P2P Network
Structured Networks (IV)
Distributed Hash Tables

Pedro García López

Universitat Rovira I Virgili

Pedro.garcia@urv.net

Departament d'Enginyeria
[DEIM]
Informàtica i Matemàtiques
Universitat Rovira i Virgili

Index

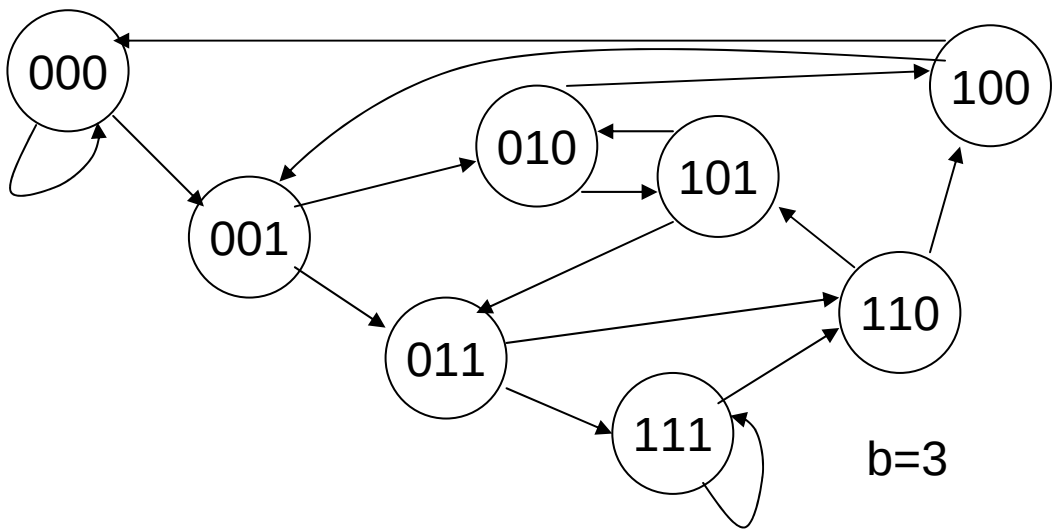
- Koorde: Degree optimal DHT
- Constructing overlay applications and services

Koorde: Degree Optimal DHT

- Parallel Architectures: Hypercubes, Butterflies, De Bruijn graphs, ...
- Koorde is a degree optimal DHT based on De Bruijn graphs
- It is very difficult to improve this !!!
  - O(1) degree & O(log) routing

