



Brief Summary on Topology and Performance of Distributed Hash Tables

Zhirong Yang

Helsinki University of Technology

rozyang@cc.hut.fi



Agenda

- Introduction
- Basic DHTs
 - Pastry (mentioned later)
 - CAN (coming soon)
 - Tapestry (omitted)
 - Chord (in detail)
- Newly proposed designs
 - Heterogeneity (mOverlay, MDHT, Expressway)
 - Churn (Bamboo)
 - Routing table size vs. network diameter (Ulysses)
 - Hot-spot problem (YAPPERS)
- Conclusion



DHT-based Application Examples

- Cooperative mirroring
- Simultaneous downloading
- Time-Shared storage
- Keyword search

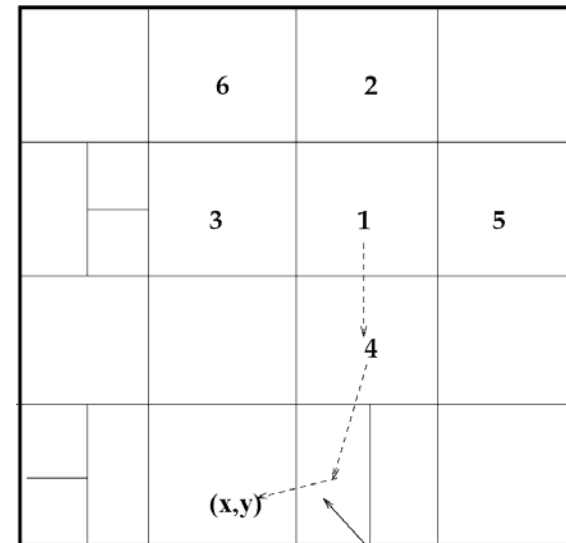
All the above applications rely on one operation:

- given a key, look up the node(s) containing corresponding value

Query principles

- Both nodes and keys are hashed into a virtual space
- Each node is responsible for a zone nearby which contains some keys
- The query can be launched from any node in the system, but the result is deterministic.
- The routing from originating node to destination node is done in an asymptotic manner.

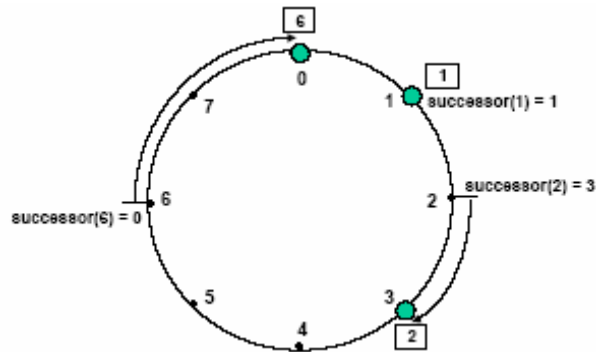
CAN as an example



sample routing path from node 1 to point (x,y)

