Peer-to-Peer Systems

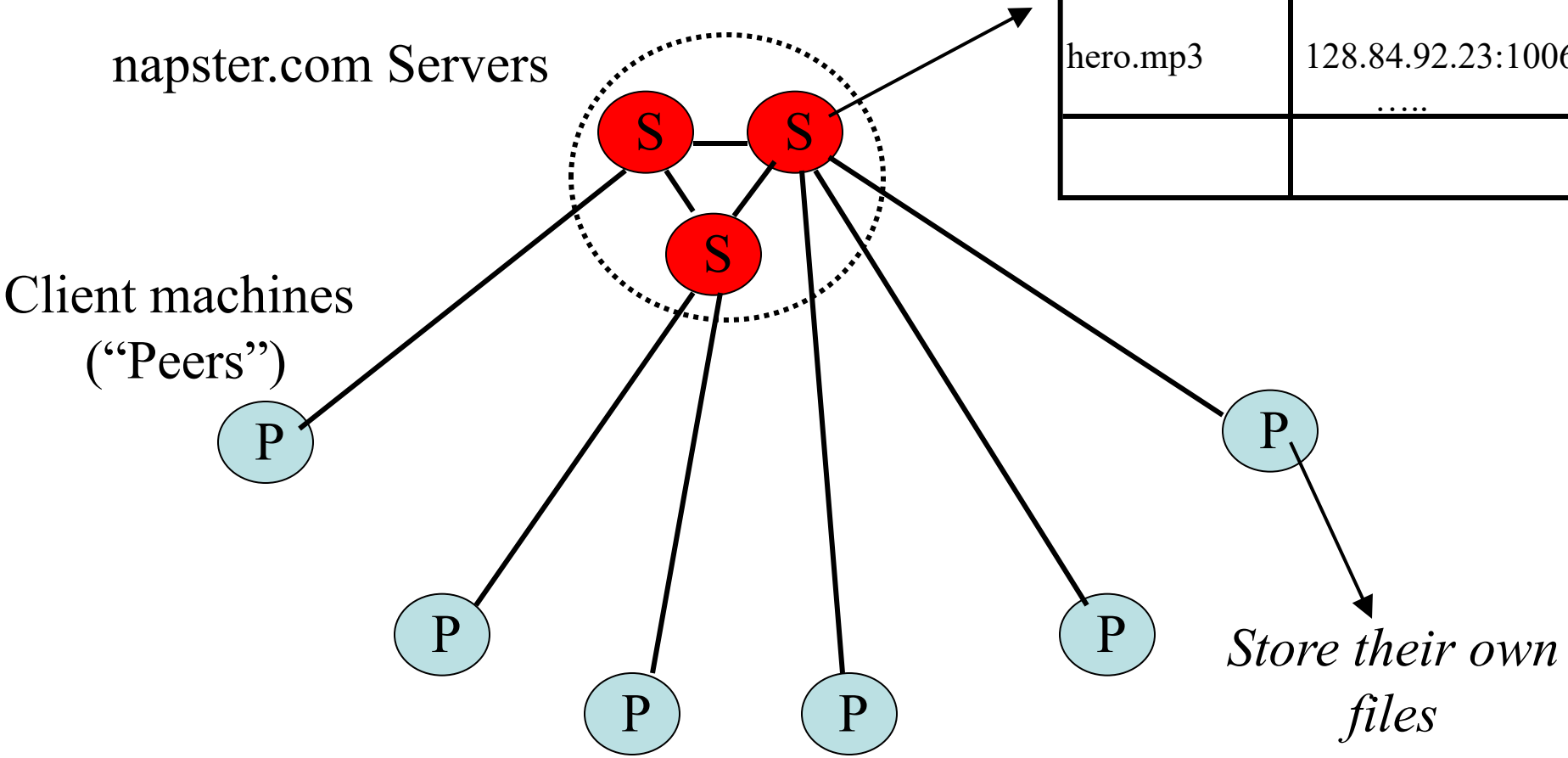
Provenance of P2P

- MP3 file sharing
  - Get your favorite songs via Internet
  - Napster, Gnutella, BitTorrent,…
  - Specialized, light-weight, easy-to-use software to enable Internet-scale file sharing

Napster Structure

Store a directory, i.e., filenames with peer pointers

Filename	Info about
hero.mp3	128.84.92.23:1006 …..