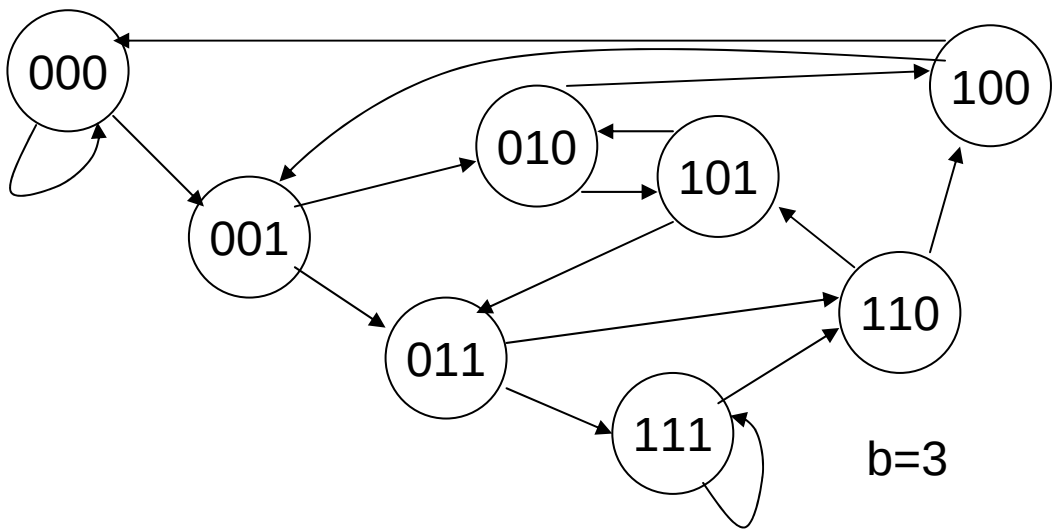
De Bruijn graph

- Each node connects to node $2m \bmod 2^b$ and node $2m+1 \bmod 2^b$ (2 links !!!)
  - Node x = 001 (1)
    - 2m (x<<1): 2
    - 2m + 1 (x<<1 +1): 3



Routing in the graph

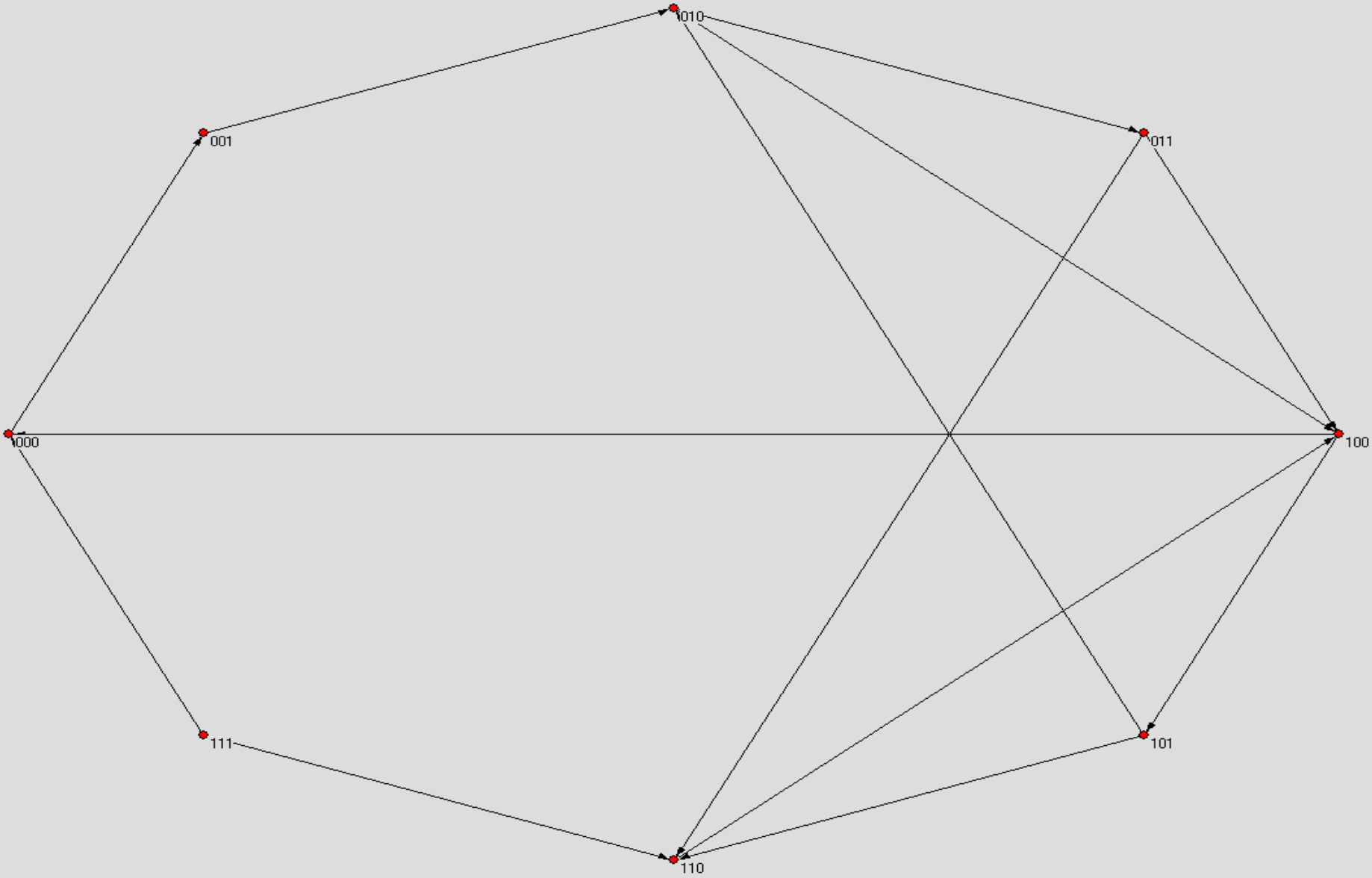
```
m.lookup(k,ks)
        if k=m
          return m
        else
          t=m ◦ topbit(ks)
          return t.lookup(k,ks<<1)
```



