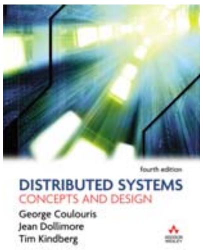
Chapter 10:
Peer-to-Peer Systems



From Coulouris, Dollimore and Kindberg
Distributed Systems:
Concepts and Design
Edition 4, © Addison-Wesley 2005

Introduction

- To enable the sharing of data and resources on a very large scale without any separately-managed servers and infrastructure
- P2P applications
  - Applications that exploit resources available at the edges of Internet – storage, cycles, content, human presence

Characteristics

- Their design ensures that each user contributes resources to the system
- Although they may differ in the resources that they contribute, all the nodes in a P2P system have the same functional capability and responsibilities
- Their correct operation does not depend on the existence of any centrally-administered systems
- They can be designed to offer a limited degree of anonymity to the providers and users of resources
- A key issue for their efficient operation is the choice of an algorithm for the placement of data across many hosts and subsequent access to it in a manner that balances the workload and ensures availability without adding undue overloads

It is the time

- Powerful personal computer
- Always-on, broadband connections

Three generations

- 1st
  - Napster music exchange service
- 2nd
  - Offering greater scalability, anonymity and fault tolerance
    - Freenet, Gnutella, Kazaa, BitTorrent
- 3st
  - P2P middleware
    - Pastry, Tapestry, CAN, Chord, Kademlia
  - Designed to place resources on a set of computers widely distributed throughout the Internet and to route messages to them on behalf of clients
    - Relive clients of decisions about placing resources and holding information
    - Provide guarantees of delivery for requests in a bounded number of hops

Globally Unique Identifiers (GUIDs)

- Resources are identified by GUIDs that are derived as a secure hash from resource's state
- Hash makes a resource "self certifying"
  - Client can check the validity of resources
  - Good for immutable objects

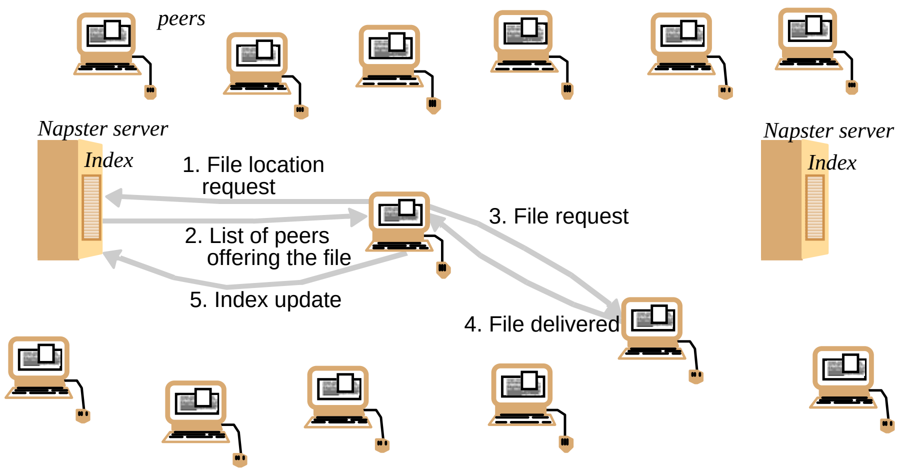
Figure 10.1: Distinctions between IP and overlay routing for peer-to-peer applications

	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 divorced 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

- Centralized indexes but users supplied the files
- Clients are expected to add their own music files to the pool of shared resources by transmitting a link to the Napster indexing service for each available file
  - Fulfill "shared resources at the edges of the Internet"
- Shut down as a result of legal proceedings

Figure 10.2: Napster: peer-to-peer file sharing with a centralized, replicated index



Resource Location Problem

- A key problem is to provide a mechanism to enable clients to access data resources quickly and dependably
  - Napster maintained a unified index of available files
  - 2nd P2P file systems employ partitioned and distributed indexes, but the algorithm used are specific to each system
- This location problem existed in several services that predate the P2P paradigm
  - E.g. NFS addresses this need with the aid of VFS

P2P Middleware

- P2P middleware is designed to meet the need for the automatic placement and subsequent location of distributed objects
- Functional requirements
  - To simplify the construction of services that are implemented across many hosts in a widely distributed network
    - It must enable clients to locate and communicate with any individual resource made available to a service
  - Add and remove resources, and add and remove hosts to services
  - Offer a simple programming interface to application programmers that is independent of the types of distributed resource that the application manipulates

P2P Middleware, cont.

