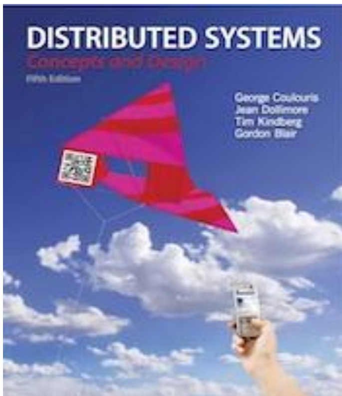
Slides for Chapter 10: Peer-to-Peer Systems



From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems: Concepts and Design

Edition 5, © Addison-Wesley 2012

Text extensions to slides © David E. Bakken, 2012

Introduction [10.1]

- Motivational observations
  - Lots of resources at the edge of the Internet
  - If there is no central server no single point/entity for
    - Copyright liability
    - Cyber-attacking
    - Getting overloaded
- Goal: design a system that is
  - Fully decentralized
  - Self-organizing
  - Uses above observations

Introduction (cont.)

- Characteristics of peer-to-peer (P2P) systems:
  - Each user contributes resources to the system
  - All nodes have the same functional capabilities and responsibilities
    - “All the animals are equal, but some more equal than others.” Animal Farm (paraphrase)
  - Correct operation does not depend on the existence of centrally administered systems
  - Can be designed to offer some kind of anonymity to providers and users of resources
  - Key issue for each: placement of data so it
    - Is spread across huge # hosts
    - Can be efficiently accessed by users’ apps
    - Balances workload
    - Ensures availability, even in the face of (relatively) volatile resources

P2P system evolution

- Antecedents of P2P systems: DNS, USENET, research systems (Xerox PARC multiple)
- Generations of P2P system+apps
  1. Napster (2001): music exchange service
  2. File sharing apps (Freenet, Gnutella, Kazaa, BitTorrent)
  3. Middleware layer support (Pastry, Tapestry, CAN, Chord, …)
- 3G P2P middleware characteristics (offloading DADs)
  - Place resources on widely-distributed set of computers
  - Guarantees of request delivery in bounded # network hops
  - Place based on volatility/availability, trustworthiness of node
  - Resources identified w/GUIDs (usually secure hash: self-certifying)
    - Client can know not tampered
    - Q: how mutable/immutable can should/can the placed objects/files be?

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.

