A Scalable Content-Addressable Network

Presented by : Peyman Navidi

Outline

- White Paper
- Introduction
- Design of CAN
- Routing in a CAN
- Joining in a CAN
- Maintenance
- Design Improvements
- Conclusion

White Paper

A Scalable Content-Addressable Network

Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker

Dept. of Electrical Eng. & Comp. Sci.
University of California, Berkeley, CA, USA

SIGCOMM, August 27-31, 2001, San Diego, California, USA.

Abstract

Hash tables – which map "keys" onto "values" – are an essential building block in modern software systems. We believe a similar functionality would be equally valuable to large distributed systems.

In this paper, we introduce the concept of a Content-Addressable Network (CAN) as a distributed infrastructure that provides hash table-like functionality on Internet-like scales. The CAN is scalable, fault-tolerant and completely self-organizing, and we demonstrate its scalability, robustness and low-latency properties through simulation.

Introduction

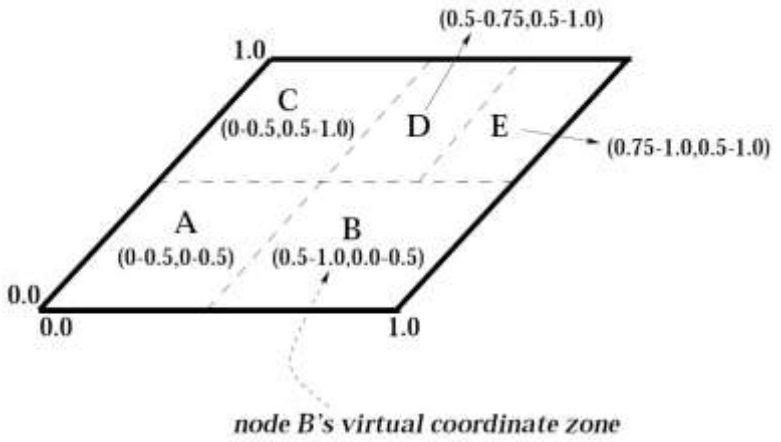
- Content-Addressable Networks (CAN) are presented as :
  - A scalable
  - Fault-tolerant
  - Completely self-organizing
  - Peer-to-Peer overlay network
- Indexing is accomplished with :
  - Distributed Hash Table
  - Mapping keys to values
- A network that provides a :
  - Routing overlay on top of a physical network
  - To optimise publishing and searching for data

Compare

- Napster
  - Locating a file is centralized.
- Gnutella
  - Floods the request for a file, not scalable
- CAN provides a solution:
  - Scalable - Nodes maintain small amount of control state
  - Distributed - Hash table is stored in all Peers, so it is

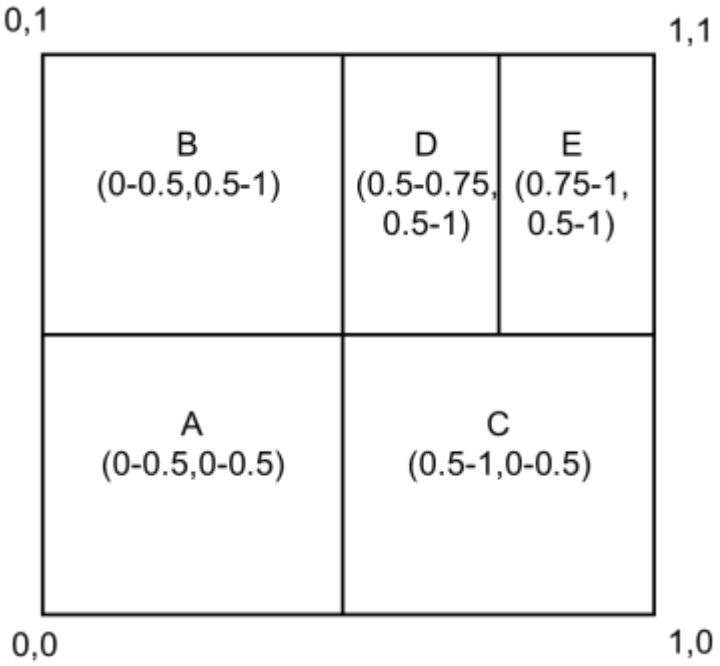
Design of CAN

- Multi-dimensional coordinate space with d dimensions (d-torus)
  - Coordinates are purely logical
- Each node owns a zone in the space
- Zone is a section of the hash table
- So each node :
  - Stores a chunk of hash table entry
  - Details of adjacent zones



