Peer-to-Peer Systems Case Study: Pastry & Tapestry

Reference: George Coulouris, Jean Dollimore and Tim Kindberg, "Distributed Systems Concepts and Design", Fifth Edition, Pearson Education, 2012

Case Studies: Pastry

- Nodes & objects assigned 128 bit GUID computed by applying Secure Hash Algorithm to node's public key or object's name or stored state.
- GUIDs randomly distributed in range 0 to 2^128 -1
- Provide no clue to value from which computed
- Clashes between GUIDs for different nodes are unlikely
- If GUID identifies node currently active, message delivered to it else to node whose GUID is numerically closest
- Delivery in O(log N) steps
- Routing uses underlying transport to transfer message to node closer to destination (may involve many IP hops)
- Pastry uses locality metric based on hop count or delay in underlying network to select appropriate neighbors when setting up routing tables

Distributed Hash Table API in Pastry

put(GUID, data)

- The *data is stored in replicas at all nodes responsible for the object* identified by *GUID.*

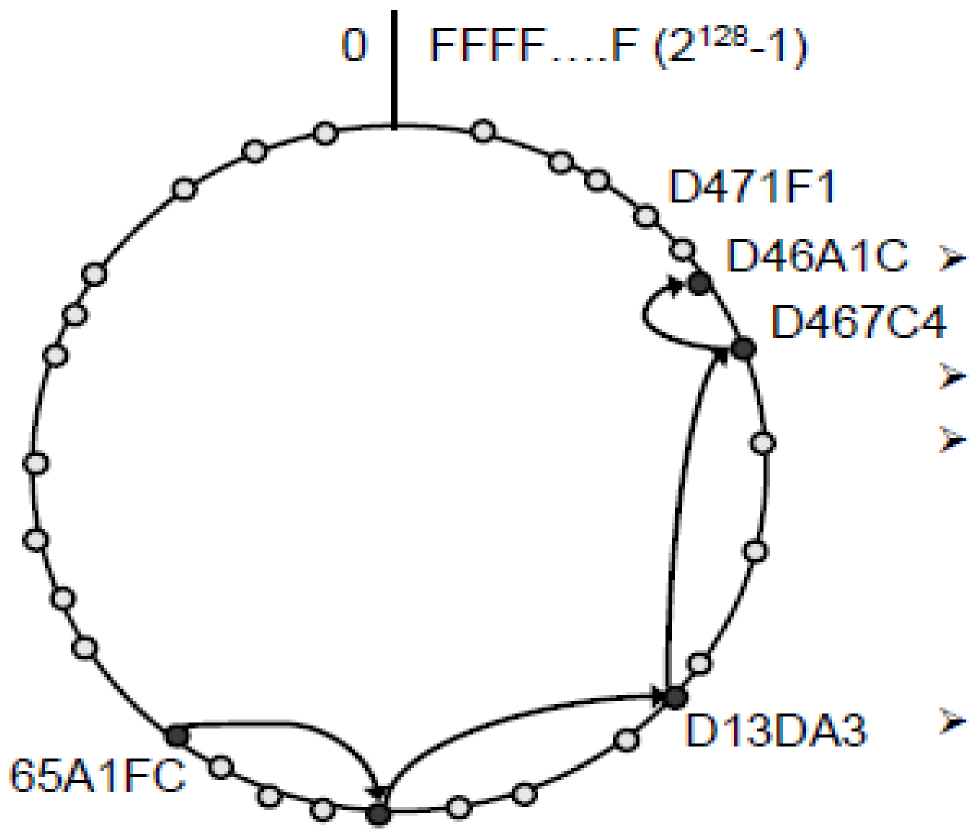
remove(GUID)

- 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 for it.

Simple Pastry Routing Algorithm



- Each node stores leaf set – vector L (of size 2l) containing GUIDs and IP addresses of l nearest nodes above and l below its GUID
- GUID space is circular: 0's neighbour is 2^{128} -1
- The dots depict live nodes.
- 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.

Simple Pastry Routing Algorithm

- Each Pastry node maintains a tree structured routing table giving GUIDs and IP addresses for a set of nodes spread throughout the entire range of 2^128 possible values with increased density of coverage for GUIDs numerically close to its own.
- For GUIDs represented as hexadecimal numbers, routing table has as many rows as hex digits in a GUID = 128/4 = 32 rows
- Any row has 15 entries - one for each possible value of the nth hex digit excluding the value in the local node's GUID
- Each entry in table points to one of the potentially *many* nodes whose GUIDs have relevant prefix

First 4 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.

Pastry Routing Algorithm

if (destination is within range of our leaf set)

forward to numerically closest member

else

if (there is a longer prefix match in table)

forward to node with longest match

else

forward to node in table which

(a) has a common prefix of length p and

(b) GUID that is numerically closer.