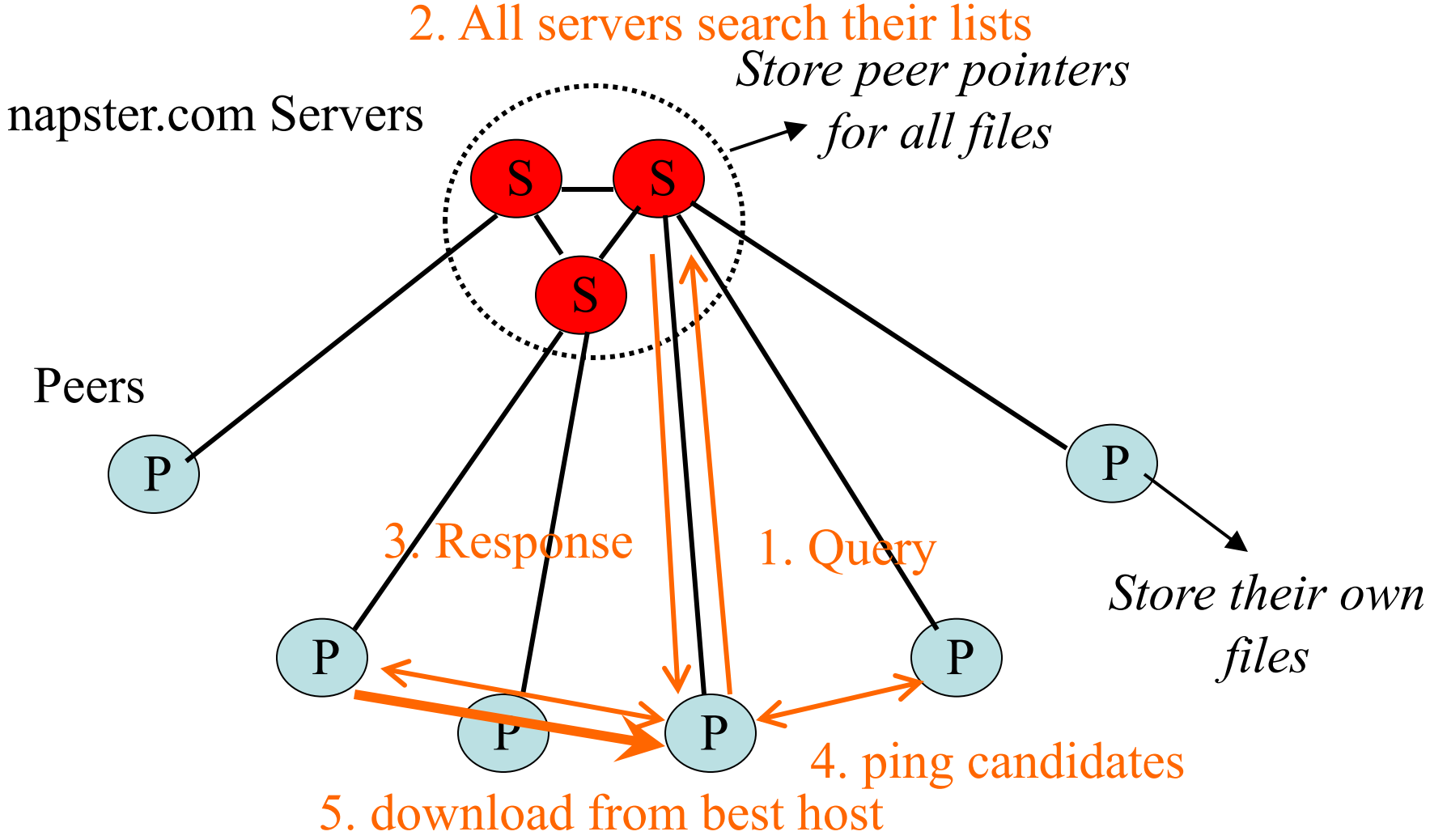


Napster Operations

Client

- Connect to a Napster server
- Upload list of music files that you want to share (**names only**, not the files themselves!)
  - Server maintains list of
    <filename, ip_address, portnum> tuples
- Search from a client:
  - Send server keywords to search with
  - (Server searches its directory with the keywords)
  - Server returns a list of matching hosts - <ip_address, portnum> tuples to client
  - Client pings each host in the list to find transfer rates
  - Client fetches file from best host
- All communication uses TCP

Napster Search



Problems

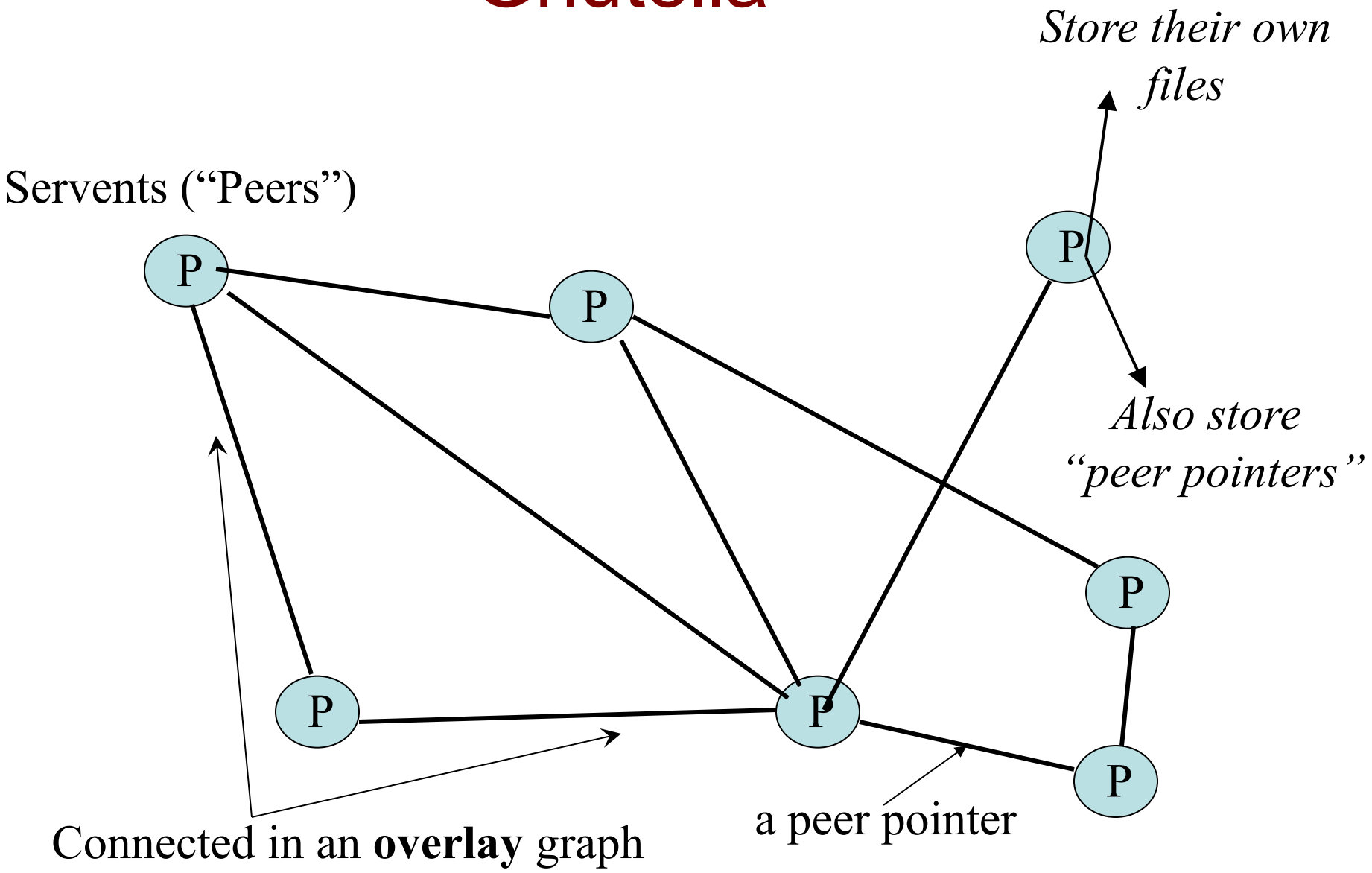
- Centralized server: a source of congestion
- Centralized server: single point of failure
- No security: plaintext messages and passwds
- napster.com responsible for allowing users' copyright violation
  - "Indirect infringement"