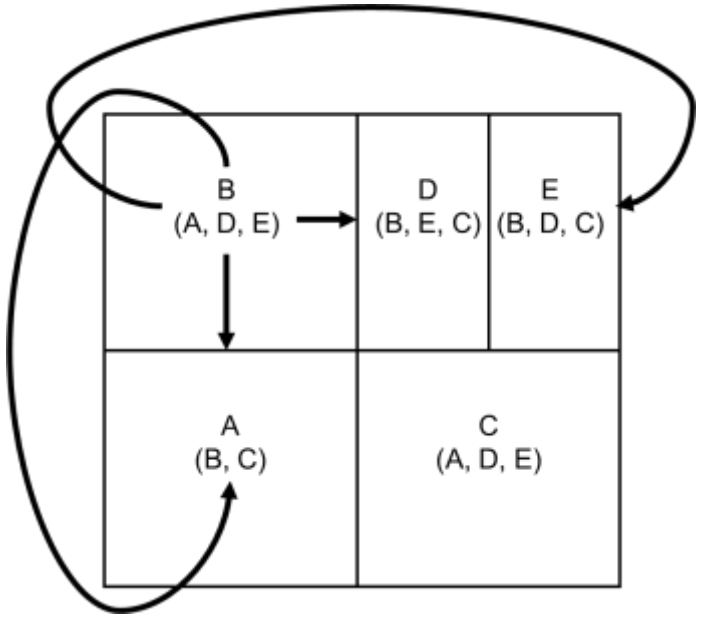
Coordinate Space

- Each node randomly picks a coordinate
- Coordinate space is dynamically partitioned
- Each node owns its individual zone



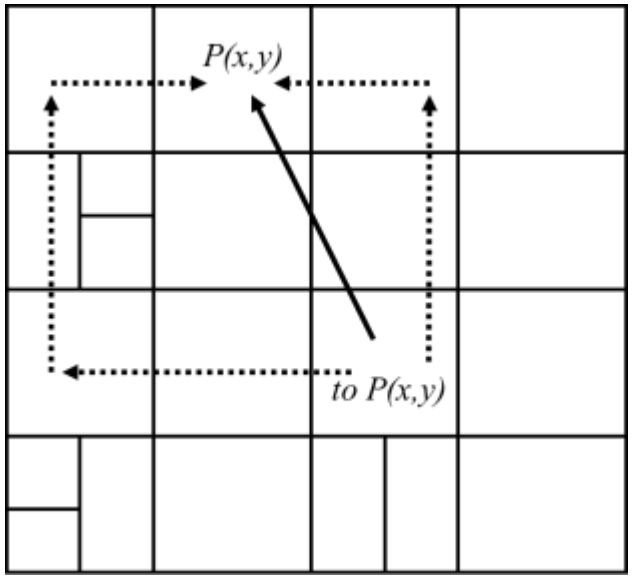
Neighbours

- A node is considered a neighbor
  - If its zone overlaps along d-1 dimensions
  - Abuts along one dimension
- A node maintains info about its neighbours
  - A contact address (IP)
  - Its zone coordinates
- An evenly divided space means each node has **2d** neighbours



Routing in a CAN

- Routing happens by following
  - A straight line path through the cartesian space
  - From source to destination coordinates
- A message with destination point P is :
  - Routed to the neighbour whose zone coordinates are closest to P
- There are multiple paths available at any point



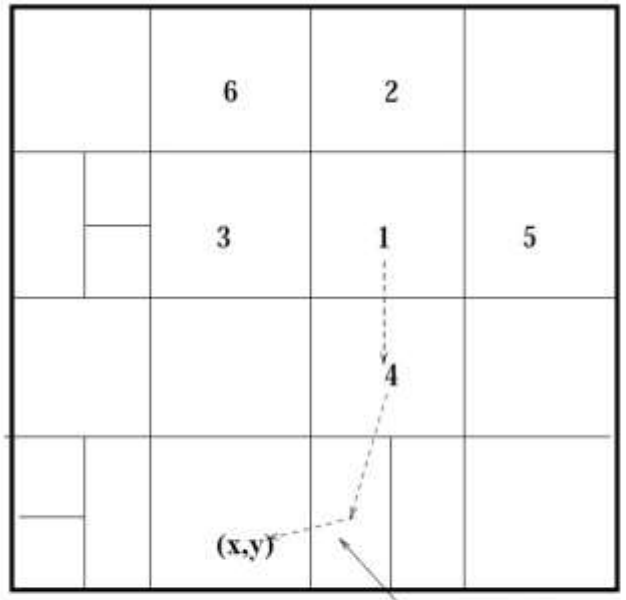
Routing in a CAN

- Greedy algorithm

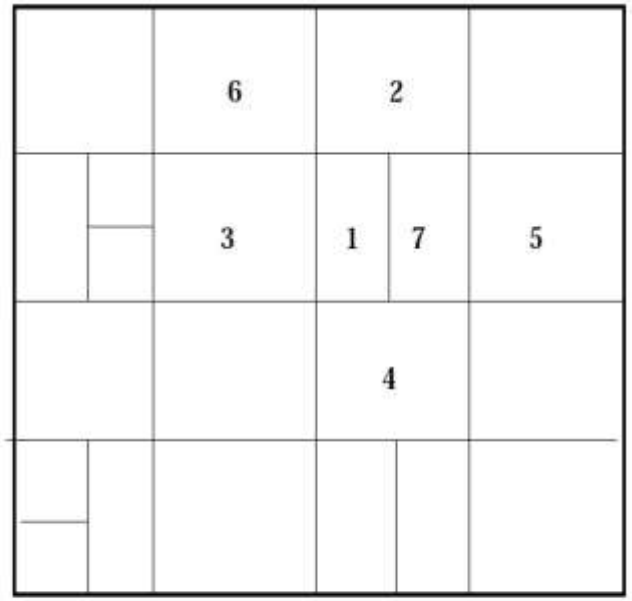
  if P is within the Zone of current node
  return (K, V)
  else
  forward the query to the neighbor with coordinates closest to P

