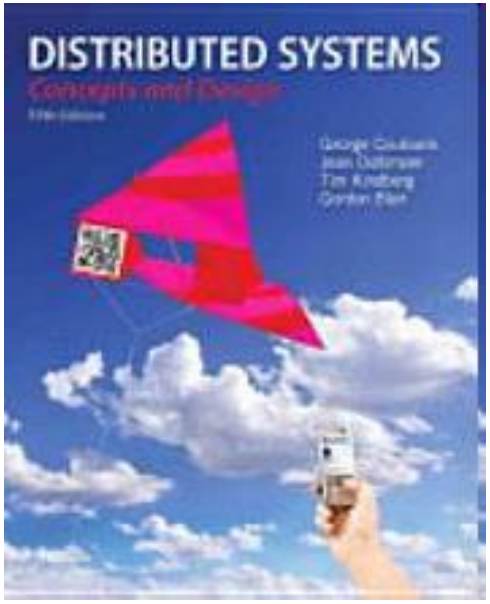
Chapter 10

Peer-to-Peer Systems



From **Coulouris, Dollimore and Kindberg**

Distributed Systems: Concepts and Design

Learning Objectives

- Understand the conceptual basis for the design of general-purpose peer-to-peer services

- Understand the algorithms that enable them to function effectively

A Good Resource For P2P System Learning

- Very good papers in P2P area:

http://www.eecs.harvard.edu/~mema/courses/cs264/cs264.html

Purpose of P2P Systems

- To support useful distributed services and applications using data and computing resources available in the personal computers that are present on the Internet

- You will study some general techniques that simplify the construction of P2P applications and enhance their scalability, reliability and security

Classification Of Peer-to-peer Networks

Pure peer-to-peer: (Gnutella and Freenet)

- Peers act as equals, merging the roles of clients and server
- There is no central server managing the network
- There is no central router

Hybrid peer-to-peer: (JXTA)

- Has a central server that keeps information on peers and responds to requests for that information.
- Peers are responsible for hosting available resources (as the central server does not have them), for letting the central server know what resources they want to share, and for making its shareable resources available to peers that request it.

Key Aspect Of A P2P System Design

- Algorithms for the placement and subsequent retrieval of information objects

- Deliver a service that is fully decentralized and self-organizing, dynamically balancing the storage and processing loads between all the participating computers as computers join and leave the service.

Characteristics of P2P Systems

- Each user contributes resources to the system
- All the nodes have the same functional capabilities and responsibilities
- Correct operation does not depend on the existence of any centrally-administered systems
- Offer a limited degree of anonymity to the providers and users of resources
- An algorithm for the placement of data across many hosts so that the workload balancing and availability can be provided

Three Generations Of P2P System

- Napster music exchange service
- File sharing with greater scalability and fault tolerance including Freenet and Gnutella
- Middleware layers for the application-independent management of distributed resources on a global scale (Pastry, Tapestry, CAN, Chord, etc.)

Overlay Network

- An overlay network is a computer network which is built on top of another network.
- Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network.
- For example, many peer-to-peer networks are overlay networks because they run on top of the Internet.
- Dial-up Internet is an overlay upon the telephone network.

Uses Of Overlay Networks

- Overlay networks can be constructed in order to permit routing messages to destinations not specified by an IP address.

- For example, Freenet and distributed hash tables can be used to route messages to a node storing a specified file, whose IP address is not known in advance.