Gnutella

- Eliminate the servers
- The first decentralized P2P search system
- Clients act as servers too, called **servents** (SERVer + cliENT)
- [3/00] release by AOL, immediately withdrawn, but 88K users by 3/03
- Original design underwent several modifications; we'll look at the initial version

http://www.limewire.com

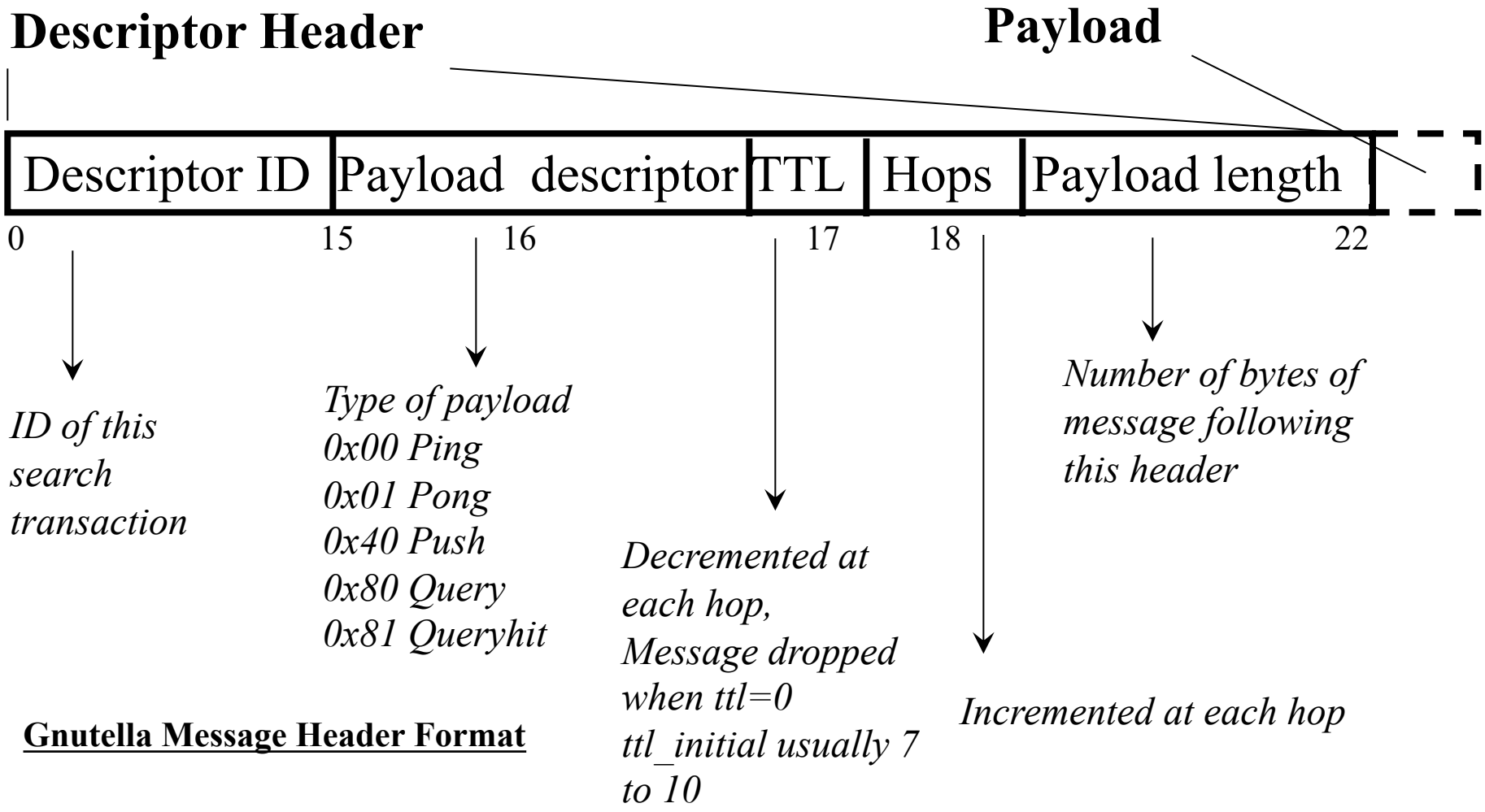
Gnutella


Store their own files
Servents (“Peers”)
P
P
P
Also store “peer pointers”
P
P
P
P
Connected in an **overlay** graph
a peer pointer

Search in Gnutella

- Gnutella *routes* different messages within the overlay graph
- Gnutella protocol has 5 main message types
  - Query
  - QueryHit (response to query)
  - Ping (to probe network for other peers)
  - Pong (reply to ping, contains address of another peer)
  - Push (used to initiate file transfer)
- We'll go into the message structure and protocol now

How do I Search the File?