De Bruijn Overlay

- Difficult to maintain and construct
- Technique to construct koorde:
  - DeBruijn ring
  - Each node has 2 links:
    - succesor link
    - De Bruijn 2m link
  - But:
    - Where is the 2m +1 ??
    - We have to change the routing protocol

Koorde ring



Koorde routing

```
m.lookup(k,ks,i)
1     if k ∈(m, m.successor] return successor
2     else if i ∈(m, m.successor]
3               return d.lookup(k,ks<<1, i°topbit(ks))
4             else
5               return m.successor.lookup(k,ks,i)
```

1) If my succesor is the responsible of the key return successor
2) If exists the De Bruijn 2m link from this node to imaginary node I take the 2m hop

   i = 3, m = 2 OK (2-6-3 in De Bruijn)

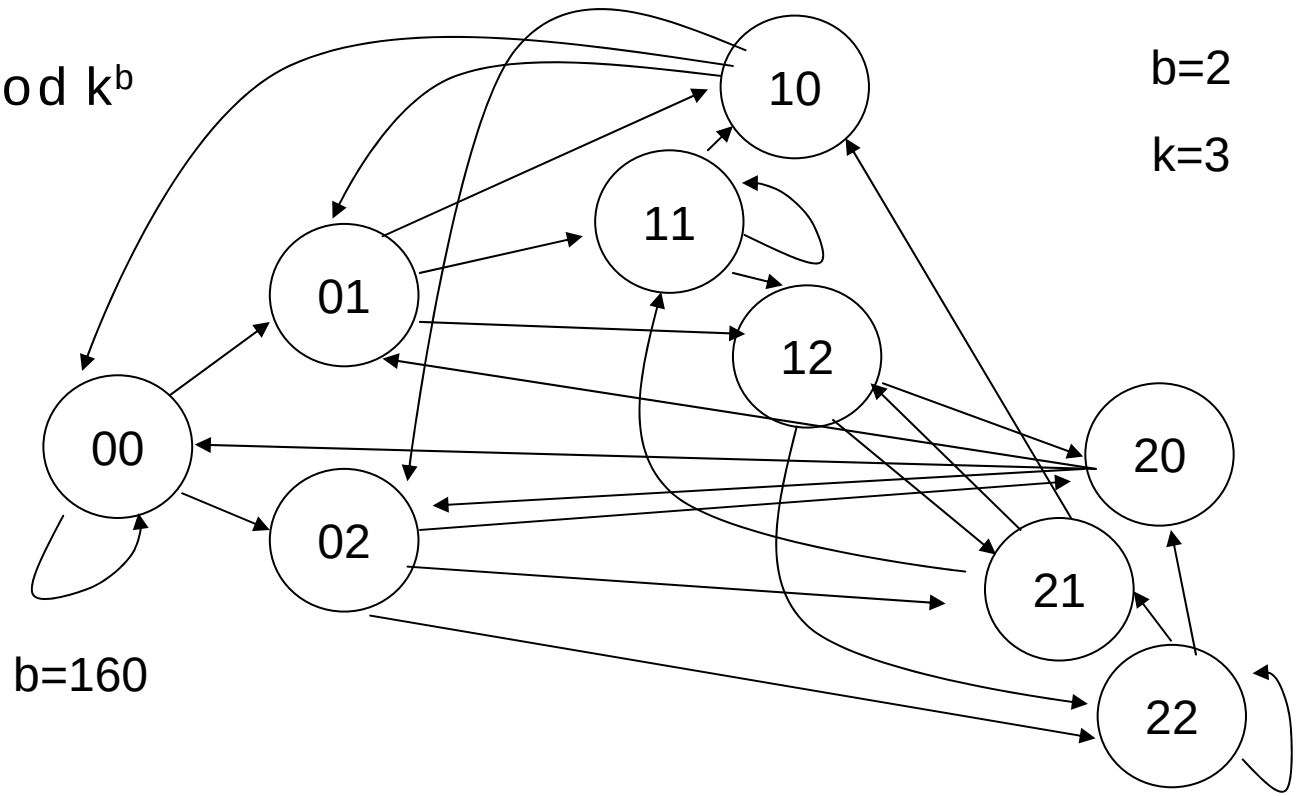
   i = 5 m = 2 NO ! (5 =2*2 +1, Koorde only has 2m links !)
5) If I do not find a De Bruijn hop, I go to the successor node.

