Peer-to-peer systems

INF 5040 autumn 2015

lecturer: Roman Vitenberg

Motivation for peer-to-peer

- Inherent restrictions of the standard client/server model
  - Centralised design lacks scalability & fault-tolerance
    - Processing
    - Network traffic
- P2P systems take care of distributing processing load and network traffic between all nodes that participate in a distributed information system
  - Solve the bottleneck but must pay in the form of considerably more complex mechanisms and lack of control

What is P2P?

- In a P2P system, each participating node behaves as both client and server, and “pays” for participation by offering access to some of its own resources
  - Typically processing power and storage resources
  - But it can also be a logical resource (a service)
- An application-level network on top of the Internet (overlay network)

Essential characteristics of P2P systems

- Each participant contributes its own resources to the system
- All nodes have the same functional capabilities and responsibility
- No dependency on a central entity for administration of the system (self-organising)
- The effectiveness critically depends on algorithms for data placement over many nodes and for subsequent access to them
- Unpredictable availability of processes and nodes

The evolution of P2P systems and applications

- First generation
  - Napster
    - Sharing/exchange of music files
    - Hybrid Client/Server og P2P (central index server)
- Second generation
  - Gnutella, Freenet, Kazaa, BitTorrent, eMule, Kademlia...
    - Decentralised file-sharing system
- Third generation
  - P2P middleware
    - Application-independent middleware layer for management of distributed resources in the global scale
    - Pastry, Tapestry, CAN, Chord, ...

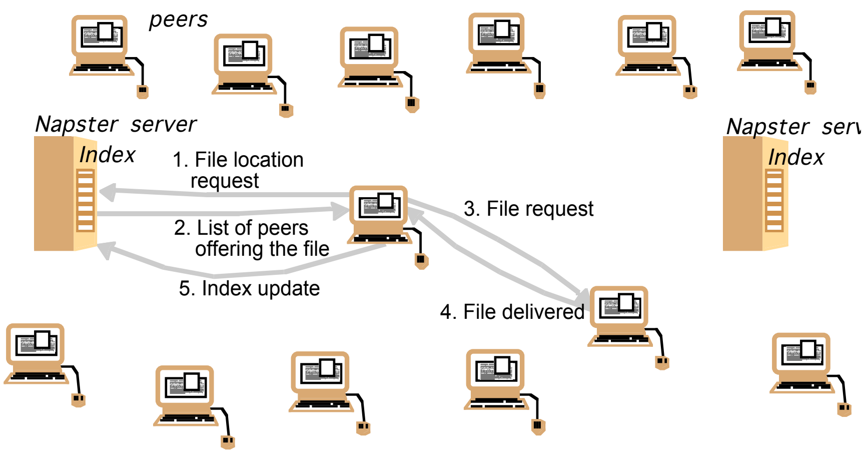
P2P middleware characterisation

- The main objectives are to
  - Place resources (data objects and files) on participating nodes that are widely spread over the Internet
  - Route messages to them on behalf of the clients
  - Hide the location of resources from the clients (transparency)
- Provide performance guarantees (number of hops, etc.)
- Place resources in a structured fashion to satisfy requirements of availability, trust, load-balancing and locality
- Resources are identified by GUIDs (derived from “secure digest function” – see the textbook chapter 7.4.3).
  - Randomised distribution of resources over nodes in different organisations in the entire world

The difference between IP and overlay routing for P2P applications

	IP	*Application-level routing overlay*
Scale	IPv4 is limited to 2^{32} addressable nodes. The IPv6 name space is much more generous (2^{128}), but addresses in both versions are hierarchically structured and much of the space is pre-allocated according to administrative requirements.	Peer-to-peer systems can address more objects. The GUID name space is very large and flat (>2^{128}), allowing it to be much more fully occupied.
Load balancing	Loads on routers are determined by network topology and associated traffic patterns.	Object locations can be randomized and hence traffic patterns are decoupled from the network topology.
Network dynamics (addition/deletion of objects/nodes)	IP routing tables are updated asynchronously on a best-efforts basis with time constants on the order of 1 hour.	Routing tables can be updated synchronously or asynchronously with fractions of a second delays.
Fault tolerance	Redundancy is designed into the IP network by its managers, ensuring tolerance of a single router or network connectivity failure. *n*-fold replication is costly.	Routes and object references can be replicated *n*-fold, ensuring tolerance of *n* failures of nodes or connections.
Target identification	Each IP address maps to exactly one target node.	Messages can be routed to the nearest replica of a target object.
Security and anonymity	Addressing is only secure when all nodes are trusted. Anonymity for the owners of addresses is not achievable.	Security can be achieved even in environments with limited trust. A limited degree of anonymity can be provided.