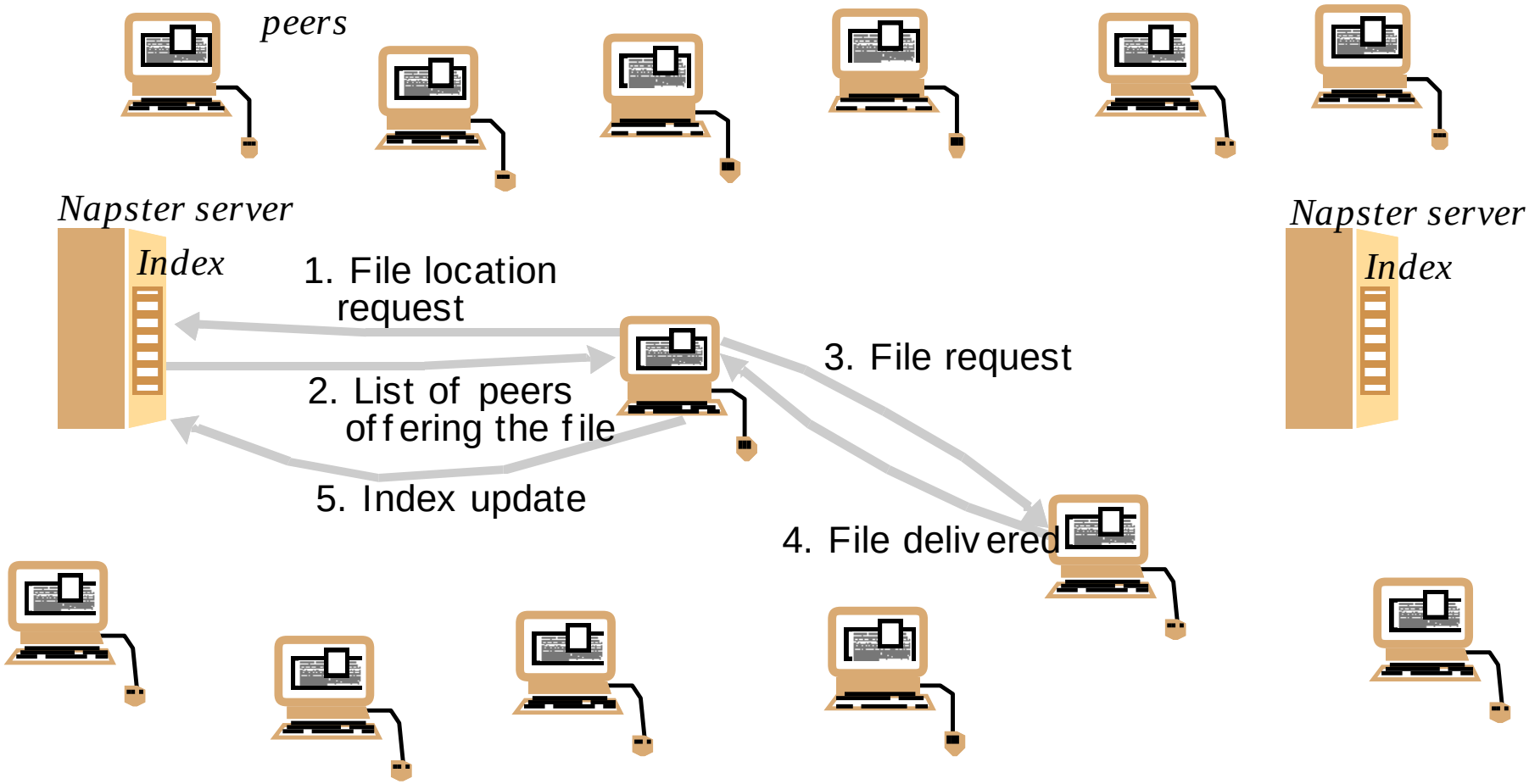
Figure 10.1: Distinctions between IP and overlay routing for peer-to-peer applications

	IP	*Application-level routing overlay*
Scale	IPv4 is limited to 232 addressable nodes. The IPv6 name space is much more generous (2128), 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 (>2128), 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 And Its Legacy

- The first application in which a demand for a globally-scalable information storage and retrieval service emerged in 1999.
- Several million users were registered and thousands were swapping music files simultaneously.
- Centralized indexes but users supplied the files, which were stored and accessed on their personal computers.

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



Lessons Learned from Napster

- Demonstrating the feasibility of building a useful large-scale service which depends wholly on data and computers owned by ordinary Internet users.
- Napster takes account of network locality when allocating a server to a client requesting a song.