Gnutella Message Header Format

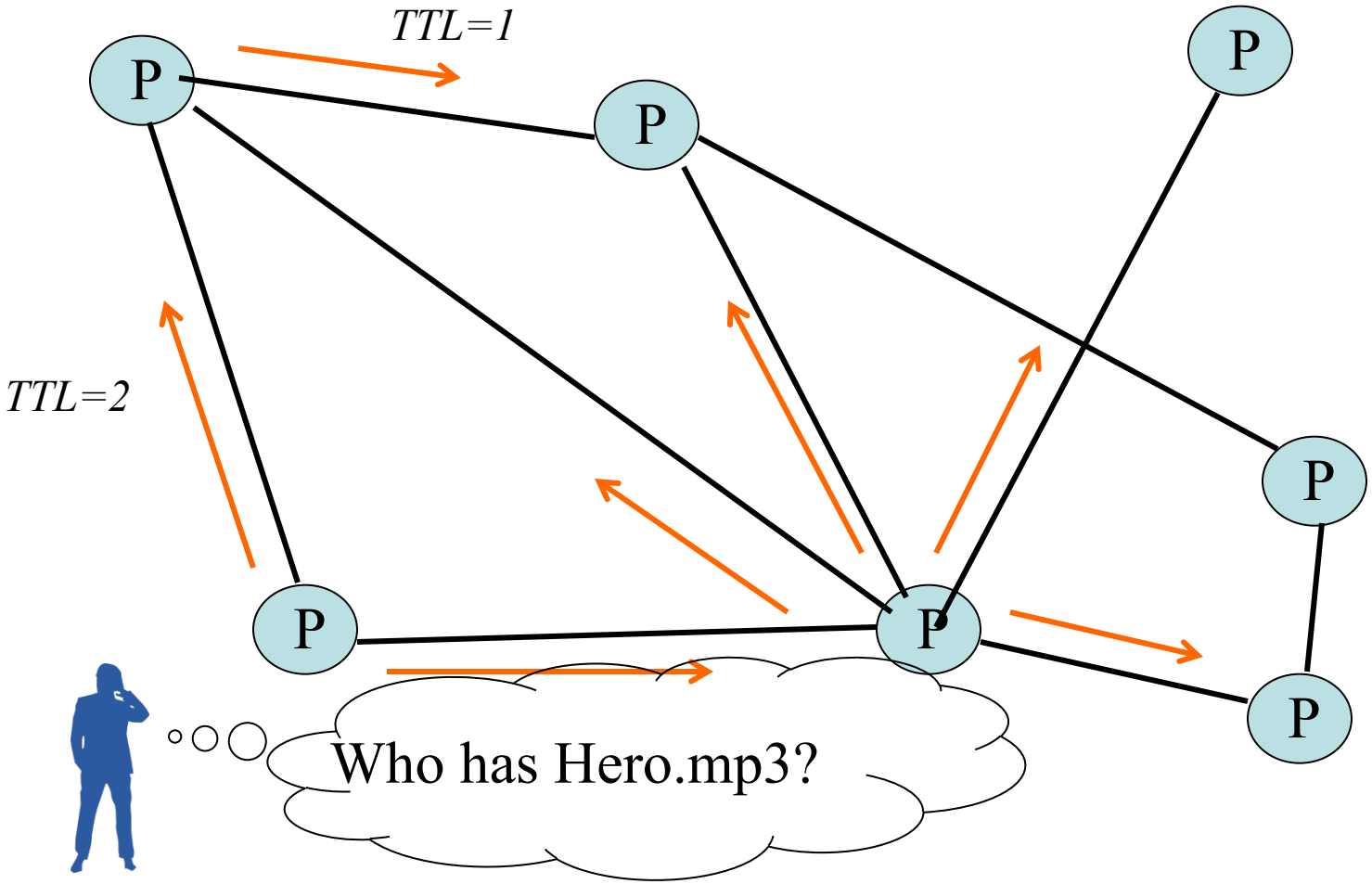
Query

Query (0x80)

Payload Format in Gnutella Query Message

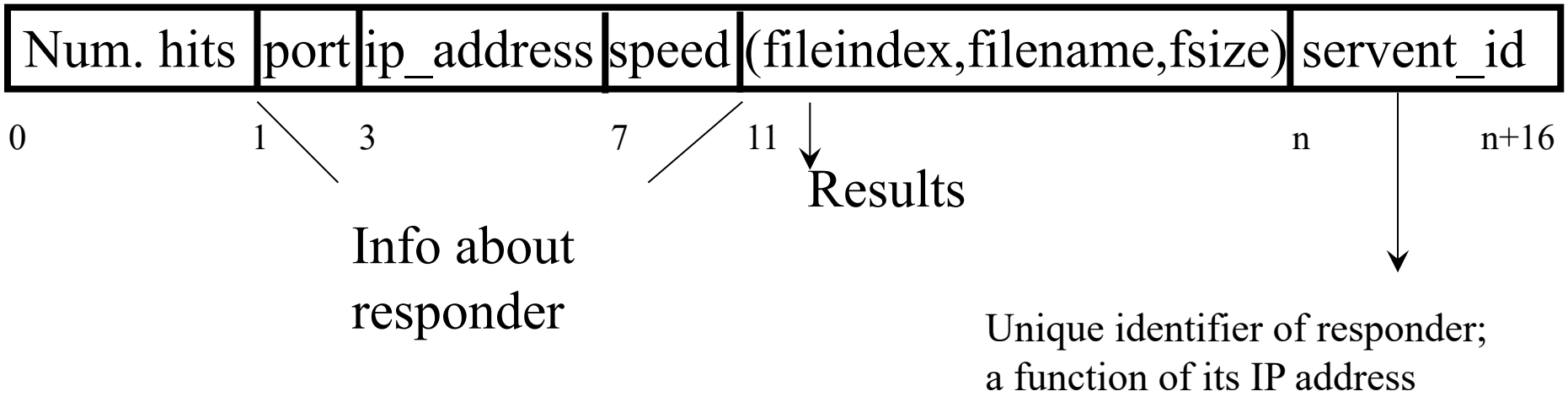
Gnutella Search

Query's flooded out, TTL-restricted



QueryHit

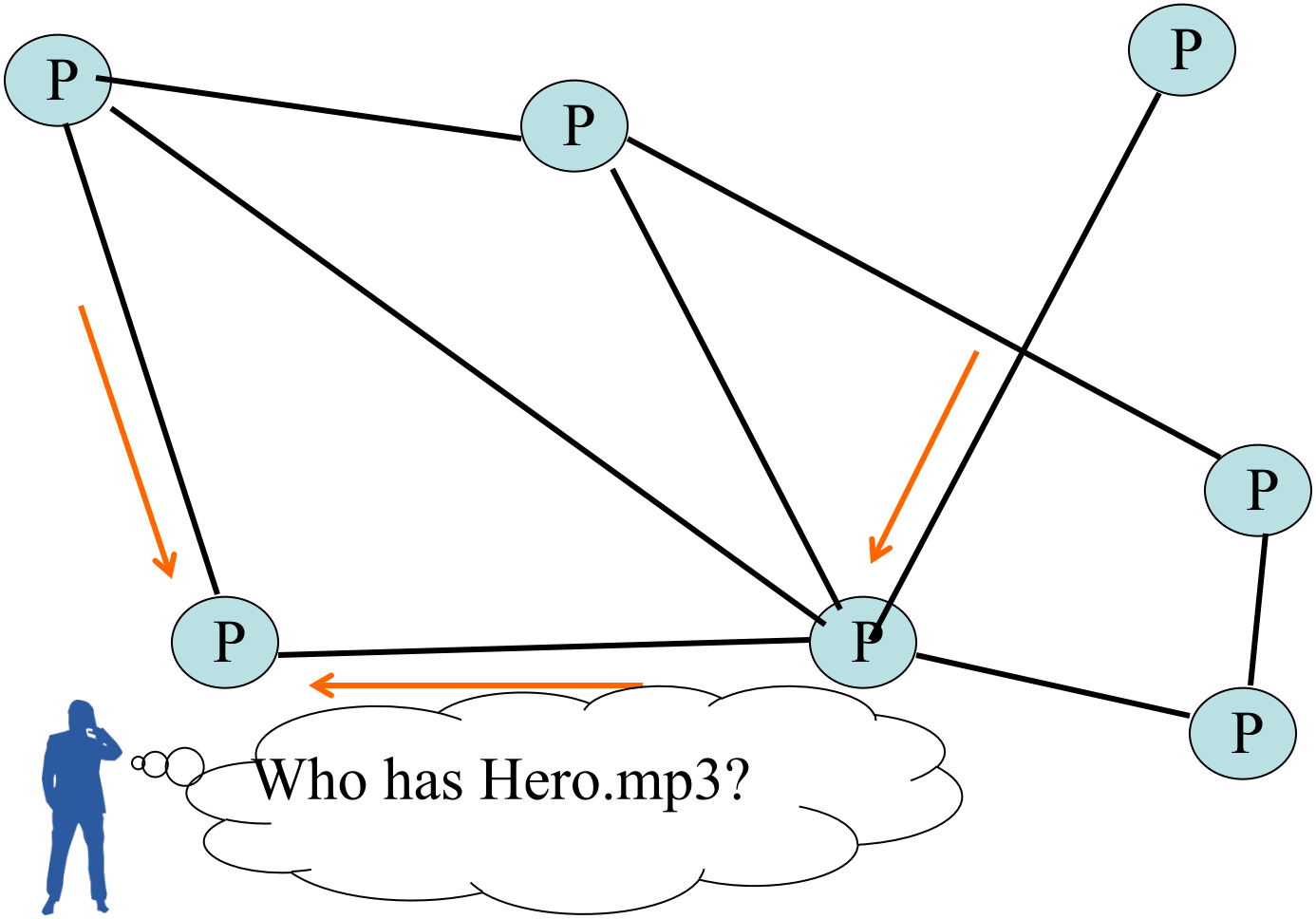
QueryHit (0x81) : successful result to a query