© Pearson Education 2012

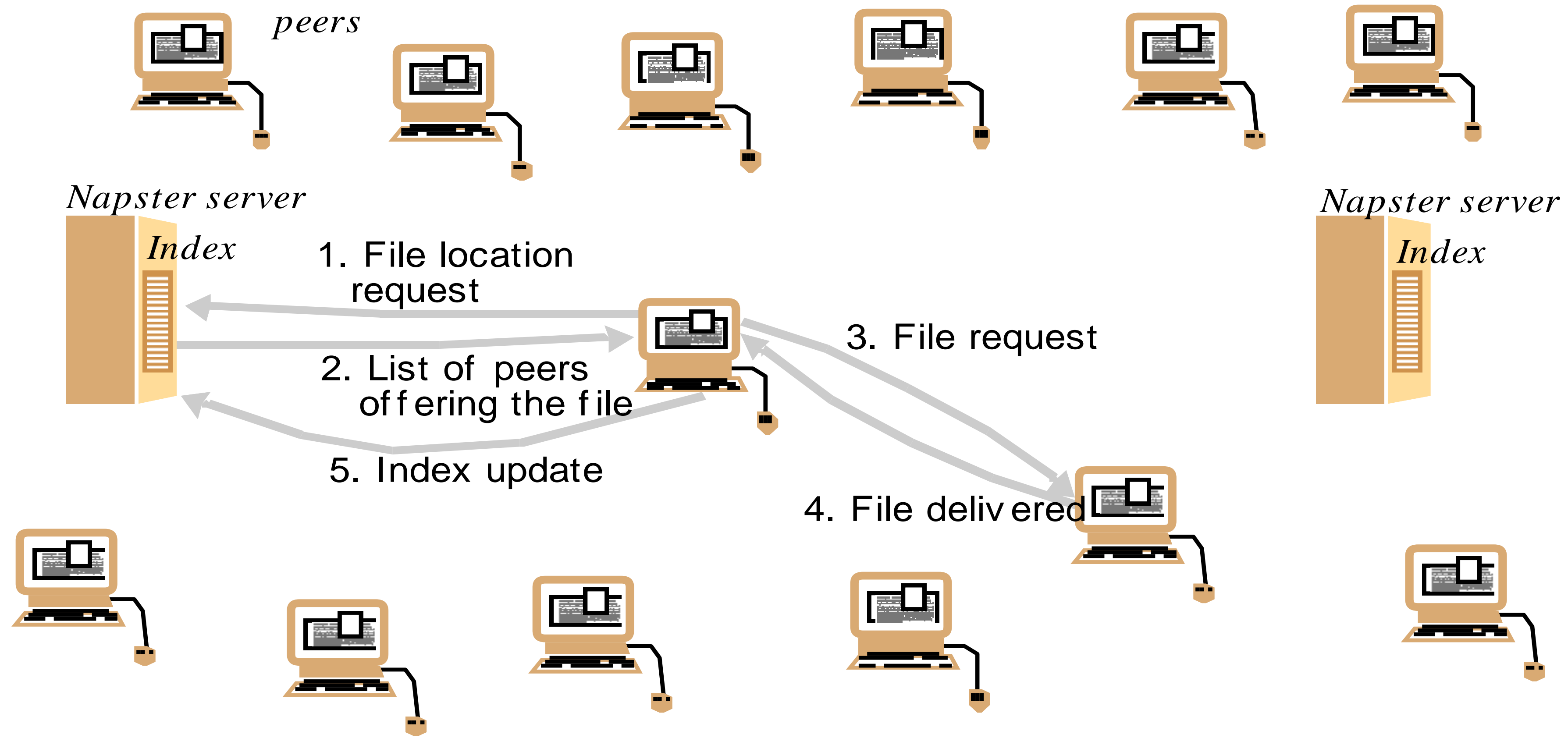
Distributed computation

- Lots of spare computing power on end-user computers
- E.g., SETI@home
  - HUGE amount of computations done
  - Unusual: no communication/coord. Between computers while processing the tasks (send to central server)
  - Other successor projects: protein folding, large prime numbers, …

Napster and its legacy [10.2]

- Napster launched in 1999 … music files mainly
- Architecture (Fig 10.2)
  - Centralized indexes
  - User supplied files (stored on their PCs)
- Shut down by legal proceedings
  - Centralized index servers deemed essential part of process
- Lessons learned
  - Feasible to build huge P2P sys. w/ (almost all) resources @edges
  - Network locality can be successfully exploited
- Limitations: replicated index not consistent. (Matters?)
- Application dependencies:
  - Music files never updated
  - No availability guarantees for a file (can download later)