Limitations:

- Music files are never updated
- No guarantees are required concerning the availability of individual files

Flooding-Style Networks (How it works)

- Old Internet P2P applications typically provide locator functions using time to-live (TTL) controlled-flooding mechanisms
- The querying node wraps the query in a single message and sends it to all known neighbors
- The neighbors then check to see whether they can reply to the query by matching it to keys in their internal database
- If they find a match, they reply; otherwise, they forward the query to their own neighbors and increase the message's hop count
- If the hop count passes the TTL limit, forwarding stops
- The TTL value thus defines a boundary or "horizon" for the query that controls its propagation

Flooding-Style Networks (An Example)

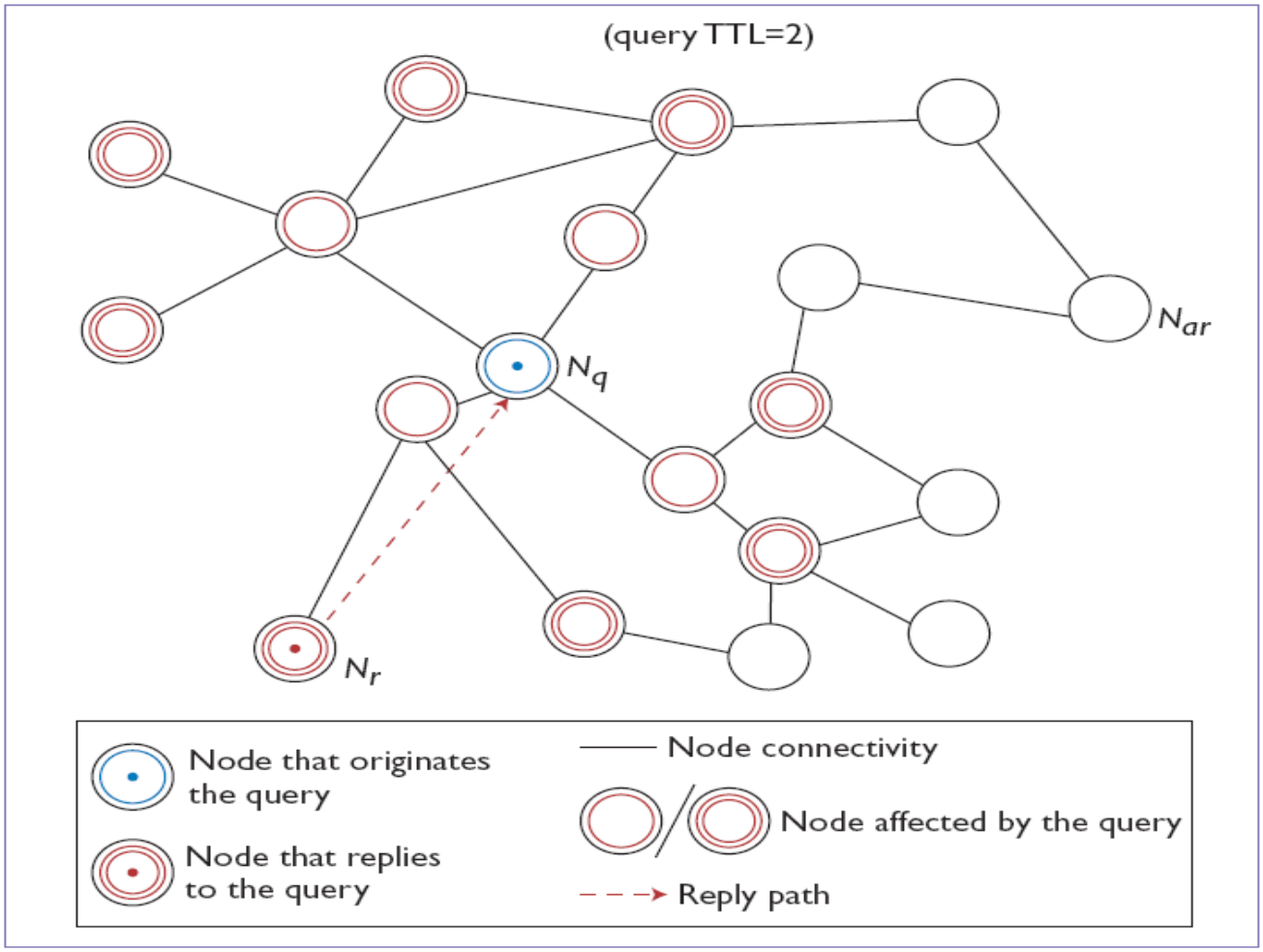


Figure 1. Sample TTL-based P2P network and query. The N_q node transmits a query requesting the value of a key located in N_r. Concentric circles indicate the number of message hops.