Payload Format in Gnutella QueryHit Message

Gnutella Search

Successful results QueryHit's routed on *reverse path*



Avoiding Excessive Traffic

- To avoid duplicate transmissions, each peer maintains a list of recently received messages
- Query forwarded to all neighbors except peer from which received
- Each Query (identified by DescriptorID) forwarded only once
- QueryHit routed back only to peer from which Query received with same DescriptorID
- Duplicates with same DescriptorID and Payload descriptor (msg type, e.g., Query) are dropped
- QueryHit with DescriptorID for which Query not seen is dropped

Ping-Pong

- Peers initiate Ping's periodically
- Pings **flooded out** like Querys, Pongs routed along **reverse path** like QueryHits
- Pong replies used to update set of neighboring peers
  - to keep neighbor lists fresh in spite of peers joining, leaving and failing

Gnutella Summary

- No servers
- Peers/servents maintain “neighbors”, this forms an overlay graph
- Peers store their own files
- Queries flooded out, ttl restricted
- Query Replies reverse path routed
- Periodic Ping-pong to keep neighbor lists fresh in spite of peers joining, leaving and failing
  - List size specified by human user at peer : heterogeneity means some peers may have more neighbors
  - Gnutella found to follow power law distribution:

P(#neighboring links for a node = L) ~ L^{-k} (k constant)

Problems

- Ping/Pong constitutes 50% traffic
  - Solution: Multiplex, *cache* and reduce frequency of pings/pongs
- Repeated searches with same keywords
  - Solution: *Cache* Query and QueryHit messages
- Large number of *freeloaders*
  - 70% of users in 2000 were freeloaders
- Flooding causes excessive traffic
  - Is there some way of maintaining meta-information about peers that leads to more intelligent routing?

➔ Structured Peer-to-peer systems

Distributed Hash Table (DHT)

- A hash table allows you to insert, lookup and delete objects with keys
- A *distributed* hash table allows you to do the same in a fully-distributed, self-organizing setting (objects=files)
- Performance Concerns:
  - Load balancing
  - Fault-tolerance
  - Efficiency of lookups and inserts
  - Locality
- Unstructured P2P systems
  - Gnutella, Napster
- Structured P2P systems
  - Pastry, Chord, CAN, etc.

PASTRY

- Pastry is a self-organizing P2P system based on this concept of distributed hashing
- Basic structure
  - Each node has a 128 bit unique identifier
  - Thus nodes are organized in a ring structure
  - When given a message and a key, a node routes the message to the node who's nodeId is numerically closest to the key

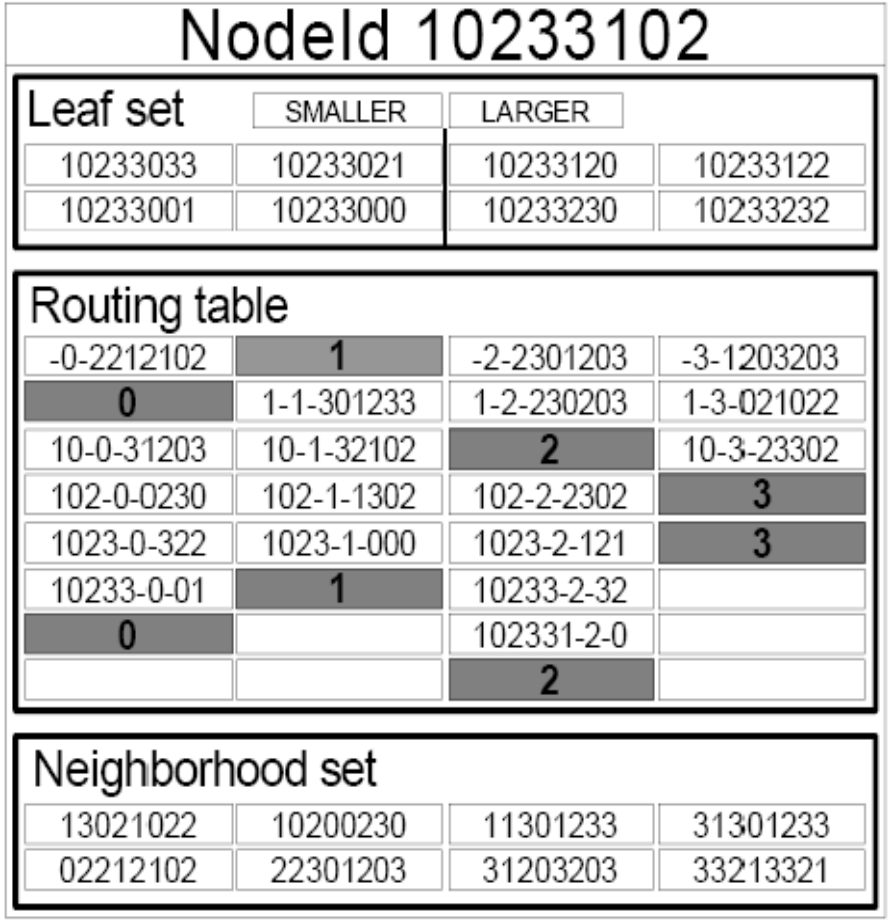
PASTRY Routing Algorithm

- Pastry's routing algorithm is designed to not only route a message to a node with the closest ID, but also to minimize the physical distance traveled (according to some definable metric e.g. IP hops)
- Each node maintains a routing table, and the criteria for choosing which node to forward a message to is dependent both on nodeId's and the proximity metric described