Napster



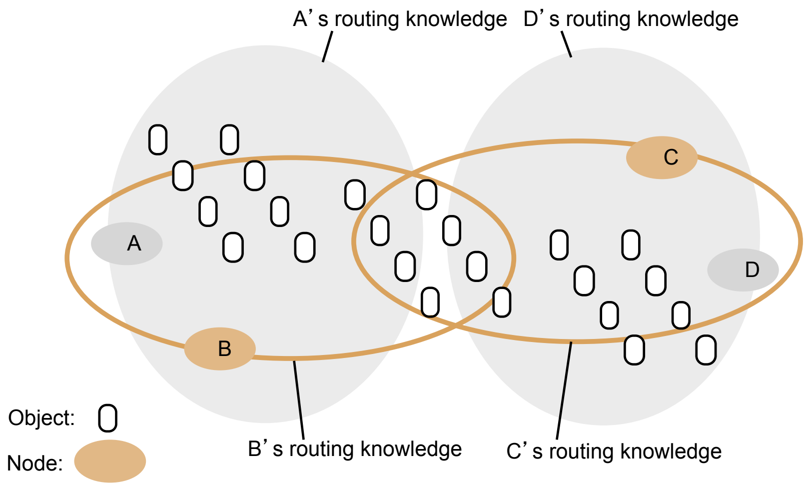
P2P middleware (1 of 2)

- Challenge: offer a mechanism that gives fast and reliable access to resources in a location-transparent fashion
- Functional requirements
  - Facilitate construction of services that are implemented over many nodes in a distributed network
    - Make it possible to locate and communicate with all available resources
    - Possible to add new resources and remove old ones
    - Possible to add new nodes and remove old ones
    - Simple application- and resource-independent API

P2P middleware (2 of 2)

- Non-functional requirements
  - Global scalability
  - Load-balancing
  - Optimisation for local interaction between neighbour peers
  - Coping with high node and object "churn"
  - Security of data in an environment with heterogeneous trust
  - Anonymity and resilience to censorship

Distribution of information in a "routing overlay"



Routing overlay

- Application-level algorithm that locates nodes and stored data objects (independently of network routing)
- Possible to implement at the middleware level
- Ensures that each node can access every object by routing requests through a sequence of nodes and exploiting the knowledge of each of them to locate the object
- Responsible for managing the lifecycle of objects and nodes

Essential API for a Distributed Hash-Table (Pastry)

- Object GUID is derived from all or part of its state using a secure digest function (e.g., SHA-1).
- GUIDs are used to place objects and to locate them (hence called distributed hash-table)

put(GUID, data)
The *data* is stored in replicas at all nodes responsible for the object identified by *GUID*.
remove(GUID)
Deletes all references to *GUID* and the associated data.
value = get(GUID)
The data associated with *GUID* is retrieved from one of the nodes responsible it.

Case study: Pastry

- Nodes and objects are assigned a 128-bit GUID
  - By applying a secure digest function on node "public key" and object name or (part of) its state
- In a network with N nodes, Pastry routing algorithm delivers a message addressed to any GUID in O(log(N)) steps
  - If the GUID maps to an active node, the message is delivered to it. Otherwise, the message is delivered to the node with numerically closest GUID.
- Fully self-organising
  - O(log(N)) messages when a participant joins, leaves, or fails