Flooding-Style Networks (Limitations)

- However, flooding-based systems don't scale well because of the bandwidth and processing requirements they place on the network
- They provide no guarantees as to lookup times or content accessibility
- Overlay networks can address these issues
- Overlay networks have a network semantics layer above the basic transport protocol

Overlay Networks (Features)

- Guaranteed data retrieval
- Provable lookup-time horizons (typically O(logN) with N being the number of network nodes)
- Automatic load balancing
- Self-organization

How They work?

- Each overlay node has two neighbors: the node whose value is the next available (higher) integer, and the node whose value is the previous available (lower) integer

- If the current node is the network's lowest or highest identifier, one of the neighbors will be the opposite value in the available node range (that is, the highest or the lowest, respectively)

- To join the network, a node must perform an out-of-band request — such as a broadcast — to find another network node.

Overlay Networks

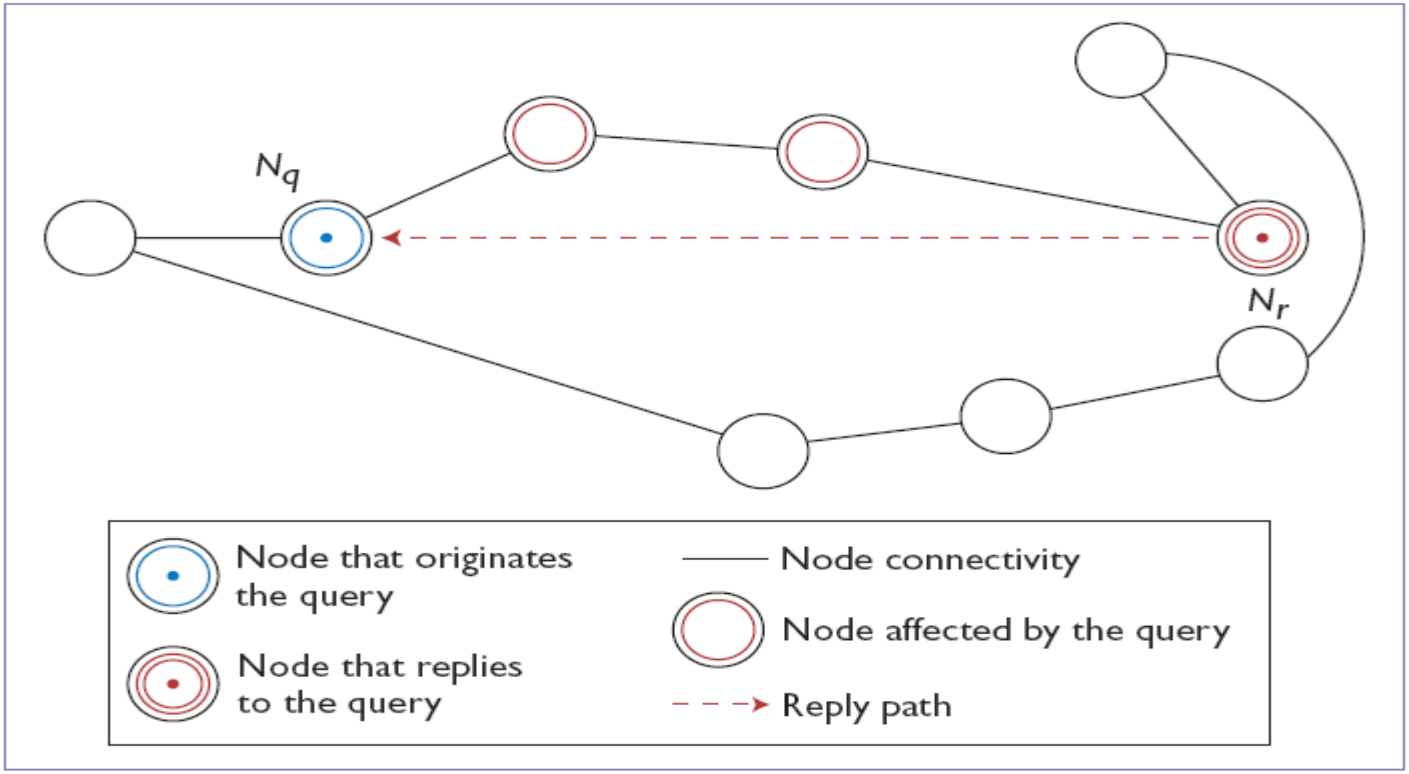


Figure 2. Sample overlay network and query. This structured topology is typical of overlay networks, though the algorithms that build the overlay (and resulting structure) vary according to network type.

