- Consider a file storage application provided by a client/ server system
  - Architecture
    - Single server, cluster, remote replication
  - File location
    - Well known location(s)
  - Scalability
    - Storage, bandwidth
  - Availability
    - What proportion of requests are satisfied
  - Manageability / control
    - How well can we control the service
  - (Total) Cost
    - Purchase, operation, administration, maintenance

- Peer-to-Peer systems offer an alternative to the traditional client/server approach
  - "exploit resources available at the edges of the Internet"
  - commodity personal computers or workstations
  - becoming increasingly possible and desirable
- What resources?
  - storage capacity
  - stored content
  - CPUs
  - users?
- Examine:
  - characteristics and challenges
  - middleware
  - applications

- Each peer contributes resources to the system
  - the aggregated set of resources is used to provide the service
- All peers in the system have the same capabilities
  - although peers may differ over time in the roles they perform or the resources they provide
- No dependence on the availability of a centralized component or service
  - What well known P2P application violated this characteristic?
- A key issue in the design of a peer-to-peer system is the placement of migratable resources in the system (and the subsequent location of these resources)
  - balancing workload, availability, overheads (e.g. latency)

# Volatility

- owners/managers/users of peers (hosts/nodes) in a peer-to-peer system will usually not (or cannot) provide guarantees for the availability of individual peers, leading to *churn*
- Therefore, a peer-to-peer service cannot guarantee the availability of a resource
  - although a system can be designed to make the probability of failure (I.e. unavailability) small
- This is an issue called 'harvest'
  - what proportion of the normally available resources can currently be accessed?

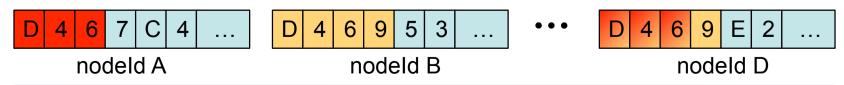
- 3<sup>rd</sup> Generation Peer-to-Peer systems
  - 1st Generation: e.g. Napster
  - 2<sup>nd</sup> Generation: e.g. Freenet, Kazaa, Gnutella, BitTorrent
- examples: <u>Pastry</u>, Tapestry, CAN, Chord, Kademlia
- Peer-to-peer middleware systems
  - place resources (files, data objects, ...) on a set of computers that are widely distributed throughout the Internet
  - route messages to these resources on behalf of clients
  - relieve client applications of this responsibility
- Resources are identified by globally unique IDs (GUIDs)
  - a secure hash from some or all of the object's state
- (Overlay) Routing takes place in the GUID space, not the IP address space

- Assign resources to nodes on behalf of applications
  - Add new resources and remove existing ones
  - Load balancing: achieve an even/fair distribution of load across peers through random placement and replication of resources
- Subsequently, locate resources on behalf of applications
- Optimise for local interactions between nearby clients (in the network space, **not** the GUID space)
- Add new hosts and remove existing ones
  - Hosts in a peer-to-peer system will usually not provide guarantees about availability in the same way that centralised servers do
  - Accommodate volatile host availability (churn)
  - Redistribute resources / load
- Other issues of less interest to us: security, anonymity, freedom

**Pastry** 

7

- A. Rowstron and P. Druschel, "Pastry: Scaleable, decentralized object location and routing for large-scale peer-to-peer systems", 2001
  - Required reading
- Nodes and objects are assigned 128-bit GUIDs
  - Computed using a secure hash function
  - Nodes in close proximity w.r.t. the nodeld space are likely to be distant w.r.t. the network, geographic, admisistrative, ... spaces
- Pastry uses prefix routing to route a message with destination D to the node that is numerically closest to D
  - in each routing step, a message is (usually) forwarded to a node B that shares with D a nodeld prefix that is b bits (if b=4, then 1 hex digit) longer than that shared between A and D, for example:



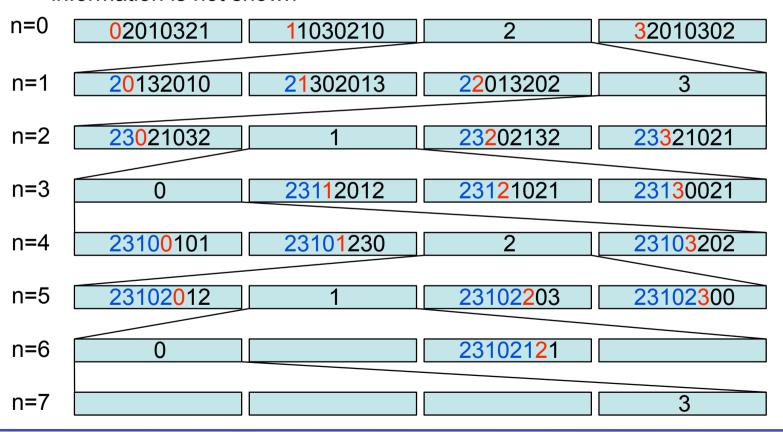
Each Pastry node maintains routing state information

# Routing Table R

- log<sub>2</sub><sup>b</sup> N rows
- 2b 1 entries in each row
- each entry contains nodeld and associated IP address
- each entry in row n refers to a node whose nodeld shares the same first n digits but whose n+1<sup>th</sup> digit has one of the 2<sup>b</sup>-1 other possible values other than the n+1<sup>th</sup> digit in this node's nodeld
- usually there will be many nodes that could be placed in a given entry of the routing table, but we only record one of them
- in practice, an effort is made to ensure that the node selected for a given entry in R is *close* to the current node (according to some definable proximity metric, e.g. network latency)
- if no suitable node for a given entry is known, then the entry is left blank
- the maximum number of rows in the routing table is 128/b but in practice only (approx) log<sub>2</sub><sup>b</sup> N rows will be populated

### Routing table example

- b=2, nodeld=23102103
- each entry also contains the IP address of the associated node but this information is not shown



#### Leaf Set L

- Contains the set of nodes numerically closest to the current nodeld (i.e. w.r.t. nodeld, not the proximity metric)
- |L| / 2 closest nodes with smaller nodelds and |L| / 2 closest nodes with larger nodelds
- Used during message routing (more later)

### Neighbourhood Set M

- Contains the nodelds and IP addresses of the |M| closest to the current node, w.r.t the chosen proximity metric (e.g. network latency)
- Not normally used for routing messages but is used to maintain/improve routing locality properties (more later)
- Typical values for both |L| and |M| are 2<sup>b</sup> or 2<sup>b+1</sup>

 Routing algorithm describes the actions taken by a node A on receipt of a message D to forward the message to the nest node along the route (to D)

### Some notation

- R<sub>I</sub> is the routing table entry at column i and row I
- L<sub>i</sub> is the i<sup>th</sup> closest nodeld in the leaf set, with –ve values indicating smaller nodelds and +ve values indicating larger nodelds
- D is the GUID of the message destination
- D<sub>I</sub> is the value of the I<sup>th</sup> digit in D
- shl(A,B) is the length of the prefix shared between A and B, measured in digits

```
 if (L<sub>-⌊|L|/2⌋</sub> ≤ D ≤ L<sub>⌊|L|/2⌋</sub>) {

(2)
     // D is within range of our leaf set
        forward to L_i, s.th. |D - L_i| is minimal;
(3)
(4) } else {
(5) // use the routing table
(6) Let l = shl(D, A);
(7) if (R_i^{D_l} \neq null) {
          forward to R_i^{D_I};
(8)
(9)
(10)
        else {
(11)
            // rare case
     forward to T \in L \cup R \cup M, s.th.
(12)
(13) shl(T, D) \ge l,
(14) |T - D| < |A - D|
(15)
(16) }
```