routing table size $O(d)$
lookup cost $O(dN^{1/d})$

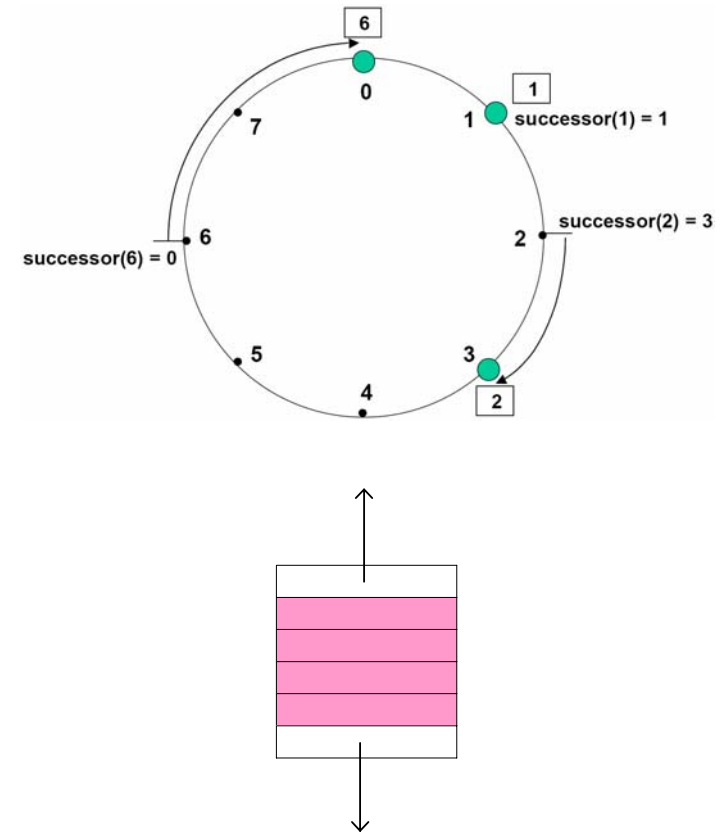
Chord(1)



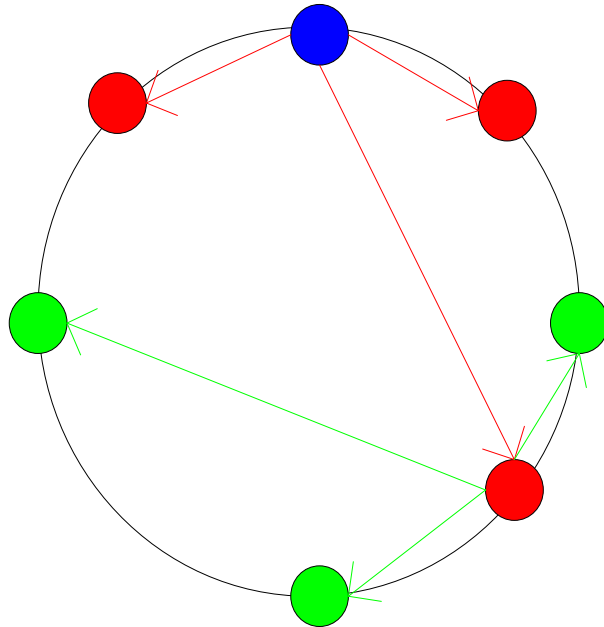
Notation	Definition
$finger[k].start$	$(n + 2^{k-1}) \bmod 2^m, 1 \leq k \leq m$
$.interval$	$[finger[k].start, finger[k+1].start)$
$.node$	first node $\geq n.finger[k].start$
$successor$	the next node on the identifier circle; $finger[1].node$
$predecessor$	the previous node on the identifier circle

Chord(2)