Peer-to-peer Middleware

Meet the need for the automatic placement and subsequent location of the distributed objects managed by P2P systems and applications

Functional Requirements

- Simplify the construction of services that are implemented across many hosts in a widely distributed network
- Enable clients to locate and communicate with any individual resource made available to a service
- The ability to add new resources and to remove them at will
- The ability to add hosts to the service and remove them

Non-functional Requirements

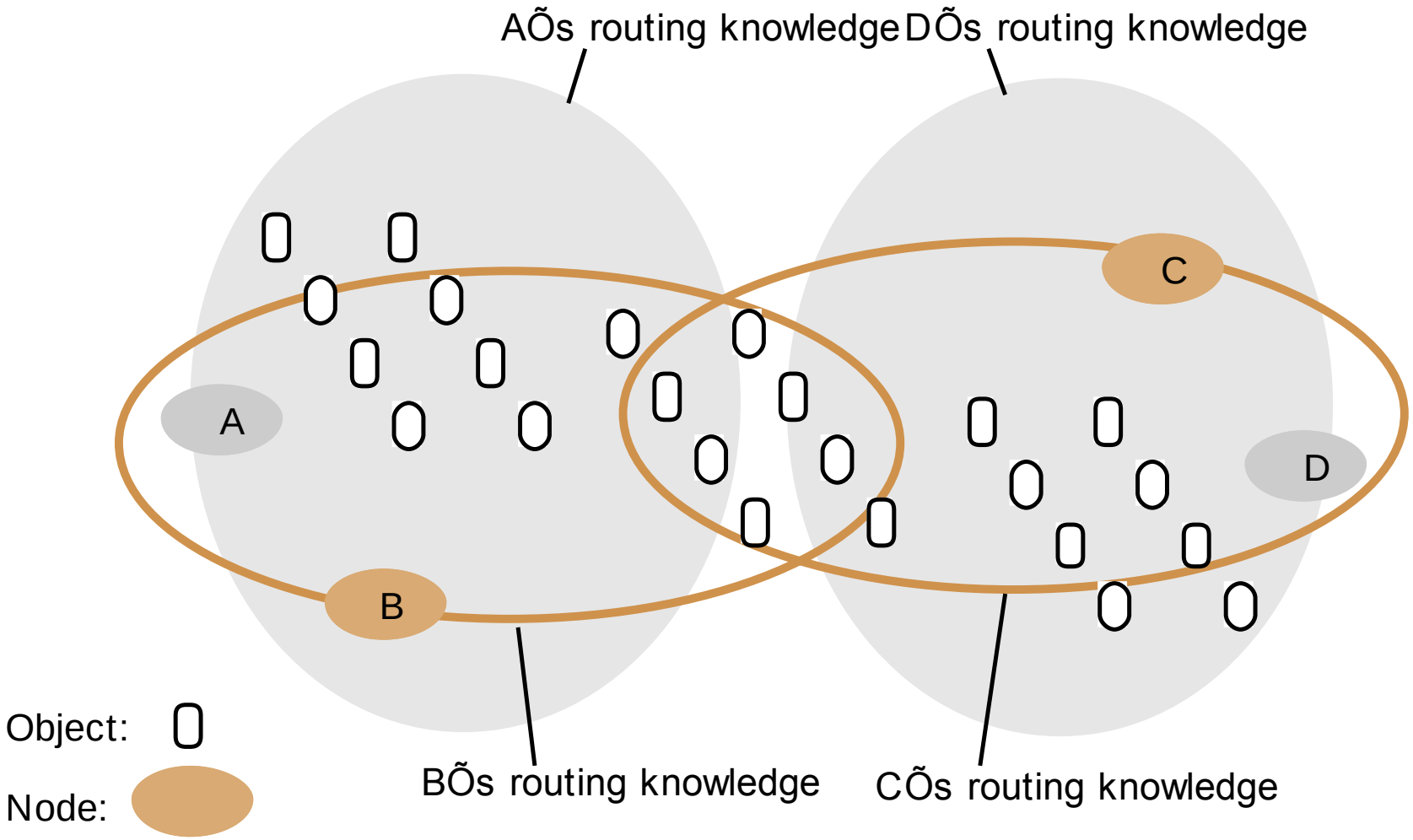
- Global scalability: access millions of objects on hundreds of thousands of hosts
- Load balancing: random placement and replicas
- Optimization for local interactions between neighboring peers: place resources close to the nodes that access them the most
- Accommodating to highly dynamic host availability: provide a dependable service on many dynamic factors
- Security of data: trust must be built up by authentication and encryption mechanisms
- Anonymity, deniability and resistance to censorship: deny responsibility for holding or supplying data