Improvement

- This aproach is inefficient O(3b)
- Example: 2 (010)->6 (110)
  - De Bruijn Routing (2,5,3,7)(010-101-011-110)
  - 2.Lookup(6,6,i=5)
    - 2->3->4(i=5)->0(i=3)->1->2(i=3)->4->**5**->6 !!!!!!!!
- Improvement:select the i that is closer to the target and is in range (m,msucc)
  - 2.lookup(6,6,i=3)
    - 2->4->**5**->6

Koorde Base-K

- De Bruijn Base K
  - km mod k^b
  - km+1 mod k^b
  - …
  - km+(k-1) mod k^b



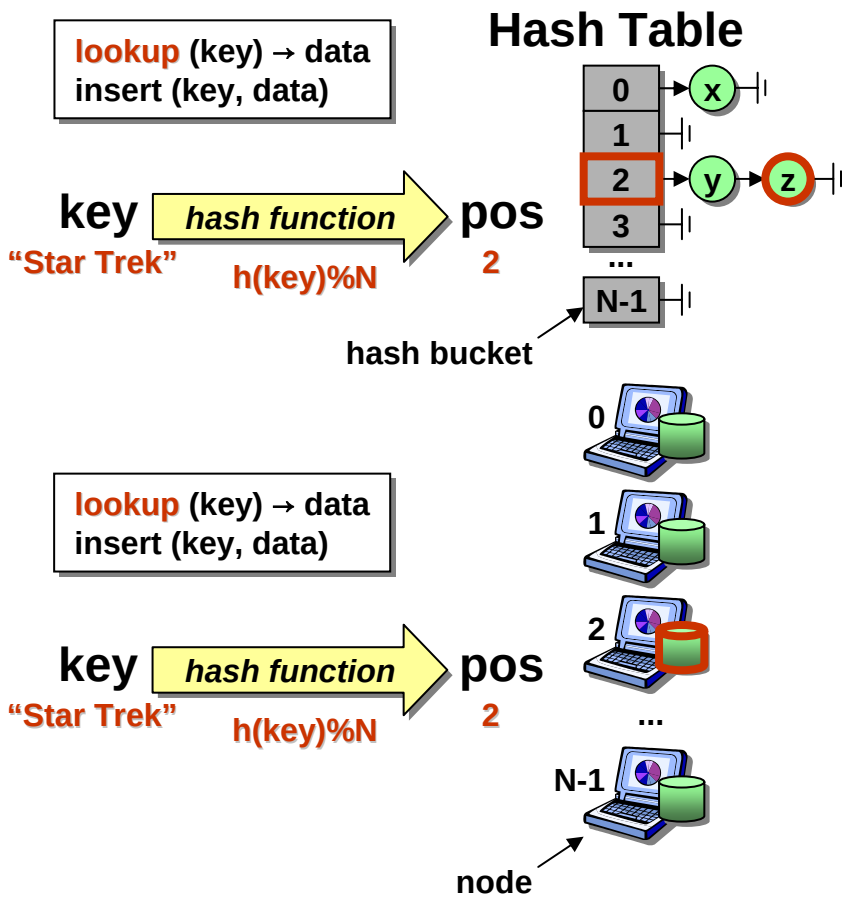
Constructing overlay applications and services

