However, there is no relationship between the overlay- and physical topology
  - Neighbors in the overlay network are rarely neighbors on the physical network
  - DHTs may show high **Overlay Stretch**
    - The ratio of the accumulated physical route during the overlay routing for the search of a key and the real path length between source and destination nodes

\[
\text{Overlay Stretch} = \frac{\text{Real path length}}{\text{Path length in overlay network}}
\]
Overlay vs Physical Topology (2)

Overlay Stretch = 17/2
Overlay vs Physical Topology (3)

- Several methods to reduce the overlay stretch
  - **Proximity Neighbor Selection (PNS)** in Pastry
  - Each Pastry-node selects periodically a random entry from row i from its routing table
  - This random node replies with the i-th row from its routing table
  - After the response, the node computes pair wise the distance to the own entries and the entry from the response (e.g. PING/PONG)
  - If needed the own entry is replaced by the entry from the response if it is physically closer than the own entry (e.g. hop count, round trip time)
Overlay vs. Physical Topology (4)

- **Landmarking**
  - Fixed set of Landmark-Nodes
  - Each node measures periodically its distance to all Landmark-Nodes
    - Sequence of the Landmark-Nodes
  - **Idea:** Nodes with the same Landmark-Distance-Sequence are with high probability physically close to each other
    - Nodes with same Landmark-Distance-Sequence assign each other numerical close overlay Ids
  - Example: 16 Landmark-Nodes in a Pastry-Network with hex IDs
  - Nodes with the same Landmark-Distance-Sequence assign to each other the same overlay-ID-prefix
### Summary

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

<table>
<thead>
<tr>
<th>Unstructured P2P</th>
<th>Structured P2P</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1st Generation</strong></td>
<td><strong>2nd Generation</strong></td>
</tr>
<tr>
<td>Centralized P2P</td>
<td>Pure P2P</td>
</tr>
</tbody>
</table>
| 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 |
| Hybrid P2P | DHT-Based |
| 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 |
Summary

- The Peer-to-Peer principle is not really new
- However, with the growth of the Internet and emerging of new applications a new wave of P2P applications was created
- A P2P network is a overlay network on the Application Layer
- In a P2P network all peers have the same role
  - Server + Client = Servent