Partitioned and Distributed Location Knowledge

- Knowledge of the locations of objects must be partitioned and distributed throughout the network

- Each node is made responsible for maintaining detailed knowledge of the locations of nodes and objects in a portion of the same space as well as a general knowledge of the topology of the entire name space

Figure 10.3: Distribution of information in a routing overlay



Routing Overlays

- A distributed algorithm takes responsibility for locating nodes and object
- The middleware takes the form of a layer that is responsible for routing requests from any client to a host that holds the object to which the request is addressed
- A routing mechanism in the application layer that is different from IP routing

Main Tasks of A Routing Overlay

- 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 Of Routing Overlay

- 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

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. Q: After *put(GUID, data)*, what will happen?

The DHT layer takes responsibility for choosing a location for it, storing it (with replicas to ensure availability) and providing access to it via *get()* operation.

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.

Functions of DOLR

- Objects can be stored anywhere and the DOLR layer is responsible for maintaining a mapping between GUIDs and the addresses of the nodes at which replicas of the objects are located.

- Objects may be replicated and stored with the same GUID at different hosts and the routing overlay takes responsibility for routing requests to the nearest available replica.

DHT and DOLR

- DHT: a data item with GUID X is stored at the node whose GUID is numerically closest to X

- DOLR: Locations for the replicas of data objects are decided outside the routing layer and the host address of each replica is notified to the DOLR using the *publish()* operation