Tapestry

- Tapestry implements a distributed hash table and routes messages to nodes based on GUIDs associated with resources using prefix routing in a manner similar to Pastry.
- Tapestry applications give additional flexibility:
- They can place replicas close (in network distance) to frequent users of resources in order to reduce latency and
- Minimize network load or to ensure tolerance of network and host failures

Distributed Object Location and Routing in 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 as host for* the object corresponding to *GUID.*

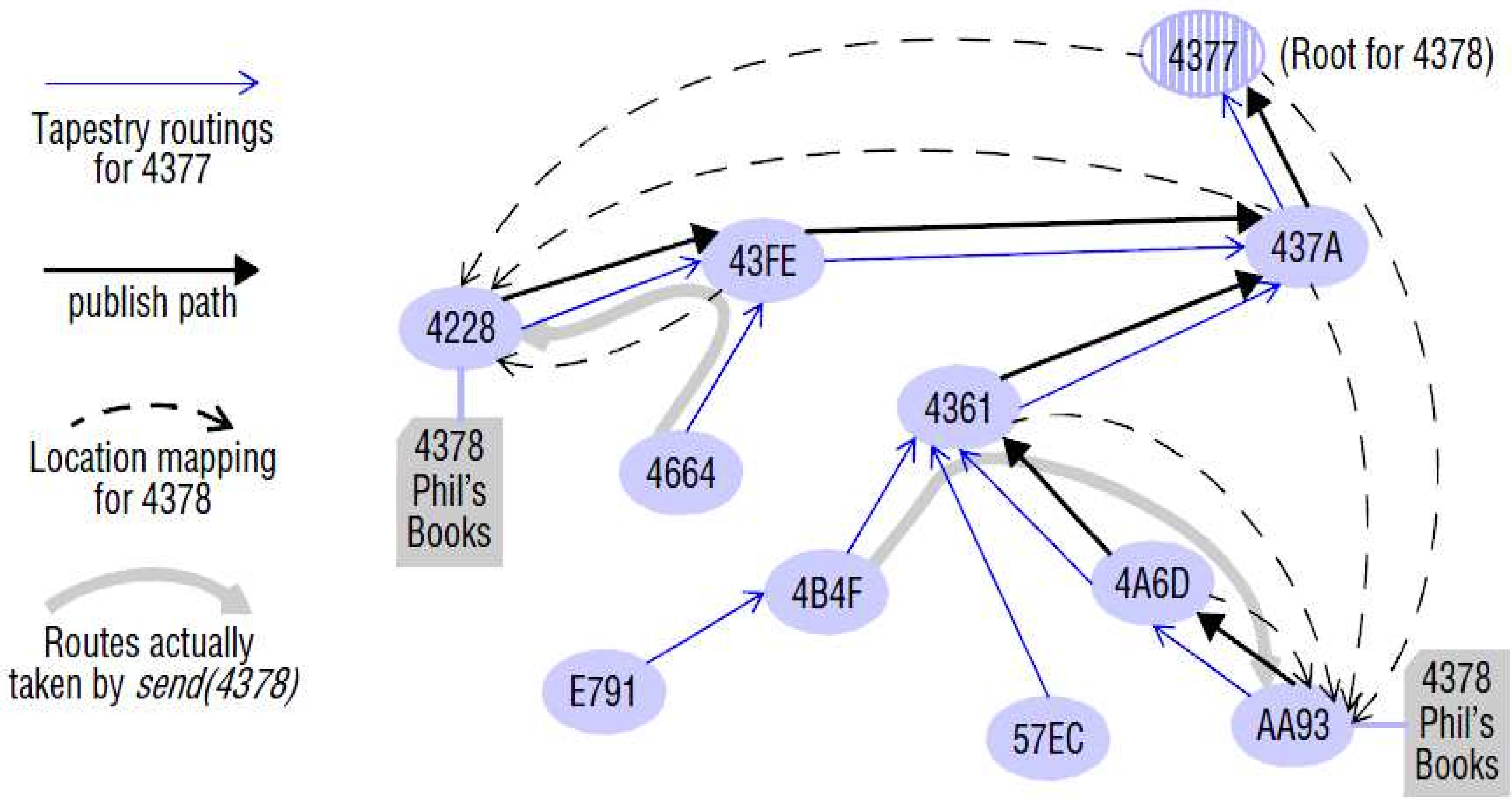
unpublish(GUID)

- 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.

Distributed Object Location and Routing in Tapestry



Structured Vs Unstructured P2P

	Structured peer-to-peer	*Unstructured peer-to-peer*
Advantages	Guaranteed to locate objects (assuming they exist) and can offer time and complexity bounds on this operation; relatively low message overhead.	Self-organizing and naturally resilient to node failure.
Disadvantages	Need to maintain often complex overlay structures, which can be difficult and costly to achieve, especially in highly dynamic environments.	Probabilistic and hence cannot offer absolute guarantees on locating objects; prone to excessive messaging overhead which can affect scalability.

Strategies for Search

- In P2P file sharing, all nodes in the network offer files to other nodes.
- Searching for a file in unstructured P2P network follows following strategies.

1. Expanded Ring Search
2. Random Walks
3. Gossiping.

Thank You