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



© Pearson Education 2012

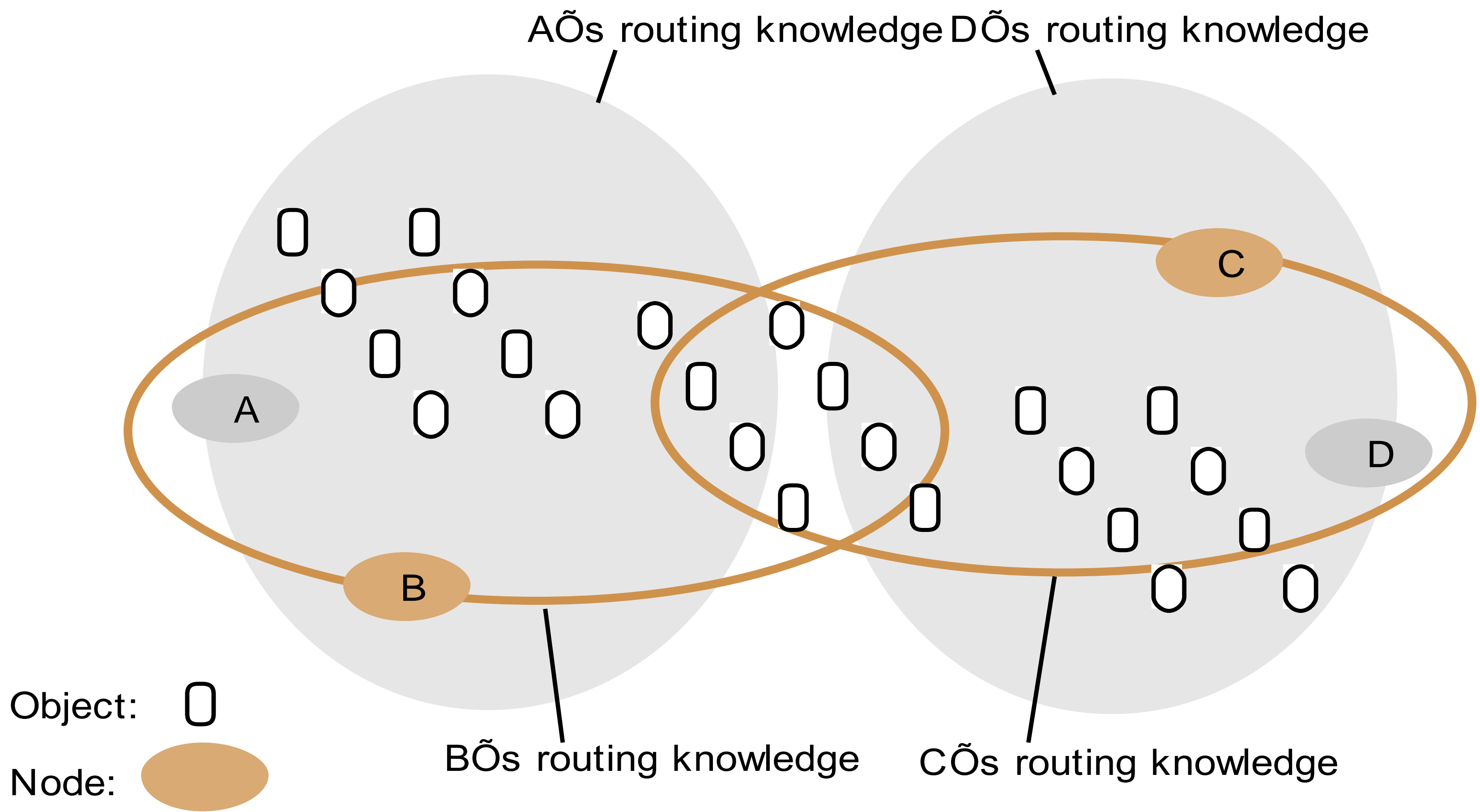
P2P MW [10.3]

- Key problem: provide mechanism to enable clients to access data resources fast & dependably from anywhere
  - 1G Napster: index replicas had complete copies of available files
  - 2G (Gnutella, freenet) partitioned & distributed indexes
- Functional requirements
  - Goal: simplify construction of P2P services
  - Enable clients to locate+get any individual resource
  - Add & remove resources
  - Add & remove participating hosts
  - Simple API independent of resource/data types

P2P MW (cont.)

- Non-functional requirements
  - Global scalability
  - Load balancing
  - Optimize for local interactions between neighboring peers
  - Accommodation for highly dynamic host availability
  - Security of data where (dynamic) trust varies widely
  - Anonyminity, deniability, resistance to censorship
- Scalability ➔must partition knowledge of location of objects through network
  - Must be replicated (up to 16 times)
- All above is a very active area of research!

Figure 10.3: Distribution of information in a routing overlay



© Pearson Education 2012

Routing overlays [10.4]

- **Routing overlay (RO)**: locates nodes and objects
  - May have been relocated, node down, …
  - Any node can access any object by routing request through series of nodes
  - GUIDs are “pure names” (AKA opaque identifiers): random bit patterns with no structure
- Tasks for the RO (AKA **distributed hash tables**)
  - Route requests to objects
  - Inserting objects:
    - Compute GUID (from part or all of state); probabilistically unique (why?)
    - Announce new object to RO
  - Delete objects
  - Node tracking: (rough) group membership on nodes

Routing overlays (cont.)

- DHT model: data item/object with GUID X stored at node
  - Whose GUID is numerically closest to X, and
  - R hosts whose GUIDs are next-closest numerically
- Distributed object location and routing (DOLR) model:
  - Location for replicas decided outside the routing layer
  - DORL layer notified of host addresses of each replica with `publish()` operation
- Both DHT (Pastry) and DOLR (Tapestry) use prefix routing
  - Uses part of GUID to find next node in path towards object

Routing overlays (cont.)