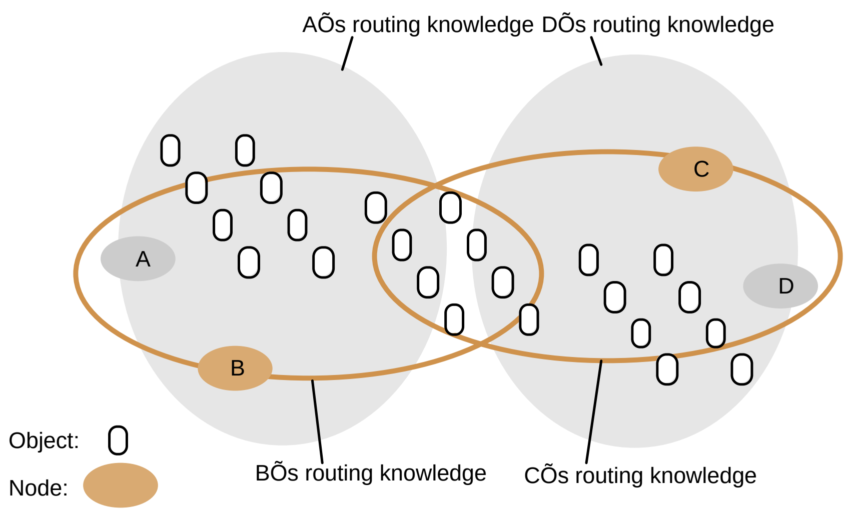
- Non-functional requirements
  - Global scalability
  - Load balancing
  - Optimization for local interactions between neighboring peers
  - Accommodating to highly dynamic host availability
  - Security of data in an environment with heterogeneous trust
  - Anonymity, deniability and resistance to censorship

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Routing Overlays

- ⌘A distributed algorithm takes responsibility for locating nodes and objects
- ⌘The name denotes the fact that the middleware takes the form of a layer that is responsible for routing requests from any client to a host
- ⌘The objects may be placed and subsequently relocated to any node in the network without client involvement
- ⌘It is termed an overlay since it implements a routing mechanisms in the application layer

Figure 10.3: Distribution of information in a routing overlay



Main Task

- A client wishing to invoke an operation on an object submits a request including the object's GUID to the routing overlay, which routes the request to a node at which a replica of the object resides.

Other Tasks

- A node wishing to make a new object available to a P2P service computes a GUID for the object and announces it to the routing overlay, which then ensures that the object is reachable by all other clients
- When clients request the removal of objects from the service the routing overlay must make them unavailable
- Nodes may join and leave the service
  - When joining, the routing overlay arranges for it to assume some of the responsibilities of the other nodes
  - When leaving, its responsibilities are distributed amongst the other nodes

Programming Interfaces

- Distributed hash table (DHT)
  - GUID is used to determine the placement of objects and retrieve them
  - Via *put()*, DHT takes responsibility for choosing a location for it, storing it and providing access to it via *get()*
- Distributed object location and routing (DOLR)
  - Objects may be replicated and stored with the same GUID at different hosts
  - DOLR is responsible for maintaining a mapping between GUIDs and the addresses of the nodes having the replicas
  - Routing overlay takes responsibility for routing requests to the nearest available replica

Figure 10.4: Basic programming interface for a distributed hash table (DHT) as implemented by the PAST API over Pastry

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.

Figure 10.5: Basic programming interface for distributed object location and routing (DOLR) as implemented by Tapestry

publish(GUID)
GUID can be computed from the object (or some part of it, e.g. its name). This function makes the node performing a *publish* operation the host for the object corresponding to *GUID*.
unpublish(GUID)
Makes the object corresponding to *GUID* inaccessible.
sendToObj(msg, GUID, [n])
Following the object-oriented paradigm, an invocation message is sent to an object in order to access it. This might be a request to open a TCP connection for data transfer or to return a message containing all or part of the object's state. The final optional parameter *[n],* if present, requests the delivery of the same message to n replicas of the object.

Pastry

- Nodes and objects are assigned 128-bit GUIDs
  - Nodes are computed by applying a secure hash function to the public key with each node is provided
  - Objects like files are computed applying a secure hash function to the object's name or some part the object's stored state
  - The resulting GUID are randomly distributed in the range 0 to 2^{128}-1

Pastry, cont.