PASTRY Routing Algorithm

- The routing mechanism in Pastry is dependent on two configuration parameters
  - *L*: Each node maintains a list of the *L* nodes in the network that have the numerically closest Id's to the particular node's Id. (Typical value 16 or 32)
  - *b*: When routing messages, nodeId's and keys are converted to base 2^b. Each hop during the routing phase brings the message to a node whose nodeId shares one more digit in common with the key than in the previous hop. (Typical value 4)
  - This allows the algorithm to route messages to the node with the numerically closest key in $O(\log_2{}^b N)$ where N is the number of nodes in the Pastry network

PASTRY Routing Algorithm



NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

State of pastry node where *b=2* and *L=8.*
Thus numbers are in base 4

L nodes which are numerically closest to node (*L/2* smaller and *L/2* larger)

Routing table contains one row for each digit in the nodeID ($\log_2{}^b N$ rows).

Each row contains an entry for each possible value for that digit. All of the preceding digits for the entry must match the current nodeId's while the following digits may be any combination.

PASTRY Routing Algorithm

- A message with key *D* arrives at a node with nodeId *A*
- The node first checks the leaf set
  - If *D* is within the range of the nodeId's in the leaf set, the message is forwarded to the node in the set whose nodeId is closest to *D*
  - If not, the routing table is used.
    - Let *k* be the number of digits that *D* and *A* have in common
    - Forward the message to the node in the k^{th} row that has the $k+1^{th}$ digit in common with *D*
    - If this entry does not exist, forward the message to the node that has the next closest numerical value out of all nodes in the leaf set and routing table

PASTRY Routing Algorithm

- Example route:

```
Message key: d46a1c

Source node: 65a1fc

0 matching digits
```

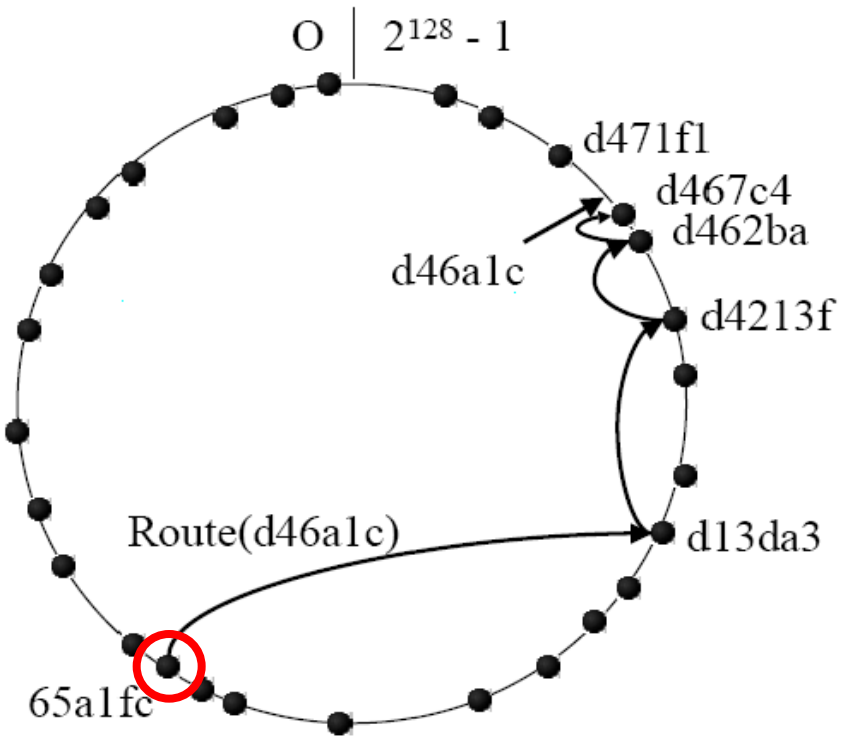


Figure 2. Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

PASTRY Routing Algorithm

- Example route:

```
Message key: d46a1c

First hop  : d13da3

1 matching digit
```

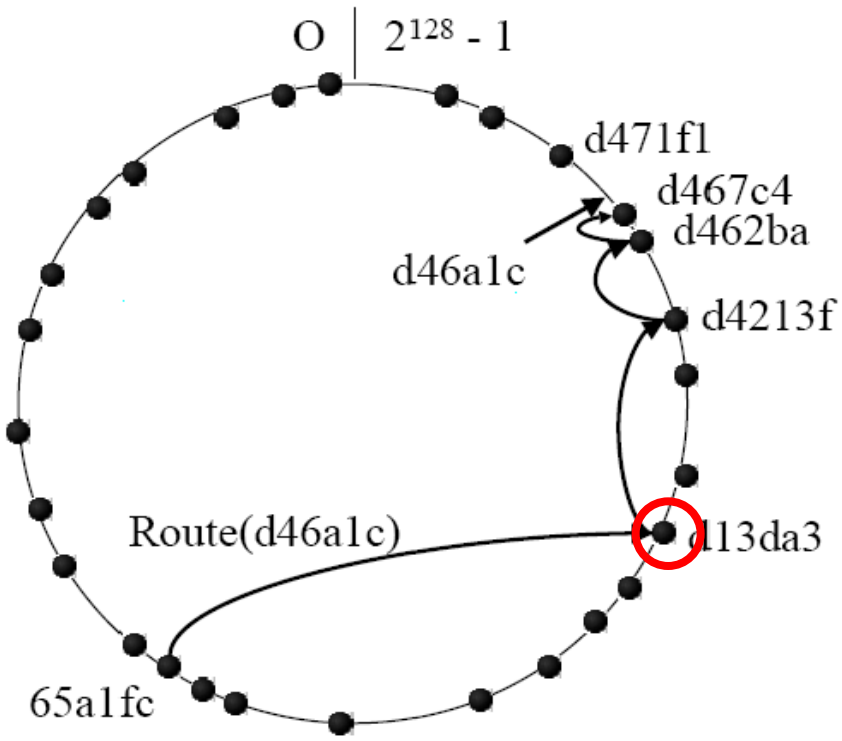


Figure 2. Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

PASTRY Routing Algorithm

- Example route:

```
Message key: d46a1c

Second hop : d4213f

2 matching digits
```