- Other routing schemes (all have distributed index schemes)
  - Chord: distance between GUID of selected node and the dest.
  - CAN: n-dimensional hypersace
  - Kademlia: XOR of pair of GUIDs as distance metric
  - BitTorrent: index → stub file with GUID, URL of tracker → file

Figure 10.4: Basic programming interface for 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.

© Pearson Education 2012

Figure 10.5: Basic API 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.

© Pearson Education 2012

Overlay case studies: Pastry, Tapestry [10.5]

- 10.5: overlay case studies
  - Pastry: straightforward and effective
  - Tapestry: more complex, supporting wider range of locality approaches
- 10.6: “Application” case studies: Squirrel, OceanStore, Ivy
  - “Application”, some really higher-level middleware (a service)
  - Squirrel/Pastry: web cache
  - OceanStore/Tapestry: file storage
  - Ivy/overlay: file storage

Pastry routing overlay [10.5.1]

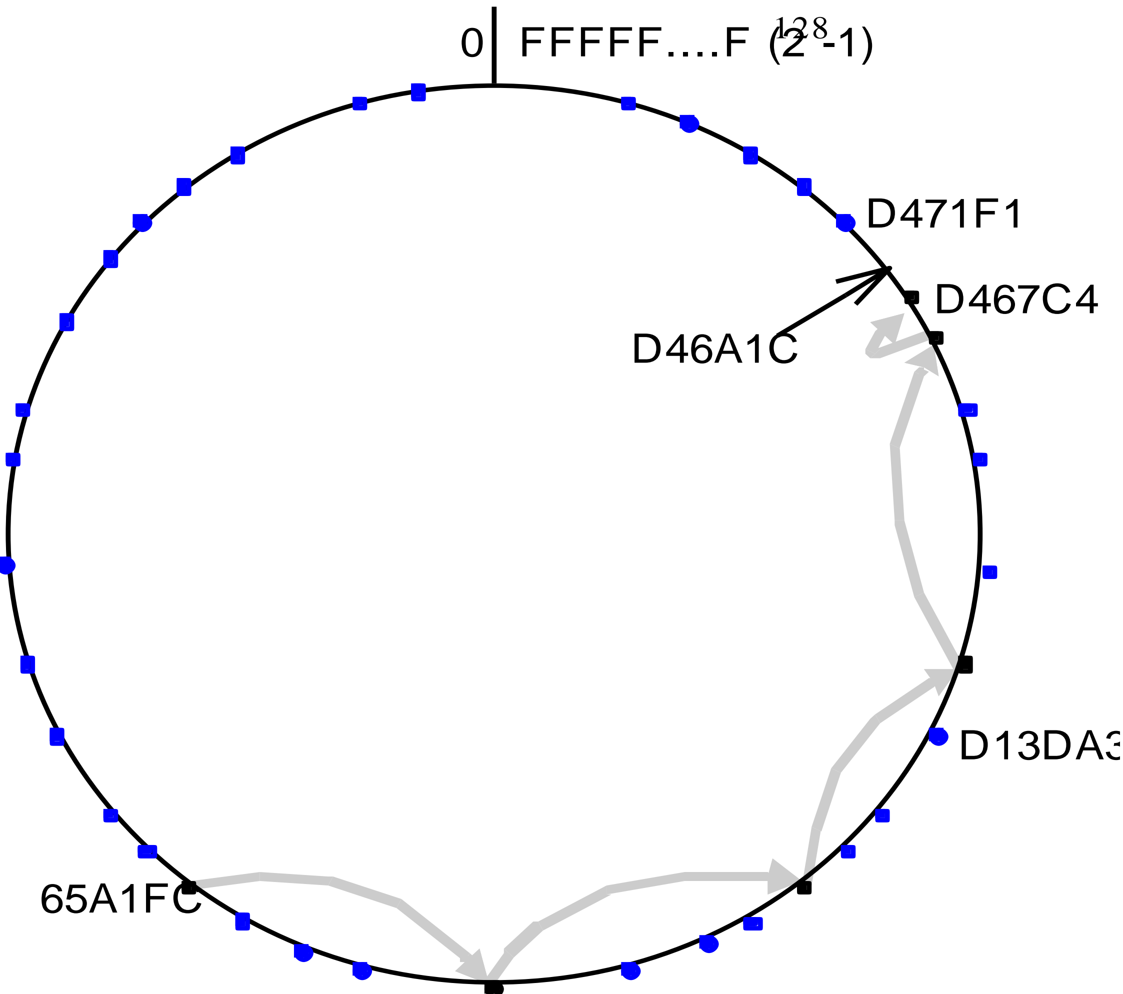
- Main characteristics as per 10.4
  - All nodes and objects assigned 128-bit GUID
    - Nodes: secure hash over node's public key (provided)
    - Objects, secure hash or over name or part of data
    - (Very unlikely) clashes in GUIDs detected and remediated
  - Scalability: network with N nodes, pastry courts to GUID in *O(log n)* steps
    - Node inactive at last hop: finds nearest active one
    - Active nodes route to numerical nearest neighbor
    - Naively, "nearest" and *O(log N)* are in terms of logical overaly topology, not network!
    - But uses locality metric based on network distance
    - Can scale to thousands of hosts
    - New nodes can construct their routing table in *O(log N)* messages
    - Same complexity for detecting and reconfiguring with failure

