The Impact of DHT Routing Geometry on Resilience and Proximity

Presented by Karthik Lakshminarayanan
at P2P Systems class
(Slides liberally borrowed from Krishna's SIGCOMM talk)

Krishna Gummadi, Ramakrishna Gummadi,
Sylvia Ratnasamy,
Steve Gribble, Scott Shenker, Ion Stoica

Motivation

- New DHTs constantly proposed
  - CAN, Chord, Pastry, Tapestry, Viceroy, Kademlia, Skipnet, Symphony, Koorde, Apocrypha, Land, Bamboo, ORDI …
- Each is extensively analyzed but in isolation
- Each DHT has many algorithmic details making it difficult to compare

Goals:

a) *Separate fundamental design choices from algorithmic details*

b) *Understand their effect on reliability and efficiency*

Approach:Component-based analysis

- Break DHT design into independent components
- Analyze impact of each component choice separately
  - compare with black-box analysis:
    - benchmark each DHT implementation
    - rankings of existing DHTs vs. hints on better designs

Different components of analysis

- Two types of components
  - *Routing-level* : neighbor & route selection
  - *System-level* : caching, replication, querying policy etc.
- Separating "routing" and "system" level issues
  - Good to understand them in isolation
  - Cons of this approach?

Outline

- DHT Design
- Compare DHT Routing Geometries
- Geometry's impact on Resilience
- Geometry's impact on Proximity

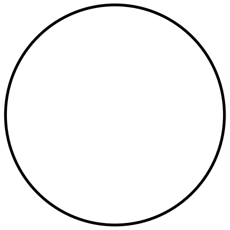
Three aspects of a DHT design

1) Geometry: a graph structure that inspires a DHT design
   - Tree, Hypercube, Ring, Butterfly, Debruijn

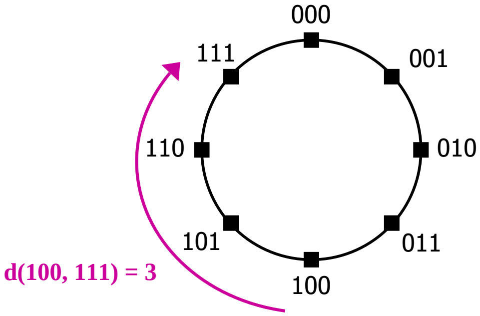
2) Distance function: captures a geometric structure
   - d(id1, id2) for any two node identifiers

3) Algorithm: rules for selecting neighbors and routes using the distance function

Chord DHT has Ring **Geometry**

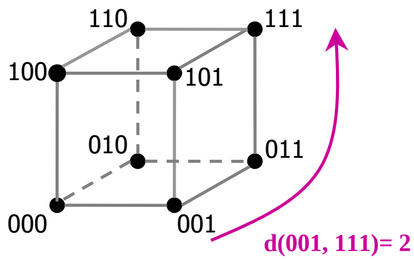


Chord **Distance** function captures Ring



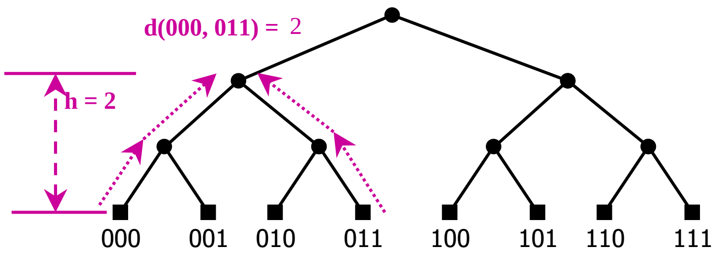
- Nodes are points on a clock-wise Ring
- **d(id1, id2) = length of clock-wise arc between ids = (id2 – id1) mod N**

CAN => Hypercube Geometry



- **d(id1, id2) = #differing bits between id1 and id2**
- Nodes are the corners of a hypercube