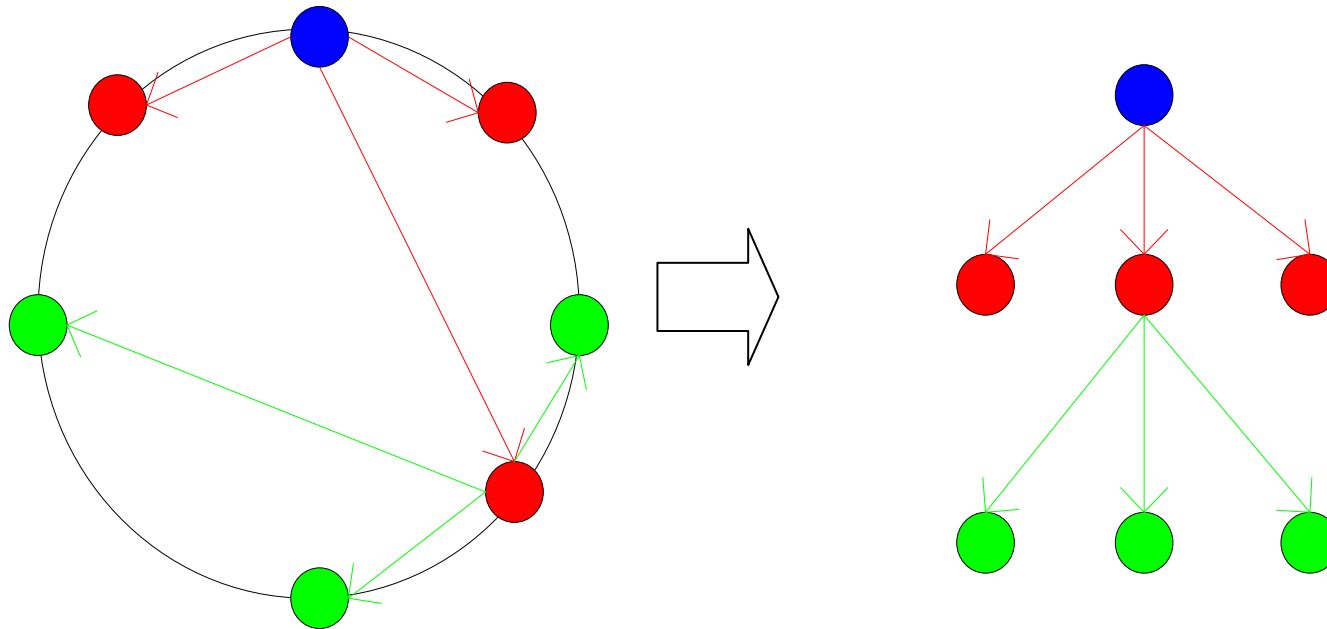
```
// ask node n to find id's successor  
n.find_successor(id)  
  n' = find_predecessor(id);  
  return n'.successor;  
  
// ask node n to find id's predecessor  
n.find_predecessor(id)  
  n' = n;  
  while (id ∉ (n', n'.successor])  
    n' = n'.closest_preceding_finger(id);  
  return n';  
  
// return closest finger preceding id  
n.closest_preceding_finger(id)  
  for i = m downto 1  
    if (finger[i].node ∈ (n, id))  
      return finger[i].node;  
  return n;
```



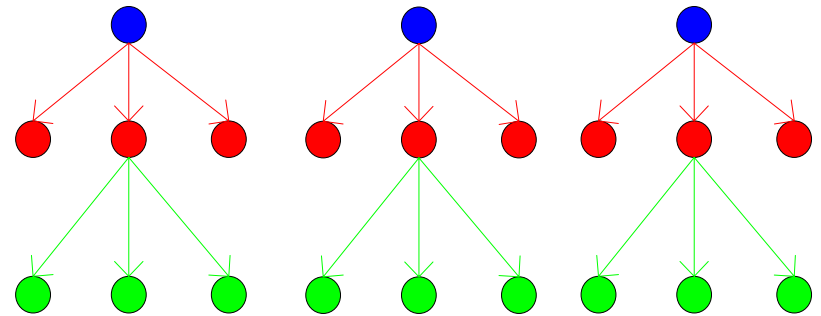
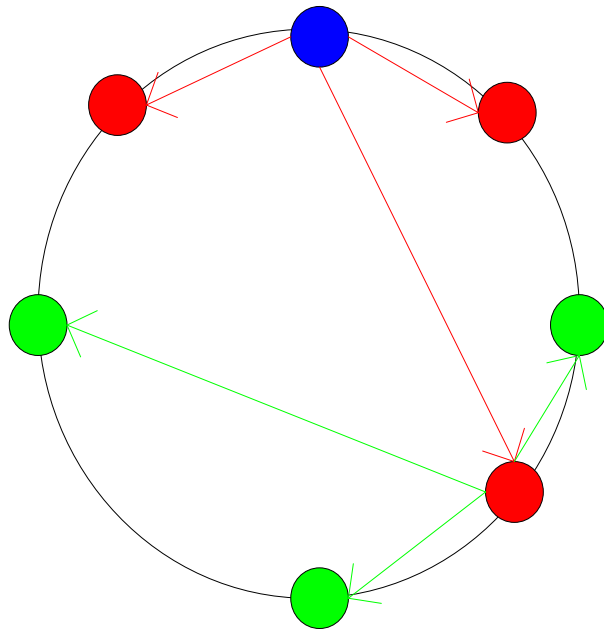
Chord(3)