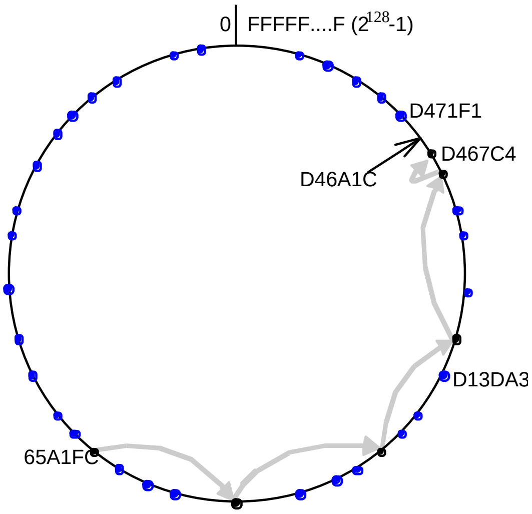
- Pastry routing algorithm will correctly route msg addressed to any GUID in *O(log N)* steps for *N* participating nodes
  - If the node is not active, the msg will be delivered to the active whose numerically closet to it
  - Active nodes take responsibility for processing requests addressed to all objects in their numerical neighborhood

Pastry routing algorithm stage 1: leaf set

- Each node has a leaf set
  - A vector L of 2l nodes whose GUIDs are l above and l below of its own
- Any node that receives a message M with destination D routes M by comparing D with its own and leaf nodes' GUIDs and forwarding M to the node amongst them that is closet to D
  - Easy to conclude at each step M is forwarded to a node that is closer to D than the current node
  - Requires ~N/2 hops to deliver M in a network with N nodes

Figure 10.6: Circular routing alone is correct but inefficient

Based on Rowstron and Druschel [2001]



The dots depict live nodes. The space is considered as 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). This is a degenerate type of routing that would scale very poorly; it is not used in practice.

Pastry routing algorithm stage 2: tree structured routing table

- Each node maintains a tree-structured routing table for nodes spread throughout the entire range of 2^{128} GUIDs, with increased density of coverage for GUIDs closet to its own.

Figure 10.7: First four rows of 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.

Figure 10.8: Pastry routing example Based on Rowstron and Druschel [2001]

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.

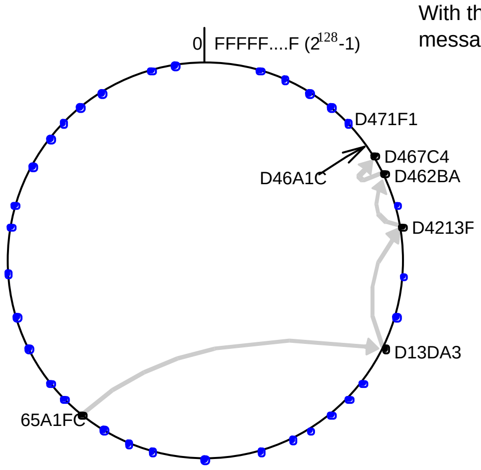


Figure 10.9: Pastry's routing algorithm

To handle a message M addressed to a node D (where $R[p,i]$ is the element at column i, row p of the routing table):

1. If ($L_{-l} < D < L_l$) { // the destination is within the leaf set or is the current node.
2. Forward M to the element L_i of the leaf set with GUID closest to D or the current node A.
3. } else { // use the routing table to despatch M to a node with a closer GUID
4. find p, the length of the longest common prefix of D and A. and i, the $(p+1)^{th}$ hexadecimal digit of D.
5. If ($R[p,i]$? *null*) forward M to $R[p,i]$ // route M to a node with a longer common prefix.
6. else { // there is no entry in the routing table
7. Forward M to any node in L or R with a common prefix of length i, but a GUID that is numerically closer.

}

}

Tapestry

- Tapestry implements a distributed hash table (DHT) and routes message using prefix routing similar to Pastry.
- Tapestry's API conceals the DHT from applications behind service interfaces
- Nodes that hold resources use *publish*(GUID) primitive to make them known to Tapestry
- Replicated resources are published with the same GUID by each node that holds a replica, resulting in multiple entries in the Tapestry routing structure.
  - Thus, Tapestry can place replicas close to frequent users of resources.
  - Note that Pastry can achieve similar flexibility by making the objects with GUIDs close to frequent users

Tapestry, cont.