GOALS:

- Decentralization
- Scalability
- Anonimity
- Self-organization (autonomic communications)
- Ad-hoc connectivity
- Shared resource aggregation
- Performance
- Security
- Fault Tolerance

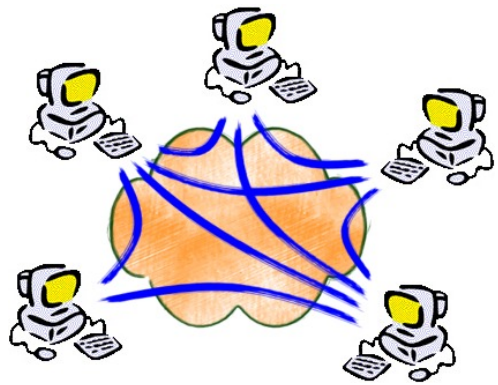
Distributed Hash Tables

- A hash table links data with keys
  - The key is hashed to find its associated *bucket* in the hash table
  - Each *bucket* expects to have #elems/#buckets elements
- In a **Distributed Hash Table (DHT)**, physical nodes are the hash *buckets*
  - The key is hashed to find its responsible node
  - Data and load are balanced among nodes



DHT applications

- File sharing [CFS, OceanStore, PAST, ...]
- Web cache [Squirrel, Coral]
- Censor-resistant stores [Eternity, FreeNet, ...]
- Application-layer multicast [Narada, ...]
- Event notification [Scribe]
- Streaming (SplitStream)
- Naming systems [ChordDNS, INS, ...]
- Query and indexing [Kademlia, ...]
- Communication primitives [I3, ...]
- Backup store [HiveNet]
- Web archive [Herodotus]
- Suggestions ?