Chord(3)



Chord(3)





Maintenance

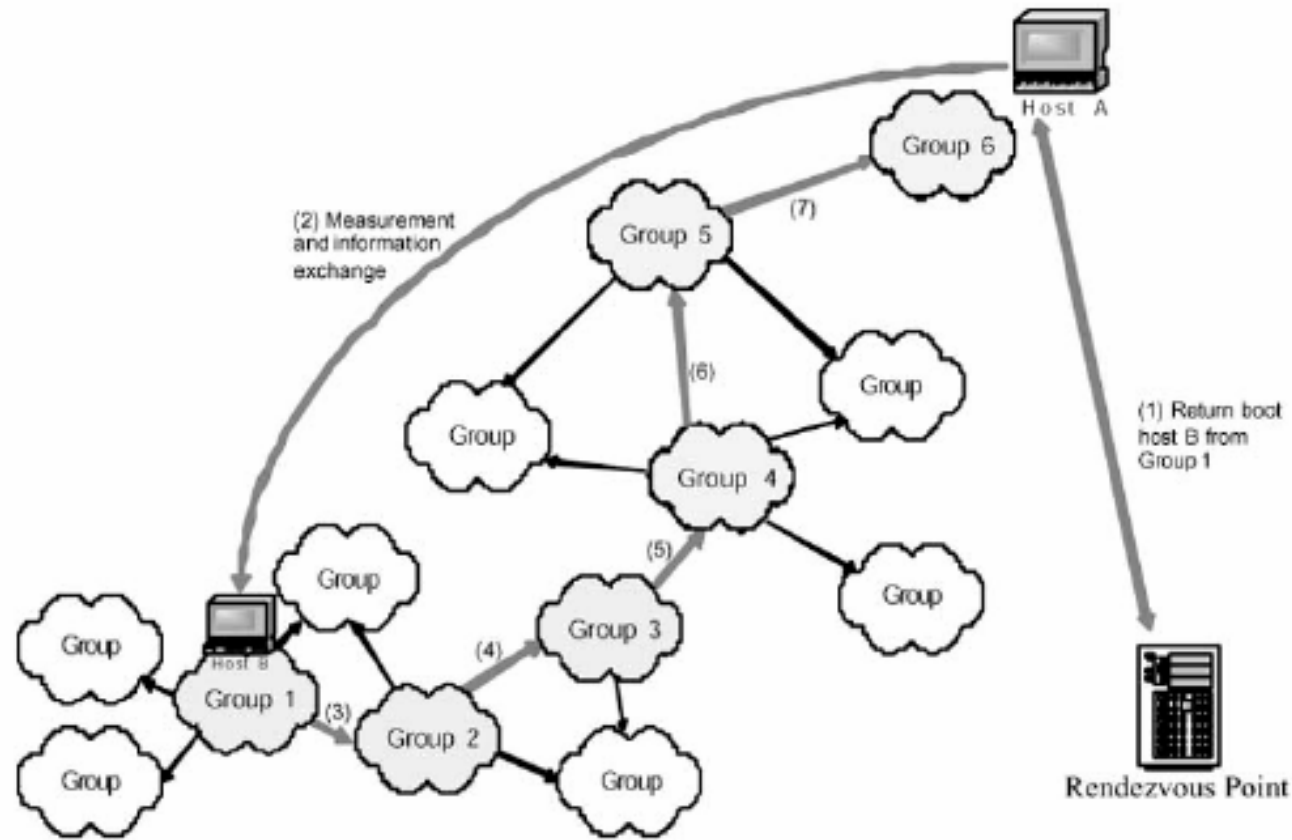
- simple \neq good
- Tradeoff between simplicity and data redundancy depends on what kind of applications the DHT is designed for.
- Two categories of strategies: event-driven vs. periodical contacts