Overlay Case Study

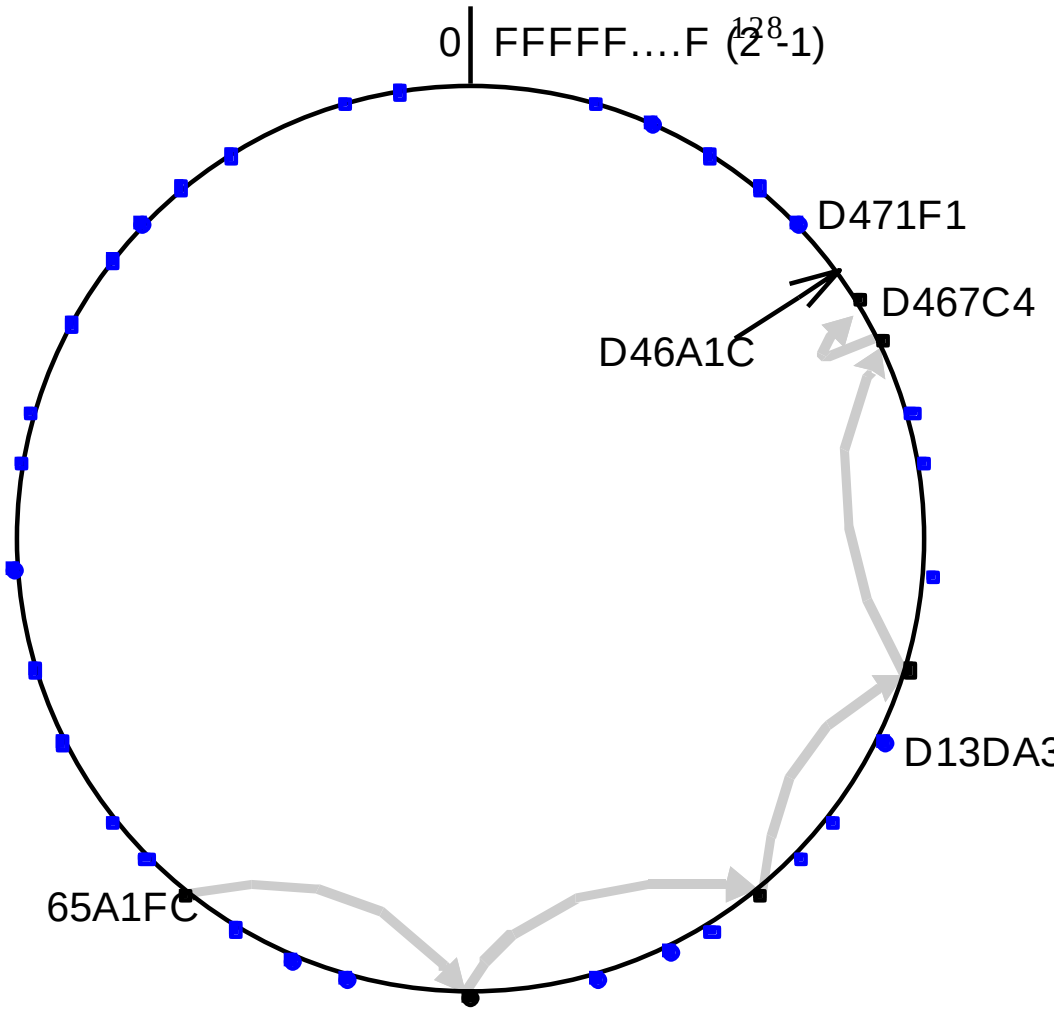
- Pastry

128-bit GUID: For nodes using public key; For objects using hash function;

Time complexity: *O(logN)*

- Tapestry

Circular routing alone is correct but inefficient (no routing table)



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. Complexity: ~$N/2l$

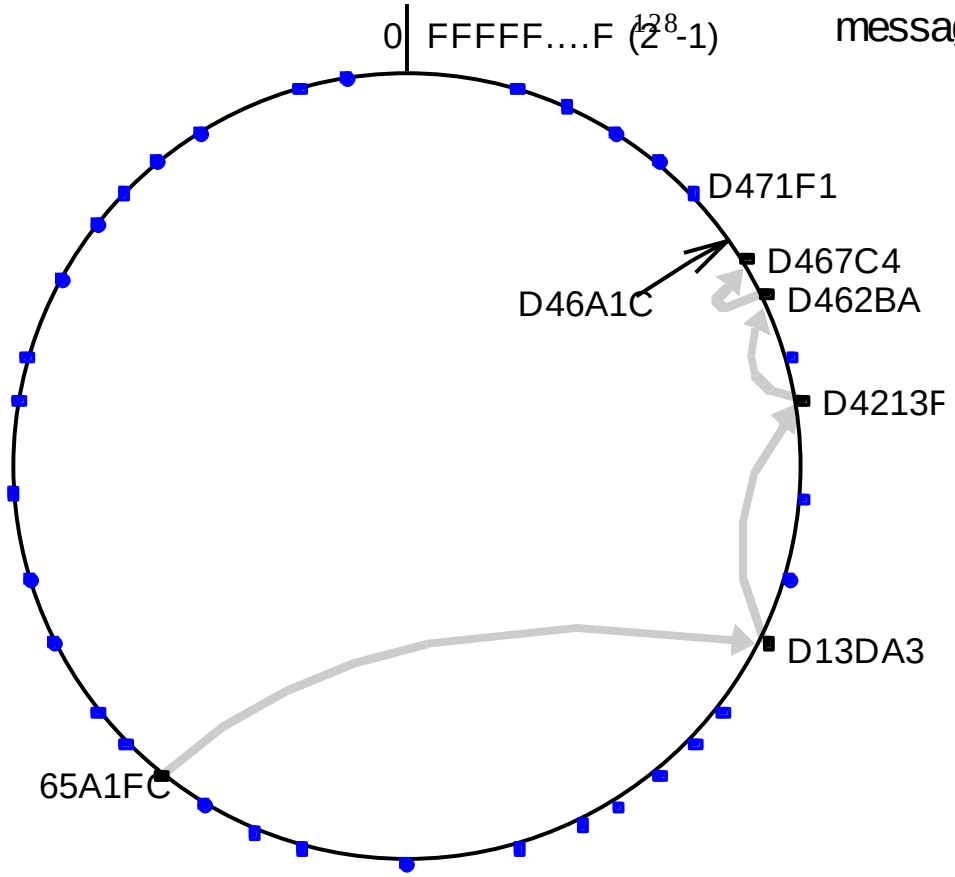
Without a routing table

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. Then 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.

}

}

Assignment #2 (Chapter 10)

- 10.4
- 10.5