PRR => Tree



- Nodes are leaves in a binary tree
- **d(id1, id2) = height of smallest sub-tree with ids = logN – length of prefix_match(id1, id2)**

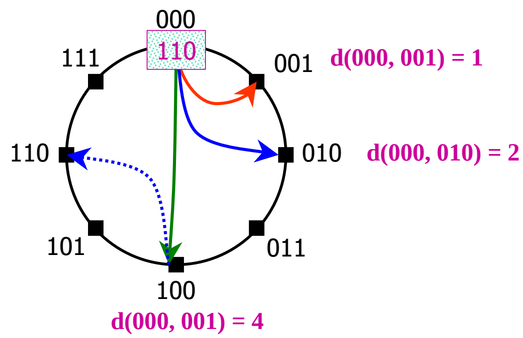
Geometry Vs Algorithm

- *Algorithm* : exact rules for selecting neighbors, routes
  - Chord, CAN, PRR, Tapestry, Pastry etc.
  - Inspired by geometric structures like Ring, Hyper-cube, Tree
- *Geometry* : an algorithm's underlying structure
  - *Distance* function is the formal representation of Geometry
  - Chord, Symphony => Ring
  - Many algorithms can have same geometry

Is the notion of Geometry clear?

- Notion of geometry is vague (as the authors admit)
- It is really a distance function on an ID-space
  - Hypercube is a special case of XOR!
- Possible formal definitions?

Chord Neighbor and Route selection Algorithms



- Neighbor selection: $\mathbf{i}^{th}$ neighbor at $\mathbf{2}^{i}$ distance
- Route selection: pick neighbor closest to destination

Geometry => Flexibility => Performance

- Geometry captures *flexibility* in selecting algorithms
- Flexibility is important for routing performance
  - Flexibility in selecting routes leads to shorter, reliable paths
  - Flexibility in selecting neighbors leads to shorter paths

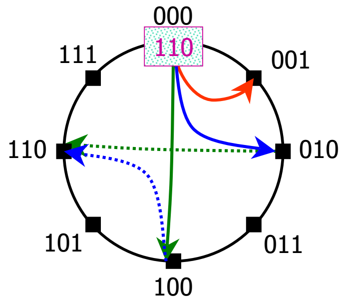
Outline

- Routing Geometry
- Comparing DHT Geometries
- Geometry’s impact on Resilience
- Geometry’s impact on Proximity

Geometries considered

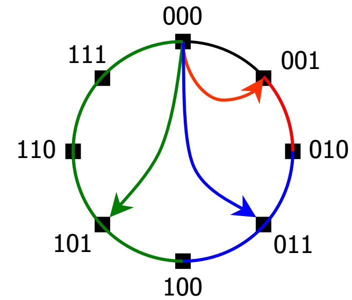
Geometry	**Algorithm**
Ring	Chord, Symphony
Hypercube	CAN
Tree	PRR
Hybrid = Tree + Ring	Tapestry, Pastry
XOR d(id1, id2) = id1 XOR id2	Kademlia

Route selection flexibility allowed by Ring Geometry



- Chord algorithm picks neighbor closest to destination
- A different algorithm picks the best of alternate paths

Neighbor selection flexibility allowed by Ring Geometry

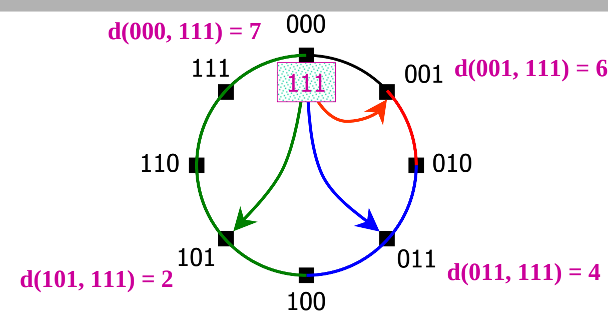


- Chord algorithm picks **i^th^** neighbor at **2^i^** distance
- A different algorithm picks $\mathbf{i^{th}}$ neighbor from $\mathbf{[2^i, 2^{i+1})}$