Heterogeneity

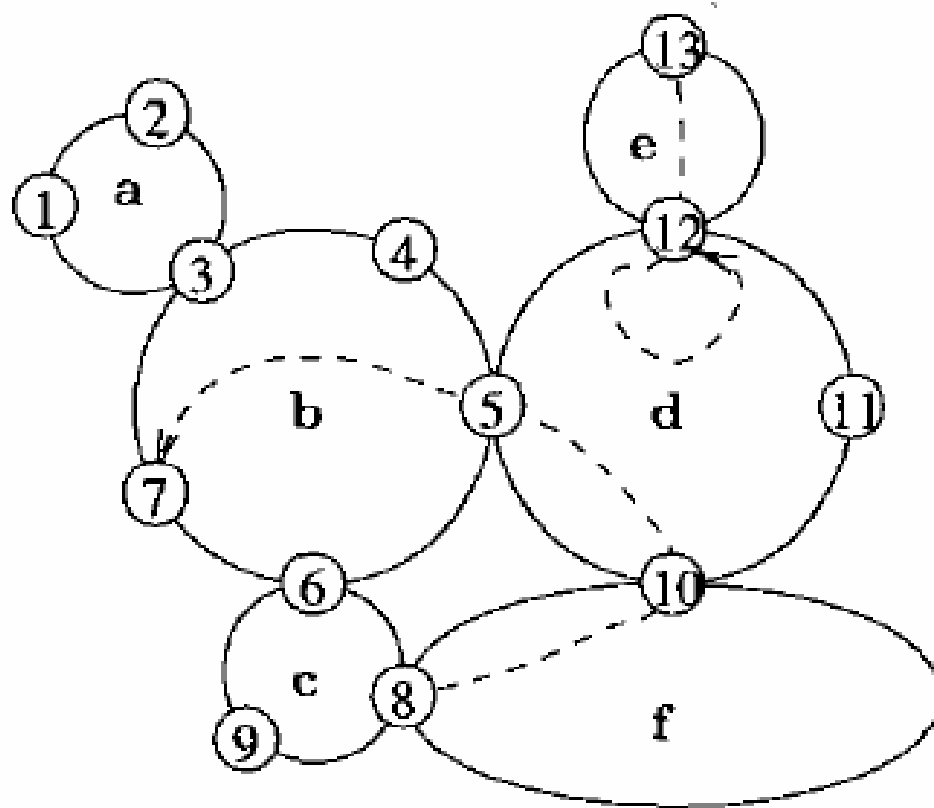
- Many DHT designs tend to treat the network homogenous, whereas there are always reasons to break the symmetry.
- It seems beneficial to take some knowledge from underlying network into account.
- Locality is addressed in this paper.