### **Pastry API**

- Joining an existing Pastry network (or creating a new one if this is the first node to join)
  - nodeId = pastryInit(Credentials, Application)
  - Credentials: Application specific used for authentication
  - Application: Object used for application callbacks
  - Returns nodeId (GUID) for the joining node
- Routing a message to a destination
  - route(msq, key)
  - Routes msg through the Pastry network to the node with the nodeld numerically closest to the key
- Application object must provide
  - deliver(msg, key): called when a msg is received and the current nodeld is closest to the key
  - forward(msg, key, nextld): called by Pastry before a msg with destination key is routed to the next node in the route identified by nodeId – the current node may modify the message or stop it
  - newLeafs (leafSet): informs application of a change in the leafSet

 Arriving nodes need to populate their routing table, neighbourhood set and leaf set

#### Procedure

- Initially, an arriving node with nodeld X contacts an existing node A (which it is told about or can find, e.g. through multicast/broadcast)
- Node A routes a **join** message to Z the existing node that is numerically closest to the new node X – through the Pastry network
- Each node in the route A..Z along the Pastry route to X sends its state information to X which uses the received information to initialize its state
- Initializing neighbourhood set
  - It is assumed that A was in close proximity to X so A's neighbourhood set is used by X
- Initializing leaf set
  - Z will be numerically close to X (last step in the Pastry route) so Z's leaf set is used by X

### Initializing routing table

- Entries in the first row of X's routing table (X<sub>0</sub>) must be in close proximity to X but are independent of X's nodeld so A<sub>0</sub> is a good choice of entries for X<sub>0</sub>
- For row X<sub>1</sub>, where each entry has a prefix of length 1 in common with X, we can use the entries in B<sub>1</sub> the second row of B's routing table, where B is the second step in the route from A .. Z
- Similarly, for row  $X_2$ , where each entry has a prefix of length 2 in common with X, we can use the entries in  $C_2$  the third row of C's routing table, where C is the third step in the route from A .. Z
- We continue like this until we have populated the routing table

# Finally

 X sends its new state to every node in R, L and N to allow them to update their own state using X's information

## Pastry node departure

- A pastry node is considered failed when its immediate neighbours can no longer communicate with it
  - Requires repair of leaf sets in which the failed node appeared
  - Node that discovers failure will obtain leaf set from a node close to the failed node and replace the failed node with one from this other leaf set
  - Works unless |L| / 2 nodes fail simultaneously
  - To repair a routing table entry R<sub>I</sub><sup>d</sup>, R<sub>I</sub><sup>i</sup> is contacted (i ≠ d) and sends its own R<sub>I</sub><sup>d</sup>
  - If this fails to produce a valid replacement, R<sub>i+1</sub> is contacted (i ≠ d), and so on
  - To repair the neighbourhood set, request Neighbourhood set from remaining neighbours and update using new information

# **Locality of Pastry Routing**

- Entries in the Pastry routing table are chosen to be in close proximity to a node
  - Each step in a pastry route forwards a message to a node that shares a prefix with the message that is greater than or equal to that shared with the present node
  - As a result, each step moves a message closer to the destination in the nodeld space, while travelling the least possible distance (ideally?) in the proximity space
- Pastry does not route messages to achieve a globally minimum routing distance
  - Each routing decision is based on local information only
  - But the routes that are produced are "good"
  - Why?

# **Locality of Pastry Routing**

- If a message is routed from a node A to a node B at a distance d, the message cannot subsequently be routed to a node at a distance of less than d from A
  - Follows from routing procedure and (correct?) maintenance of routing information
- The expected distance travelled by a message during each successive routing step increases exponentially from one step to the next
  - Each entry in row I of the routing table is chosen from a set containing N/2<sup>bl</sup> nodes
- So, a message tends to make larger steps towards its destination in each successive routing step

# **Pastry performance**

- Routing distance (nodeld space)
  - ceil(log<sub>2</sub><sup>b</sup>N) is the expected maximum number of steps in a Pastry route
  - verified experimentally
- Effect of maintaining only partial routing tables
  - 30%-40% longer routes than complete (global) routing tables
- Replica routing
  - Many application use Pastry to route to the nearest among the k nodes numerically closest to a given destination GUID
  - Pastry locates the nearest node 76% of the time and one of the two nearest nodes 92% of the time