Metrics for flexibility

- **FNS**: Flexibility in Neighbor Selection
  = number of node choices for a neighbor
- **FRS**: Flexibility in Route Selection
  = avg. number of next-hop choices for all destinations
- Constraints for neighbors and routes
  - select neighbors to have paths of **O(logN)**
  - select routes so that each hop is closer to destination

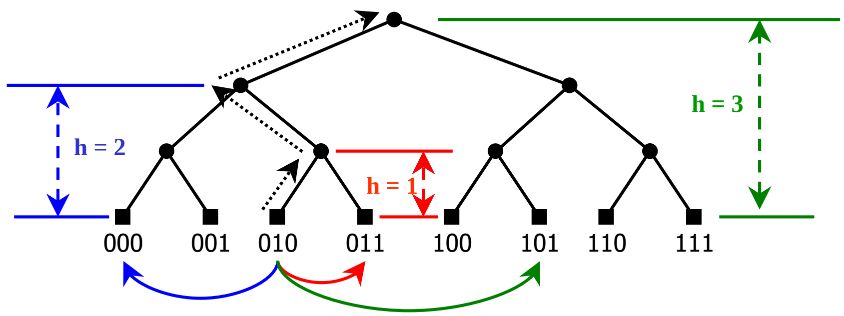
Flexibility of Ring



- **logN** neighbors at exponential distances
- **FNS = $\mathbf{2^{i-1}}$** for $\mathbf{i^{th}}$ neighbor
- Route along the circle in clock-wise direction

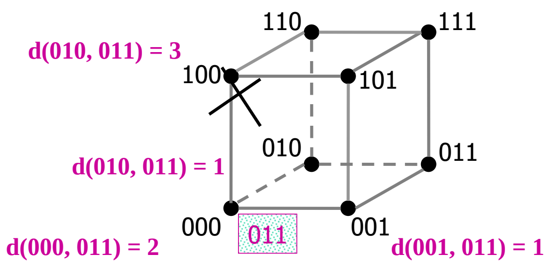
$$\mathbf{FRS} = \sum_{\forall J} \mathbf{\log(\, d(000, J)\,)/N \approx \log N}$$

Flexibility for Tree



- **logN** neighbors in sub-trees of varying heights
- **FNS = 2^{i-1}** for **i^{th}** neighbor of a node
- Route to a smaller sub-tree with destination;**FRS=1**

Flexibility for Hypercube



- Routing to next hop fixes one bit
- **FRS =Avg. (#bits destination differs in)=logN/2**
- **logN** neighbors differing in exactly one bit; **FNS=1**

Summary of flexibility analysis

Flexibility	Ordering of Geometries
Neighbors (FNS)	**Hypercube << Tree, XOR, Ring, Hybrid** (1) (2^{i-1})
Routes (FRS)	**Tree << XOR, Hybrid < Hypercube < Ring** (1) (logN/2) (logN/2) (logN)

How relevant is flexibility for DHT routing performance?

Outline

- Routing Geometry
- Comparing DHT Geometries
- Geometry's impact on Resilience
- Geometry's impact on Proximity

Static Resilience

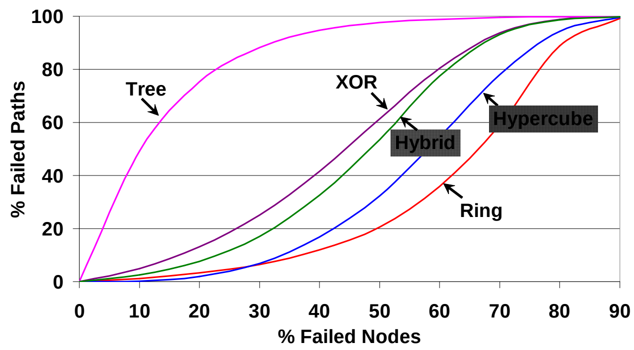
Two aspects of robust routing

- Dynamic Recovery : how quickly routing state is recovered after failures
- Static Resilience : how well the network routes before recovery finishes
  - captures how quickly recovery algorithms need to work
  - depends on FRS

Evaluation:

- Fail a fraction of nodes, without recovering any state
- Metric: **% Paths Failed**

Does flexibility affect Static Resilience?