mOverlay

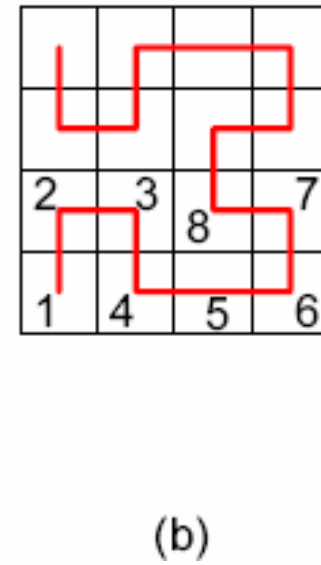
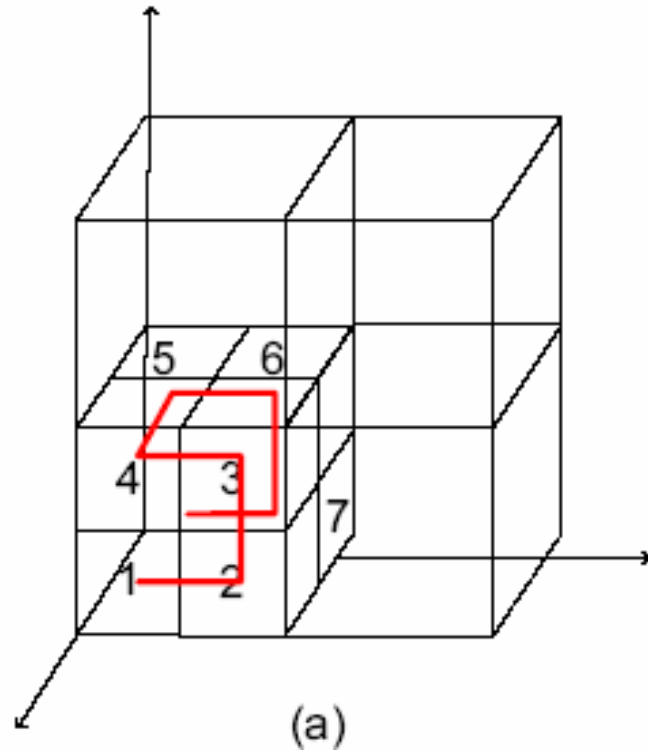




MDHT



Expressway



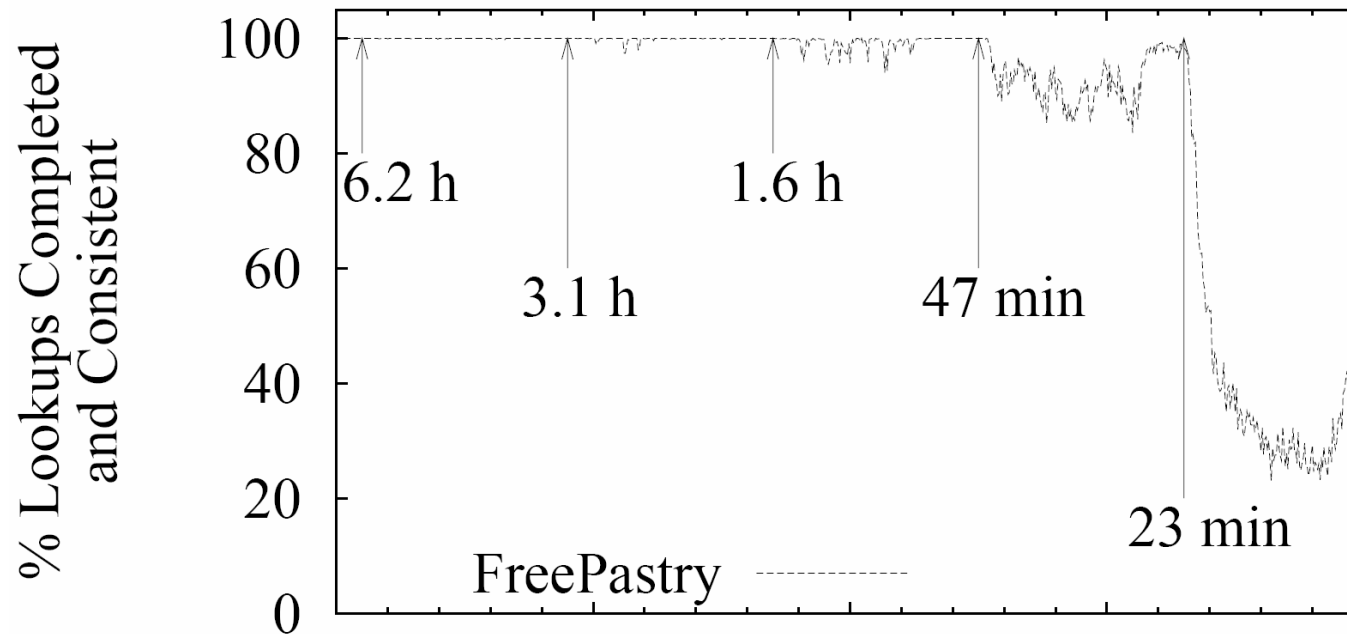


Disadvantages

- Complicates routing and maintenance;
- Against decentralization: the robustness of system heavily depends on the limited amount of host cache or bridges;
- It is impossible to elect distinguished nodes in some applications.