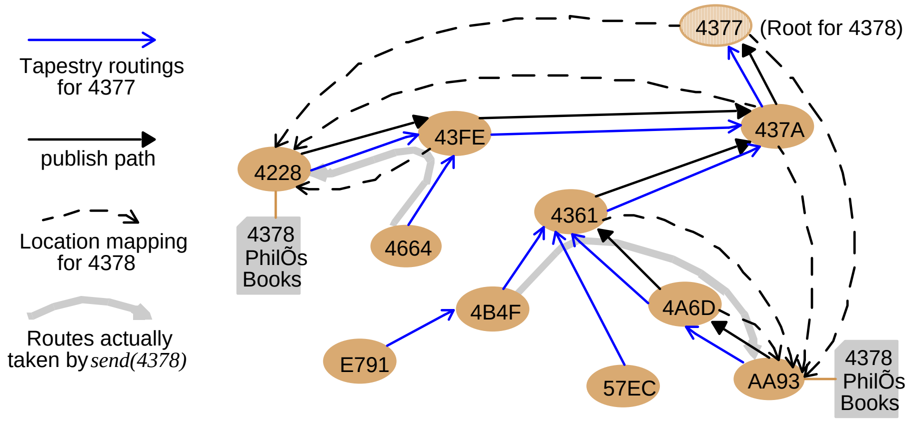
- For each resource G, there is a unique root node R_G that is closest to G.
- Host H holding replicas of G periodically invoke *publish*(G) to ensure that newly arrived hosts become aware of the existence of G.
- On each invocation of *publish*(G), a publish message is routed from the invoker towards R_G.
- On receipt of a publish message, R_G store (G, IP_H) in its routing table
  - Each node along the publication path caches (G, IP_H) too

Tapestry, cont.

- When nodes hold multiple (G, IP_H), they are sorted by the network distance to the IP address.
  - This results in the selection of the nearest available replica

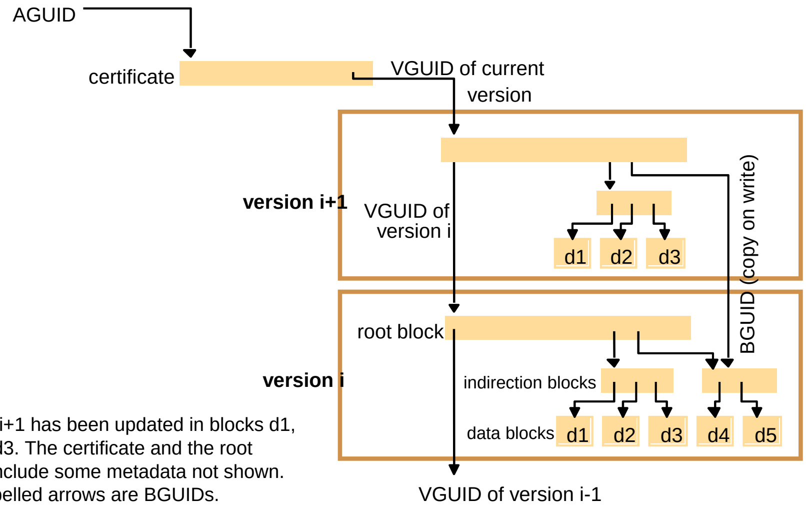
Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Figure 10.10: Tapestry routing From [Zhao et al. 2004]



Replicas of the file *PhilÕs Books*(G=4378) are hosted at nodes 4228 and AA93. Node 4377 is the root node for object 4378. The Tapestry routings shown are some of the entries in routing tables. The publish paths show routes followed by the publish messages laying down cached location mappings for object 4378. The location mappings are subsequently used to route messages sent to 4378.

Figure 10.11: Storage organization of OceanStore objects



Version i+1 has been updated in blocks d1, d2 and d3. The certificate and the root blocks include some metadata not shown. All unlabelled arrows are BGUIDs.

Figure 10.12: Types of identifier used in OceanStore

Name	*Meaning*	*Description*
BGUID	block GUID	Secure hash of a data block
VGUID	version GUID	BGUID of the root block of a version
AGUID	active GUID	Uniquely identifies all the versions of an object

Figure 10.13: Performance evaluation of the Pond prototype emulating NFS

	LAN		*WAN*		*Predominant operations in benchmark*
Phase	*Linux NFS*	*Pond*	*Linux NFS*	*Pond*	
1	0.0	1.9	0.9	2.8	Read and write
2	0.3	11.0	9.4	16.8	Read and write
3	1.1	1.8	8.3	1.8	Read
4	0.5	1.5	6.9	1.5	Read
5	2.6	21.0	21.5	32.0	Read and write
Total	4.5	37.2	47.0	54.9	

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Figure 10.14: Ivy system architecture