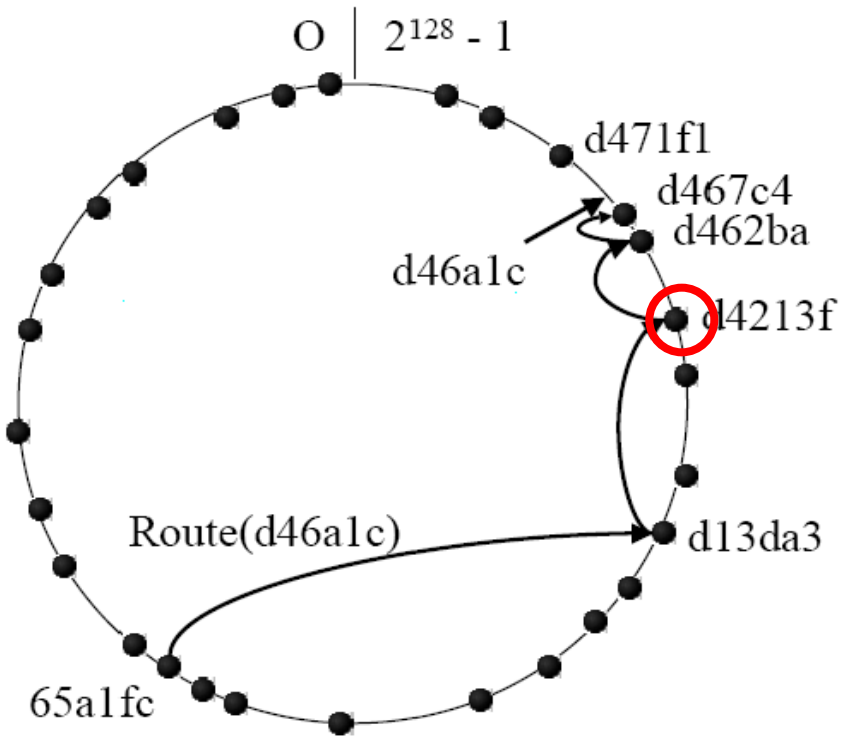


Figure 2. Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

PASTRY Routing Algorithm

- Example route:

```
Message key: d46a1c

Third hop  : d462ba

3 matching digits
```

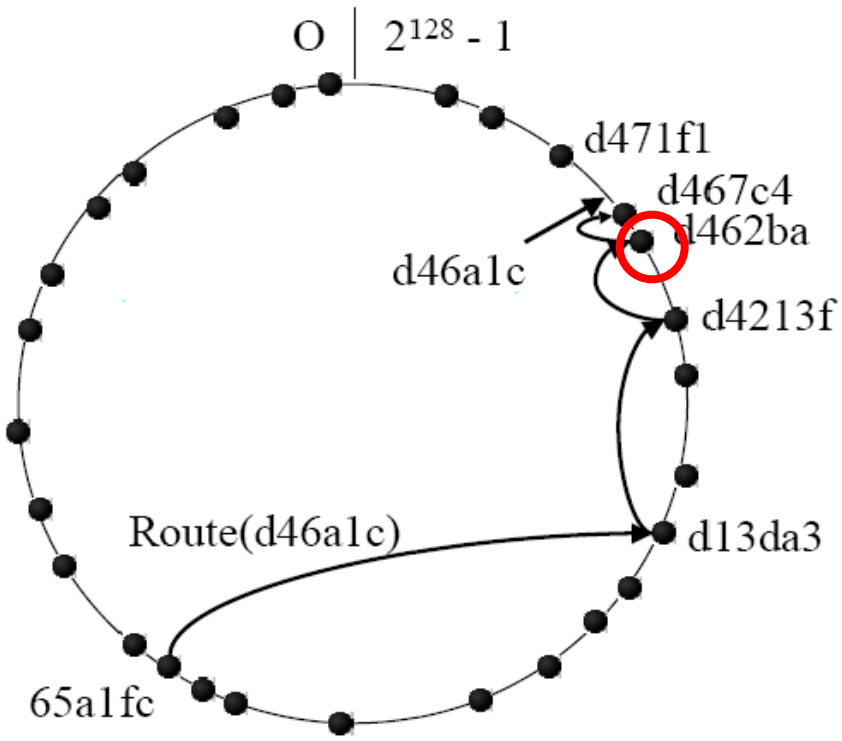


Figure 2. Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

PASTRY Routing Algorithm

- Example route:

```
Message key: d46a1c

Fourth hop : d467c4

3 matching digits
```

No live nodes with 4 matching digits exists. Forwarded to node with next closest numerical id with 3 matching digits.

Destination reached.