Churn

Churn — the continuous process of node arrival and departure.



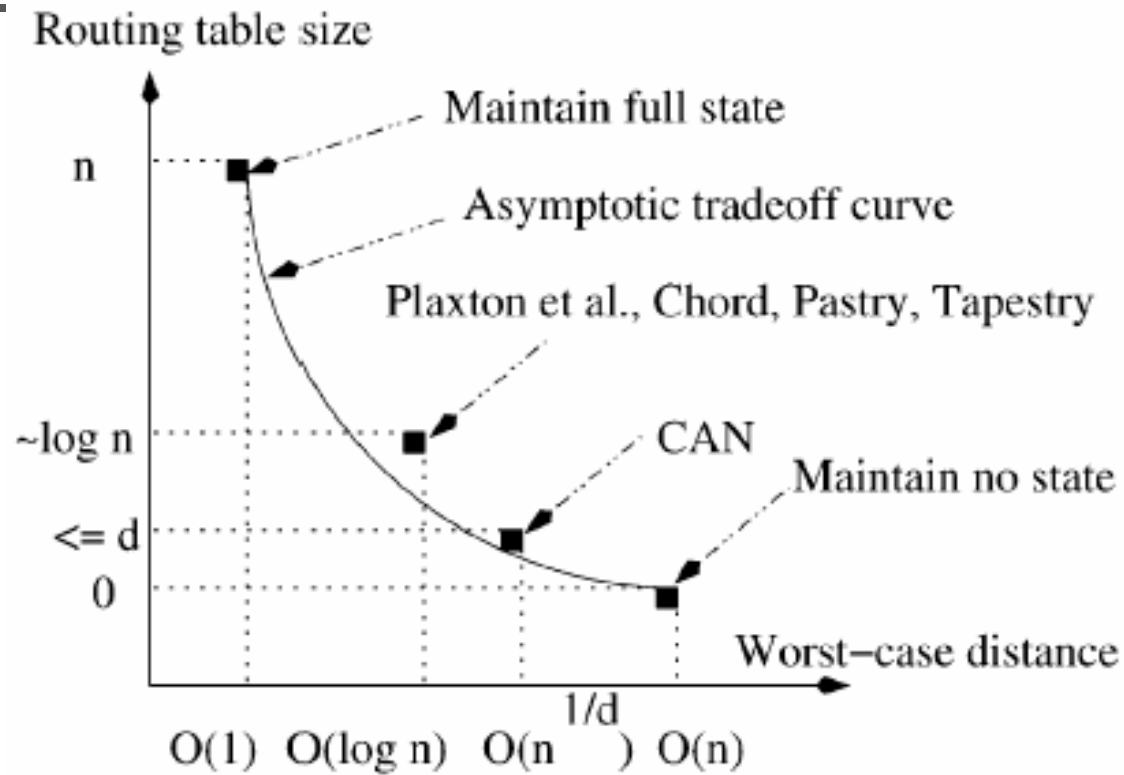
FreePastry network under increasing levels of churn: percentage of lookups that complete in a 1000-node