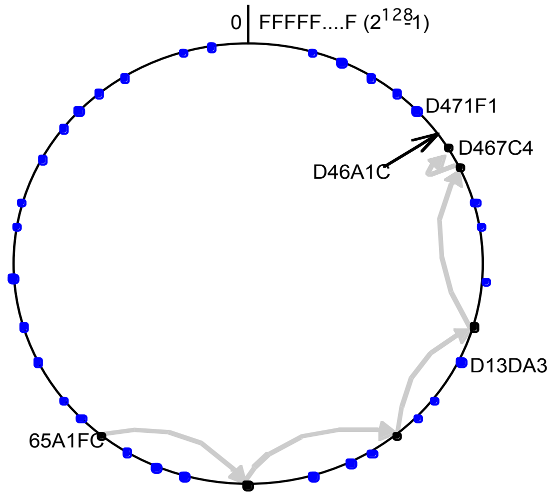
Routing algorithm in Pastry

- Includes two mechanisms:
  - Simple routing mechanism that uses information about neighbours that provides correct routing but may be inefficient
  - More complex mechanism that efficiently routes requests to an arbitrary node (using at most O (log(N)) messages) but that may be temporarily unreliable during periods of instability

Routing algorithm in Pastry: using the leaf set

- Each active node maintains an array L ("the leaf set") of length 2l, that includes GUID and IP addresses of the nodes with numerically closest GUID
  - l predecessor nodes
  - l successor nodes
- Pastry maintains L in presence of node joins, leaves, and failures

Circular routing: Correct but inefficient



The dots depict live nodes. The space is considered circular: node 0 is adjacent to node (2^{128}-1). The diagram illustrates the routing of a message from node 65A1FC to D46A1C using leaf set information alone, assuming leaf sets of size 8 (l = 4).

Routing algorithm in Pastry: using the routing table

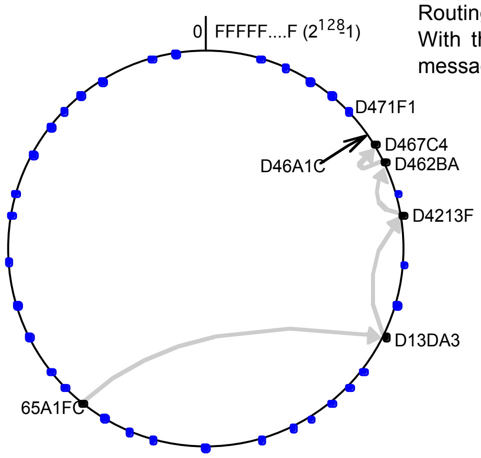
- Improves the "leaf set" algorithm
- Every Pastry node maintains a tree-structured routing table that includes GUIDs and IP-addresses for some nodes spread over all the address space of GUID values.
- The table is not uniform:
  - Dense coverage of GUIDs that are numerically close to the node own GUID
  - Density decreases with distance from the node

Example: first four rows in a Pastry routing table

p =	GUID prefixes and corresponding nodehandles n															
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	n	n	n	n	n	n		n	n	n	n	n	n	n	n	n
1	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6F	6E	6F
	n	n	n	n	n		n	n	n	n	n	n	n	n	n	n
2	650	651	652	653	654	655	656	657	658	659	65A	65B	65C	65D	65E	65F
	n	n	n	n	n	n	n	n	n	n		n	n	n	n	n
3	65A0	65A1	65A2	65A3	65A4	65A5	65A6	65A7	65A8	65A9	65AA	65AB	65AC	65AD	65AE	65AF
	n		n	n	n	n	n	n	n	n	n	n	n	n	n	n

The routing table is located at a node whose GUID begins 65A1. Digits are in hexadecimal. The *n* s represent [GUID, IP address] pairs specifying the next hop to be taken by messages addressed to GUIDs that match each given prefix. Grey-shaded entries indicate that the prefix matches the current GUID up to the given value of *p*: the next row down or the leaf set should be examined to find a route. Although there are a maximum of 128 rows in the table, only $\log_{16} N$ rows will be populated on average in a network with *N* active nodes.

Pastry routing example



Routing a message from node 65A1FC to D46A1C. With the aid of a well-populated routing table the message can be delivered in ~ $\log_{16}(N)$ hops.