Structured Overlays

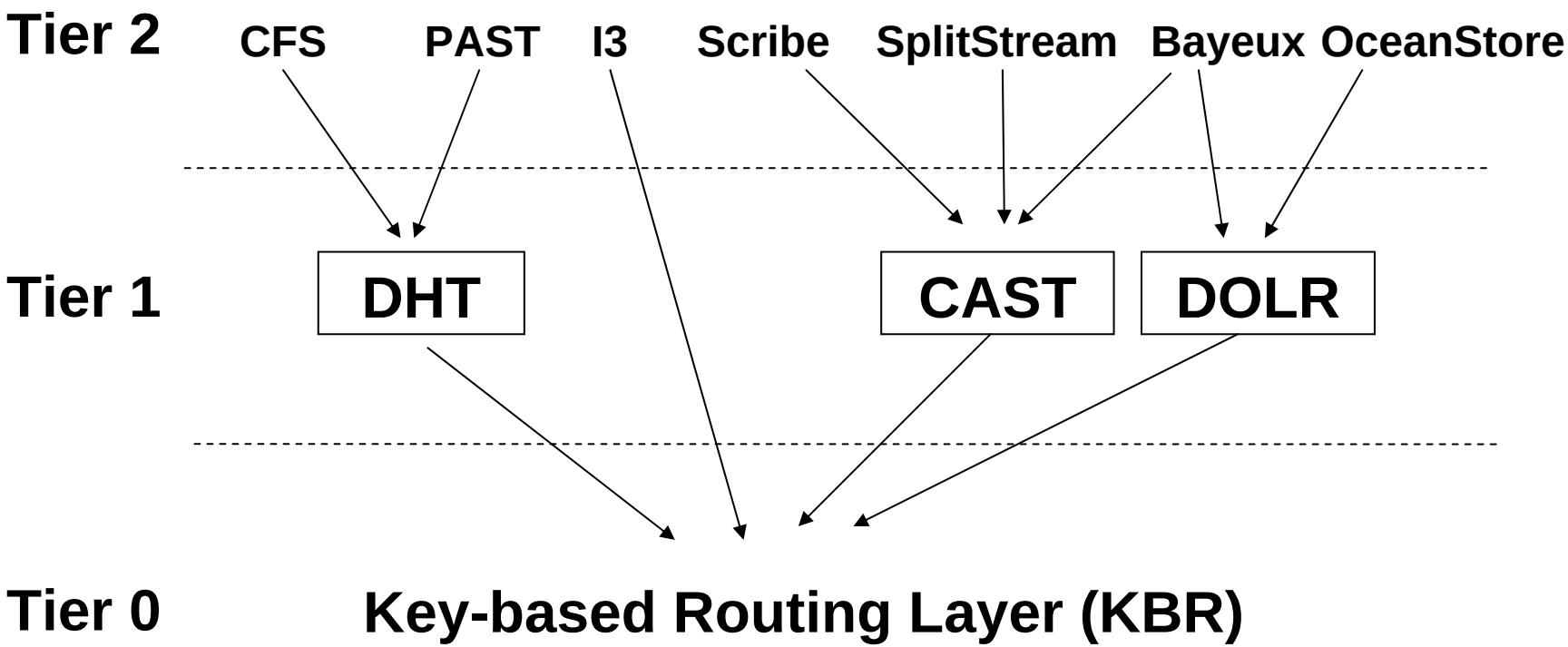
- They are:
  - Scalable, self-organizing overlay networks
  - Provide routing to location-independent names
  - Examples: CAN, Chord, Pastry, Tapestry, ...
- Basic operation:
  - Large sparse namespace N

    (integers: $0–2^{128}$ or $0–2^{160}$)
  - Nodes in overlay network have nodeIds € N
  - Given k € N, a deterministic function maps k to its root node (a live node in the network)
  - **route**(msg, k) delivers msg to root(k)

Towards a Common API

- Easily supportable by old and new protocols
- Enables application portability between protocols
- Enables common benchmarks
- Provides a framework for reusable components
- Achieves Interoperability between heterogeneous protocols

A Layered Approach: Common API



Functional Layers

- DHT (Distributed Hash Table)
  - **put**(key,data), value = **get**(key)
  - Hashtable layered across network
  - Handles replication; distributes replicas randomly
  - Routes queries towards replicas by name
- DOLR (Decentralized Object Location and Routing)
  - **publish**(objectId), **route**(msg, nodeId),

    r**outeObj**(msg, objectId, n)
  - Application controls replication and placement
  - Cache location pointers to replicas; queries quickly intersect pointers and redirect to nearby replica(s)

API (Routing)