Pastry routing algorithm

- Buildup: describe simplified, inefficient form (Stage 1), then full (Stage 2) with routing table
- Stage 1
  - **Leaf set** (each active node): vector *L* (size *2l*) with GUIDs & IP addrs of numerically closest nodes (*l* above and *l* below)
    - *l* is typically 8
  - Failure recovery fast: leaf set reflects current (configuration) state (up to some max rate of failures)
  - GUID space circular: $[0, 2^{128-1}]$ with wraparound
  - Routing is trivial
    - Compare GUID of incoming message to node's own
    - Send to node in L closest to message's GUID (likely ~*l* nodes closer)
  - Inefficient: requires ~N/2l (logical) hops on average

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 (2128-1). 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.

© Pearson Education 2012

Full Pastry routing algorithm (Stage 2)

- Each node maintains tree-structured routing table
  - GUIDs and IP addrs spread throughout the 2^{128} GUID space
  - Not uniformly spread: more dense closer to the node's GUID
- Structure of routing table
  - GUIDs viewed as hex values, classified by prefixes
  - As many rows as hex digits in GUID, e.g., 128/4=32
  - Each row contains 15 entries, one for each value of the digit except for the one for the host node's GUID
  - Each table entry points to one of the multiple nodes whose GUIDs have the relevant prefix (the one trying to forwards towards)
    - Non-null: contains [GUID, IP addr] of a node with a longer prefix than node
    - Null: no such node known: forward to any node in leaf set or routing table with same prefix length but numerically closer

Figure 10.7: First four rows of a Pastry routing table

	p =	*GUID prefixes and corresponding nodehandles n*															
(Z)	*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*
(X)	*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*
(W)	*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*
(Y)	*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*

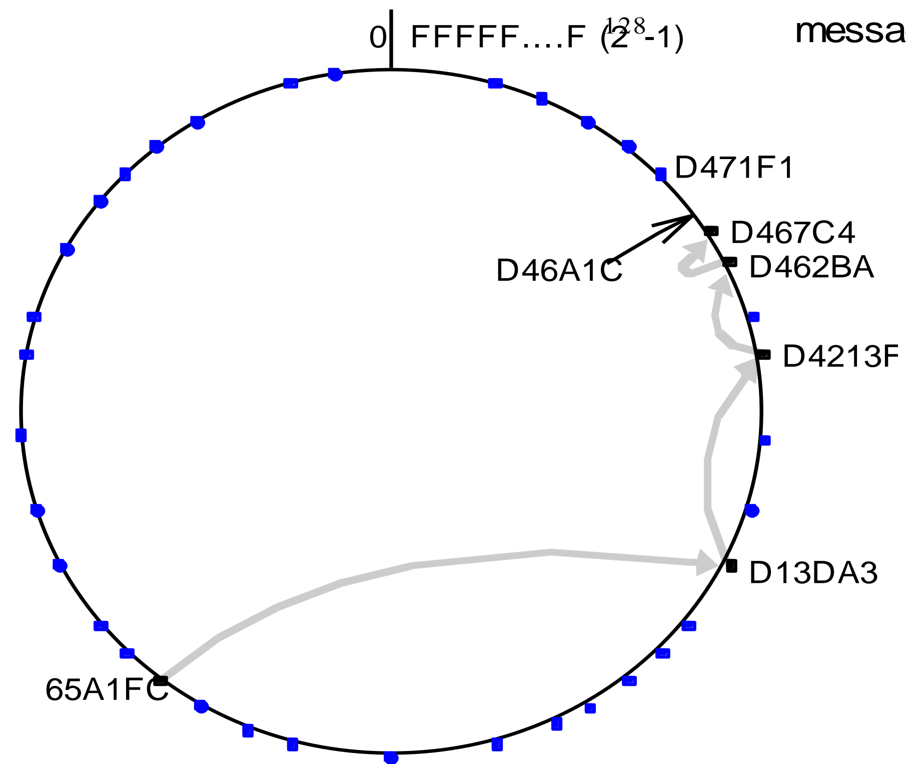
The routing table is located at a node whose GUID begins 65A1. Digits are in hex. The *n*'s represent [GUID, IP address] pairs specifying the next hop to be taken by messages addressed to GUIDs that match each given prefix. Grey- shaded entries indicate that the prefix matches the current GUID up to the given value of *p*: the next row down or the leaf set should be examined to find a route.