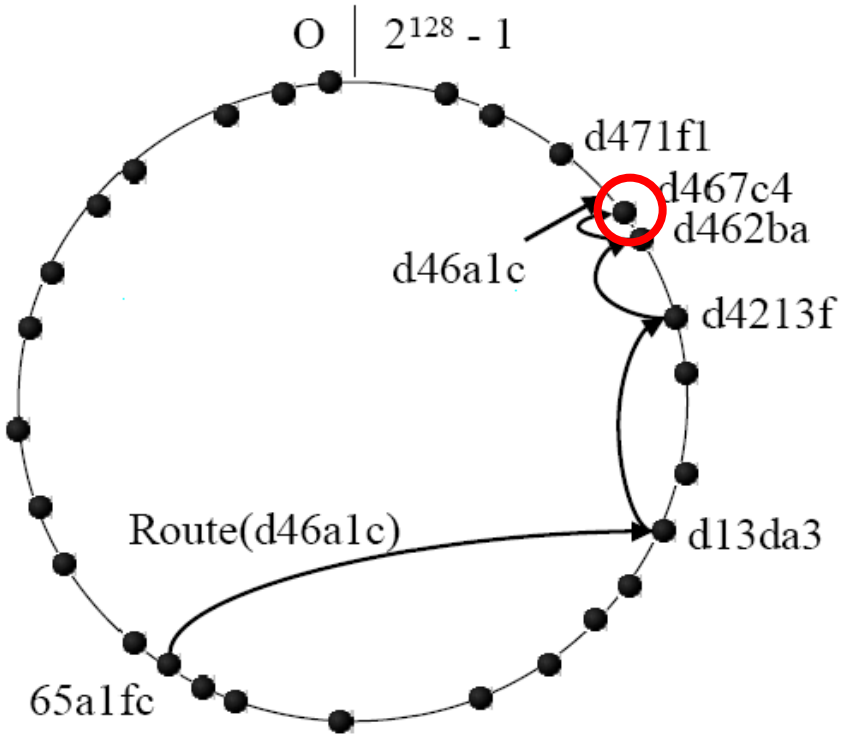
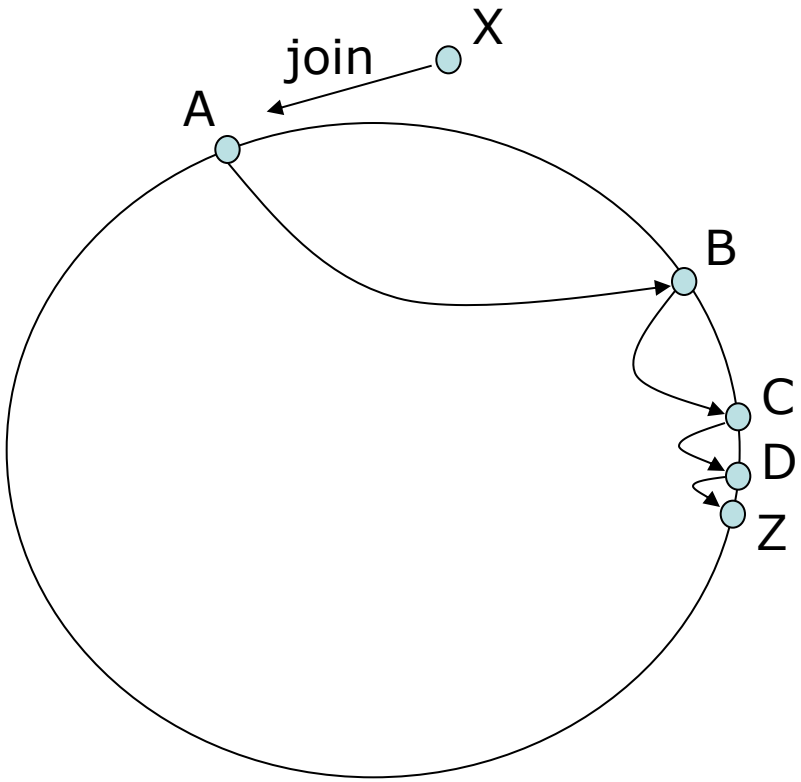


Figure 2. Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

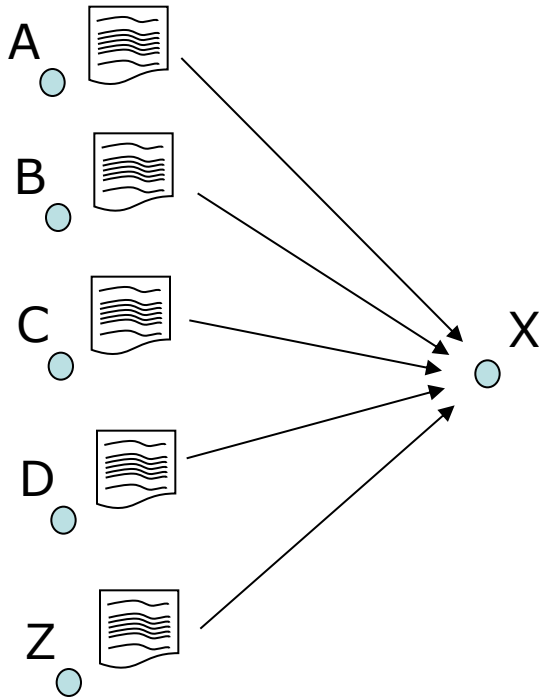
Node Additions/File Insertions

- A new node with nodeId *X* joins the Pastry ring by contacting a known node with nodeId *A*
- Node *A* sends a special join message with key *X* out to the Pastry network
- The message is routed like any other message, eventually reaching the Pastry node *Z* with the closest nodeId to *X*
- Like any other message, a number of intermediate hops are passed through



Node Additions

- Each node along the path of the join message sends its routing state information to node *X*
- Node *A* is used for the basis of the neighborhood set since it is assumed to be in close proximity to *X*
- Node *Z* is used for the basis of the leaf set since it has the closest numerical nodeId in the network
- The routing table for *X* is built using the state information from each of the intermediate nodes



Node Additions

- Routing table construction for node *X*
  - Consider the properties of the Pastry routing algorithm. The nodes along the path to the join method obey the following general properties:
    - Node *A* shares 0 digits in common with *X*
    - Node *B* shares 1 digit in common with *X*
    - Node *C* shares 2 digits in common with *X*
    - …
  - Each row in the routing table can be built using the corresponding row of the corresponding hop:
    - Row 0 can be built using Row 0 of Node *A*
    - Row 1 can be built using Row 1 of Node *B*
    - …

Node Additions

- Once node *X* has finished building its state information, it sends a notification message to each of the nodes in its routing table, leaf set, and neighborhood set
- Each of these nodes update their state information accordingly

Node Additions

- Maintaining locality
  - One last phase before node addition is complete
  - Node X requests state information from all nodes in its new routing table
  - When it receives these states, it updates its routing table according to the proximity metric, replacing nodes in its table with ones that are “closer” to it
- Total messages required for node addition: $O(log_{2^b}N)$

Node Departures/Failures

- Lazy repair
  - State information of a Pastry node is not updated for a failed or departed node until a failed attempt to contact that node
  - If the node is in the leaf set, contact the highest id node in the leaf set and repair the leaf set using that nodes leaf set
  - If the node is in the routing table
    - Contact a different node in the same row and ask for the appropriate entry
    - If no node is found from the entries in that row, ask the nodes in the next row, etc.

What Makes P2P Special?

- Equal role
  - Peers act as both clients and servers
- Self-organization
  - Does not reply on centralized infrastructure
- Decentralization
  - Fully distributed algorithms
- Churn handling
  - Dynamic node arrival/departure
- Redundancy
  - A good thing to leverage
- All these leads to inherent scalability and robustness of P2P Systems