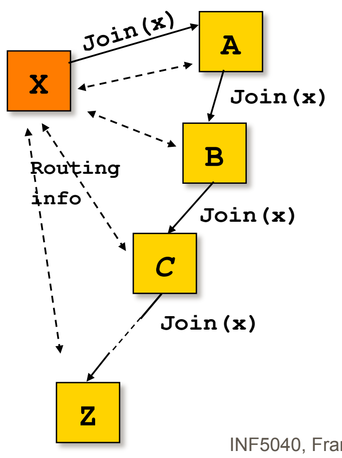
Pastry routing algorithm

When node A receives message M addressed to GUID D
(R[p,i] is the element of the routing table at row p, column i)

```
if L-1 < D < Ll {          // the destination is within the leaf set
        Forward M to leaf set element with GUID closest to D
} else { // use the routing table
        Find p, the length of the longest common prefix of D and A
            and i, the (p+1)th hexadecimal digit of D
        if (R[p,i] ≠ null) {
                Forward M to R[p,i] // common-prefix routing
        } else { // there is no entry in the routing table
                Forward M to any node in R or L that is numerically closer to D
                than A
        }
}
```


Pastry: addition of a new node

- Join protocol that constructs the routing table & "leaf set"




```
X: new node to join
A: a node that X knows
Z: GUID closest numerically to X
   (routed in the usual way)
B, C, ...: nodes the join message
   is routed via
A, Z, B, C, ... transmits relevants
parts of their routing tables and
leaf sets to X. X uses this info
to build its initial routing table
and leaf set.
X then transmits its routing table
and leaf set to A, Z, B, C, ...
```

Pastry: handling leaves and failures

- Pastry node is considered failed when its immediate neighbours (in the GUID space) cannot communicate with it any longer
  - All nodes send 'heartbeat' messages to neighbour nodes (in their own leaf set)
- When it occurs, it is necessary to repair all leaf sets that include GUID of the node that left or failed
  - A node repairs its "leaf set" L by asking a node close to the failed one to send its "leaf set" L', removing the failed node, and adding a node from L'
- Routing tables are repaired "upon discovery" (when a routing request fails)

Pastry: fault-tolerance and reliability

- Routing failure may occur
  - Because of delays in spreading the info about failed nodes
- A Pastry application should retransmit routing requests in absence of response
  - In the meantime, the failure can possibly become repaired
- Randomisation of routing choice (line 6 in the routing algorithm)
  - In some cases, choose a node in R[p,j] instead of R[p,i] (routing choice that occasionally diverges from the standard algorithm)
  - If some node blocks the route, a different path will be chosen sooner or later due to retransmissions
- MSPastry: extension of Pastry with additional dependability mechanisms
  - Ack after each hop in the routing algorithm and selection of an alternative route upon timeout
  - "heartbeat"-messages
  - Other miscellaneous improvements

Evaluation (MSPastry)

- Based on simulations [Castro et al 2004]
  - Good performance and high reliability with thousands of nodes
  - Gracefully degrading as the failure rate increases
- Reliability
  - Upon 0% loss rate of IP-messages, MSPastry was not delivering 1,5 of 100.000 routing requests; none were delivered to a wrong node
  - Upon 5% loss rate of IP-messages, MSPastry was not delivering 3,3 of 100.000 routing requests, and 1,5 of 100.000 were delivered to a wrong node
- Performance
  - Measured relative delay penalty: a ratio between the delay of request delivery via MSPastry and the corresponding delay when using UDP/IP
  - Relative delay penalty varies between about 1,8 (0% loss rate of IP-messages) and about 2,2 (5% loss rate of IP-messages)
- Overhead
  - Control-traffic accounts for approximately 2 messages per minute per node in the long run (initial cost of "setup" is relatively high)

Example of a Pastry-based application: Squirrel

- Web-caching system that makes use of storage and computational resources that are already available on desktop-machines in a local network
- GUID: applying SHA-1 on the URL gives a 128 bits Pastry-GUID.
- The node whose GUID is numerically closest to the calculated GUID becomes the "home node" for the object
- The home node is responsible for maintaining a cached copy of the object (acts as a proxy-server for this object)
- Client nodes use Squirrel to route GET or cGET requests to the home node of the web object
- Evaluation shows that the performance is comparable with the performance of a typical centralised cache (measurements including (1) reduction in the use of extern bandwidth, (2) latency perceived by the user, (3) storage and processing load on client nodes)

Summary

- P2P systems distribute processing load and network traffic between all nodes that participate in the system
- P2P systems are not dependent on a central entity for administration of the system (and self-organisation)
- The effectiveness critically depends on algorithms for placement of data over many nodes and for subsequent access to the data
- P2P middleware is an application-independent software layer that implements a ”routing overlay”
- Study and evaluation of an implementation: Pastry
- A Pastry-based application: Squirrel web-cache