Bamboo's strategies

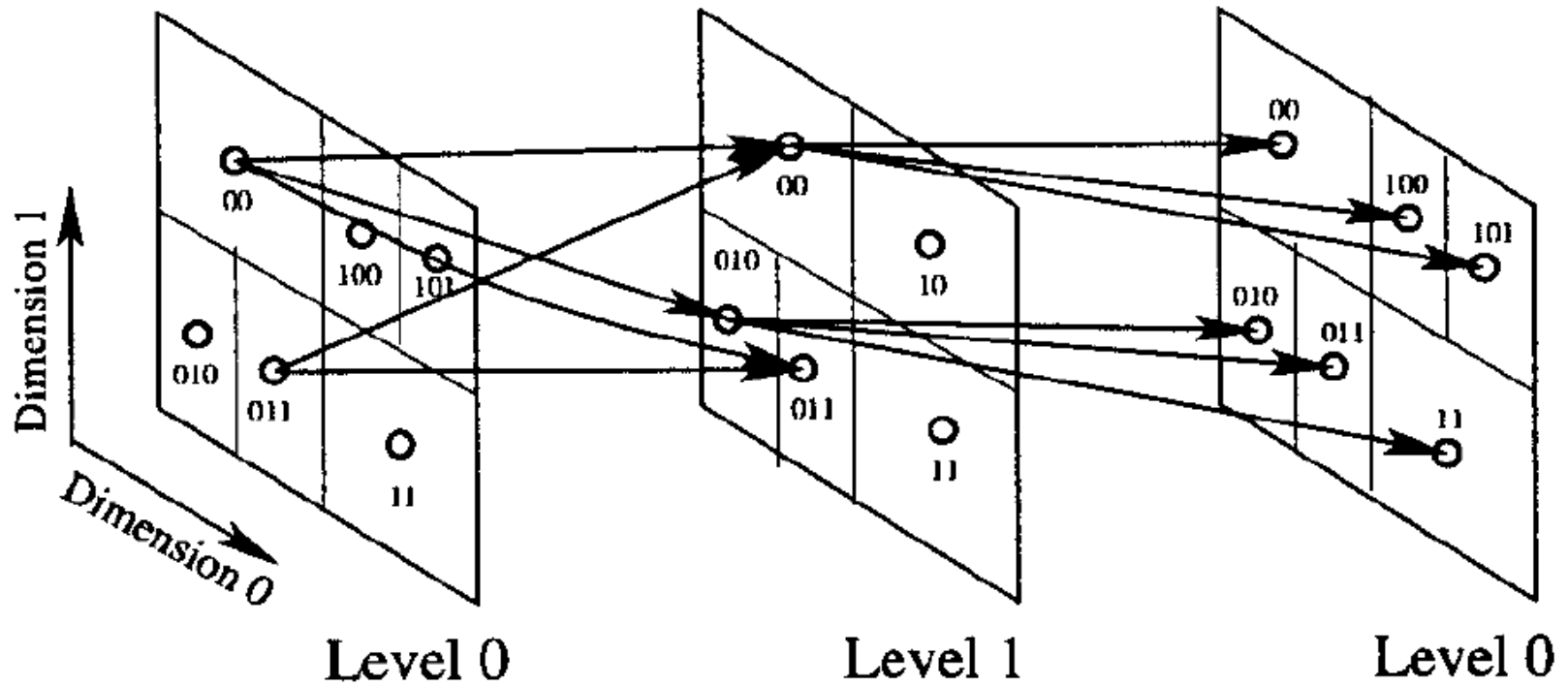
- Extends the design of Pastry, using multiple paths to handle failures and congestion.
- Simplifies the immediate joining procedure.
- Active periodical contacts between nodes:
- Employs recursive lookup instead of iterative lookup to get more exact timeout threshold.

Routing table size vs. network diameter





Ulysses





Hot-spot problem & YAPPERS

- Many DHTs are subject to hot-spot problem.
- YAPPERS solves this by simple buckets:
 - the keys are grouped into a number of buckets
 - A node with IP address IP_x is assigned key k if $\text{HASH}(k) \equiv (\text{HASH}(IP_x) \bmod b)$
- The lookup request is flooded to all the neighbors containing that key.

DHTs covered in this paper