Tree << XOR ≈ Hybrid < Hypercube < Ring

Static Resilience: Summary

- Tree << XOR ≈ Hybrid < Hypercube < Ring
  - What about trees with 2 neighbors?
- Addition of sequential neighbors helps resilience, but increases stretch
- Sequential neighbors offer more benefit, again at the cost of increased stretch

Flexibility in Route Selection matters for Static Resilience

Outline

- Routing Geometry
- Comparing flexibility of DHT Geometries
- Geometry's impact on Resilience
- Geometry's impact on Proximity
  - Overlay Path Latency
  - Local Convergence

Analysis of *Overlay Path Latency*

- Goal: Minimize end-to-end overlay path latency
- Both FNS and FRS can reduce latency
  - Tree has FNS, Hypercube has FRS, Ring & XOR have both

Evaluation:

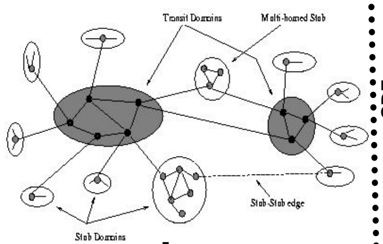
- Using Internet latency distributions

Problems with existing Network Models

- How to assign edge latencies to network topologies?
  - topology models: GT-ITM, Power-law, Mercator, Rocketfuel
  - no edge latency models, even for measured topologies
- Solution : A model using *only* latency distribution seen by a typical node

Simulations using latency distribution *only*

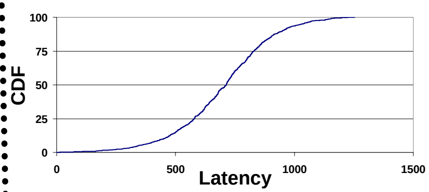
1) Topology, Edge Latencies



Simulate

Simulated Overlay Path Latency Distribution ≈

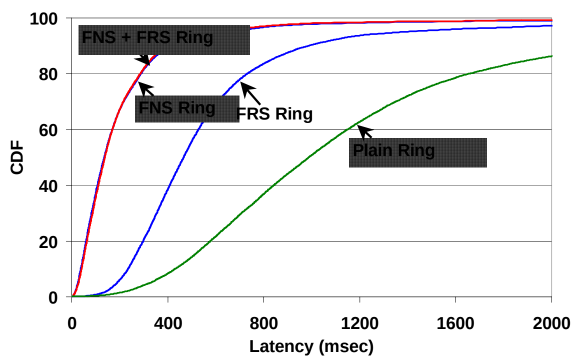
2) Latency Distribution



Compute

Computed Overlay Path Latency Distribution

Which is more useful: FNS or FRS?



Plain << FRS << FNS ≈ FNS+FRS

Neighbor Selection is much better than Route Selection

Proximity results: Summary

- Using neighbor selection is much better than using route selection flexibility
- Performance of FNS/FRS is independent of geometry beyond its support for neighbor selection
- In absolute terms, proximity techniques perform well (stretch of <2)

Local convergence: Summary

- Flexibility in neighbor selection helps much better than that in route selection
- Relevance of FRS depends on whether FNS restricted to a k-random sample closely approximates ideal FNS

Limitations

- Notion of geometry is vague (as the authors admit) – it is really a distance function on an ID-space
  - Hypercube is a special case of XOR!
- Not considered other factors that might matter
  - algorithmic details, symmetry in routing table entries
- Metrics under consideration can bias results – eg. In ring, do not distinguish between OPT and slightly sub-optimal paths