E.g., from **65A1** to **653**2 **(W)** or **60**14 **(X)** or **65A3** **(Y)** or **8**2CB **(Z)**

© Pearson Education 2012

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.

© Pearson Education 2012

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.

   }

}

© Pearson Education 2012

Pastry host integration

- New nodes use joining protocol
  - get R and L
  - Lets other nodes know they must change cause they are joined
  - Steps at node X
    1. Compute suitable GUID for X
    2. Make contact with (locally) nearby Pastry node (A), send a **join request** to it
       - Destination = X (!!!)
    3. Pastry routes request to node with GUID numerically closest to X (Z)
       - E.g., route is A→B→C→D→Z
       - Along way, {A,B,C,D,Z} send relevant parts of their R and L to X
- Text has more details on the properties it provides (not covering but testable)
  - E.g., A's first row good candidate for first row of X
  - E.g., Z is numerically closest to X, so its L is good candiate for X's
  - Initial entries updates as per discussion below on fault tolerance

Pastry host failure or departure

- Node X deemed failed when node N can’t contact X
- Must repair leaf sets (L) containing X
  - N finds live node close to X, gets it leaf set L’
  - L’ will partly overlap L, find one with appropriate value to replace X
  - Other neighboring nodes informed, they do same
- Repairing routing tables on “when discovered basis”
  - Routing still works if some nodes in table not live, try if fails

Pastry locality and fault tolerance

- Locality
  - Routing structure highly redundant: many paths from X to Y
  - Construction of R tries to use most “local” routes, ones closes to actual network topology (more candidates than fit in R)
- Fault tolerance
  - Assumed live until can’t contact
  - Nodes send **heartbeat messages** to left neighbor in L
    - That won’t spread to a lot of nodes very fast…
    - Also does not hand malicious nodes trying to thwart correct routing
  - Ergo, clients needing reliable delivery use at-least-once delivery mechanism, repeating multiple times if no response
    - Gives Pastry more time to fix L and R
  - Other failures or malicious nodes: add tiny amount of randomness to route selection (see text)

Pastry dependability

- MSPastry uses same routing algorithm, similar host mgt
  - Adds dependability measures,
  - Adds performance optimizations for host management algorithms
- Dependability measures
  - Use ACKS each hop in routing; timeout ➔ find alt. route & suspect
  - Heartbeat message timeout by detecting node (neighbor to right)
    - Contact rest of nodes in L: node failure notify, ask for replacements
    - Terminated even with multiple simultaneous failures
- Performance optimizations
  - Run simple gossip protocol periodically (20 min) exchange info
    - Repair failed entries
    - Prevent slow deterioration of locality properties

Pastry evaluation

- Exhaustive (simulation) performance eval of MSPastry
  - Looked at impact on performance and dependability:
    - Join/leave rate
    - Dependability mechanisms used
- Dependability results
  - 0% IP message loss rate ➔ failure to deliver 1.5/100K requests
  - 5% IP message loss rate ➔ failure to deliver 3.3/100K requests, 1.6/100K delivered to wrong node
    - (Per-hop ACKS ensure eventually get there)
- Performance results (scale to thousands of nodes)
  - Metric: stretch (relative delay penalty): ratio over direct UDP/IP ➔ ~1.8 with 0% msg loss, ~2.2 with 5% network msg loss
- Overhead: control traffic less than 2 msgs/minute/node

Tapestry [10.5.2] ➔ LATER, in last installment of slides