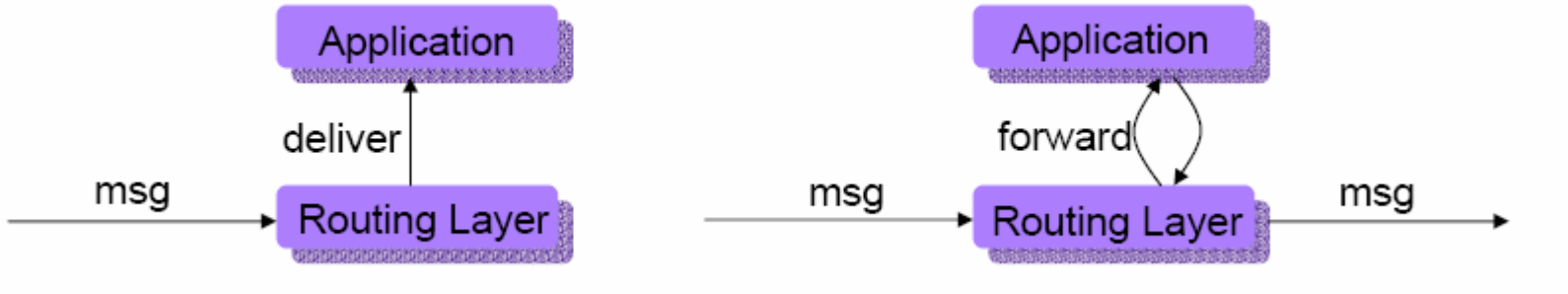
- Data types
  - Key, nodeId = 160 bit integer
  - Node = Address (IP + port #), nodeId
  - Msg: application-specific msg of arbitrary size

- Invoking routing functionality
  - **Route**(key, msg, [node])
    - route message to node currently responsible for key
    - Non-blocking, best effort – message may be lost or duplicated.
    - node: transport address of the node last associated with key (proposed first hop, optional)

API

- **nextHopSet = local_lookup(key, num, safe)**
  - Returns a set of at most *num* nodes from the local routing table that are possible next hops towards the *key*.
  - Safe: whether choice of nodes is randomly chosen
- **nodehandle[ ] = neighborSet(max_rank)**
  - Returns unordered set of nodes as neighbors of the current node.
  - Neighbor of rank *i* is responsible for keys on this node should all neighbors of rank < *i* fail
- **nodehandle[ ] = replicaSet(key, num)**
  - Returns ordered set of up to *num* nodes on which replicas of the object with key *key* can be stored.
  - Result is subset of neighborSet plus local node
- **boolean = range(node, rank, lkey, rkey)**
  - Returns whether current node would be responsible for the range specified by *lkey* and *rkey*, should the previous *rank-1* nodes fail.

API (Upcalls)



- **Deliver(key, msg)**
  - Delivers an incoming message to the application. One application per node.
- **Forward(key, msg, nextHopNode)**
  - Synchronous upcall invoked at each node along route
  - On return, will forward *msg* to *nextHopNode*
  - App may modify *key*, *msg*, *nextHopNode*, or terminate by setting *nextHopNode* to NULL.
- **Update(node, boolean joined)**
  - Upcall invoked to inform app of a change in the local node's neighborSet, either a new node joining or an old node leaving.

Common API ideas

- It is a paper, not a standard !
  - Towards a Common API for Structured Peer-to-Peer Overlays. IPTPS 2003.
  - Dabek, Zhao, Druschel, Stoica, kubiatowitcz
- It is academic !! There are not comercial implementations of the Common API
- But: it helps to provide common concepts for developing p2p applications over structured overlays

Common API = KBR

- You still have to implement the DHT, replication protocols, fault tolerance, ...
- Other approach is to construct directly applications on top of the DHT (put, get) interface. Example: OpenDHT has public XMLRPC APIs.

Ideas for applications:

- Fault tolerant replicated DHT
- Robust Service Discovery
- Query engine
- Propagation channel (CAST, BitTorrent)
- File Sharing tool

Open problems

- Range Queries over Structured Overlays
  - GoogleBase or P2P DataBase ?
- Resource handling:
  - Incentives Vs tragedy of the commons
- Security
  - Redundant routing
  - Robust systems
  - Trust Systems
- Autonomic Communications
  - Self-healing autonomic systems
  - Self-protecting autonomic systems
  - Self-optimizing autonomic systems
  - Self-configuring autonomic systems

Open problems

- Pervasive Computing
- Mobile Ad-Hoc Networks (MANET)
- Grid Computing
- Large Databases
- Content Distribution Networks
- Event-Based Systems
- Decentralized Collaborative Systems