- Draw a straight line from point in local zone to P
- Follow straight line via neighbors
- Nodes are self-organizing, making decisions dynamically
- Average path length = $^{d}/_{4}\, n^{1/d}$ hops
  - n : number of zones

Example



1's coordinate neighbor set = {2,3,4,5}
7's coordinate neighbor set = { }



1's coordinate neighbor set = {2,3,4,7}
7's coordinate neighbor set = {1,2,4,5}

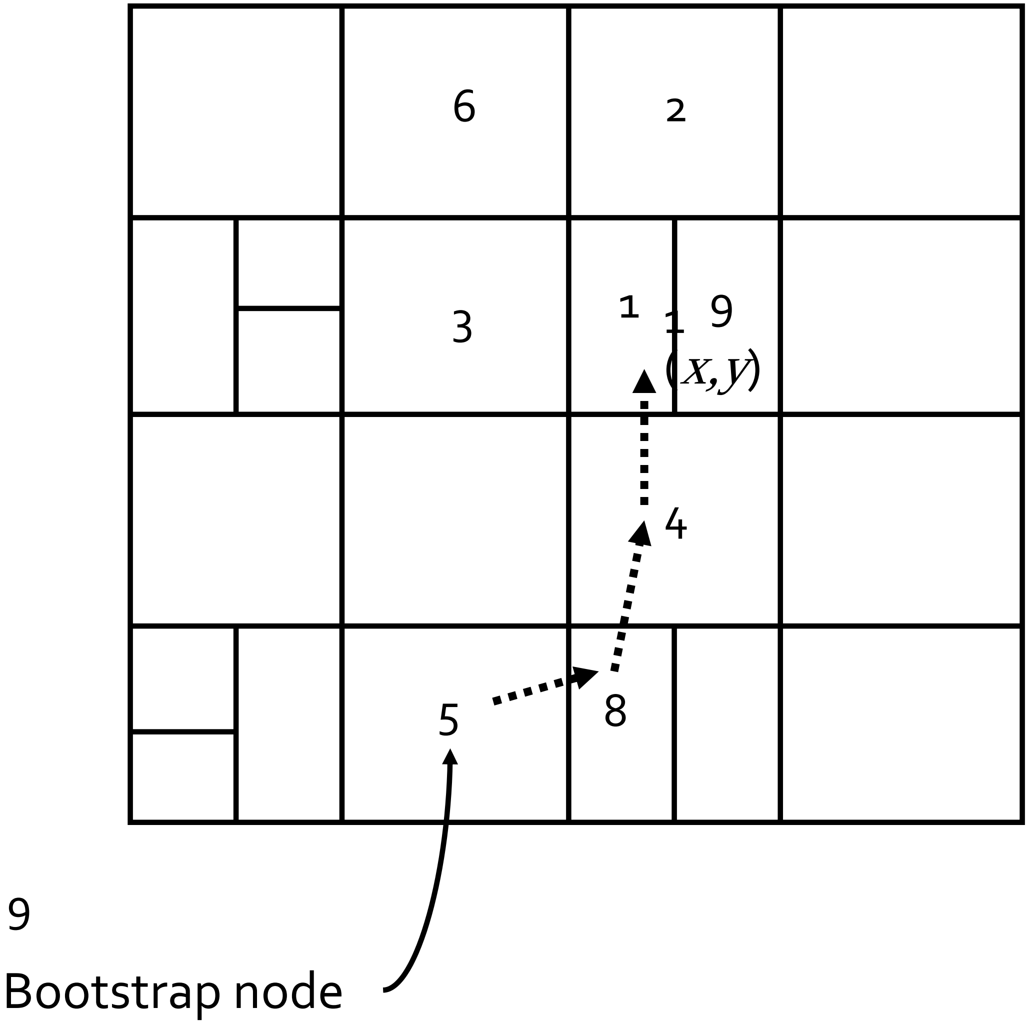
Joining in a CAN

- To join the CAN, A node needs a bootstrap node
  - i.e. the address and coords of a node already in the system
- It randomly chooses a point in the coordinate system
- The message is routed to the node whose coordinate space contains the point
- That node splits its space in half, keeping one half and handing over the other to the new node

Bootstrap

- From DNS **domain name**, one or more bootstrap nodes is determined
- A bootstrap node maintains :
  - A **partial list** of CAN nodes it believes are currently in the system
- TO join a CAN , a new node **looks up the CAN domain** name in DNS to retrieve a **bootstrap nodes IP** address
- This bootstrap node then supplies the IP address of **several randomly chosen nodes** currently in system

Example



in the space.

Let's say it's 5.

The bootstrap node initiates the Routing.

Maintenance

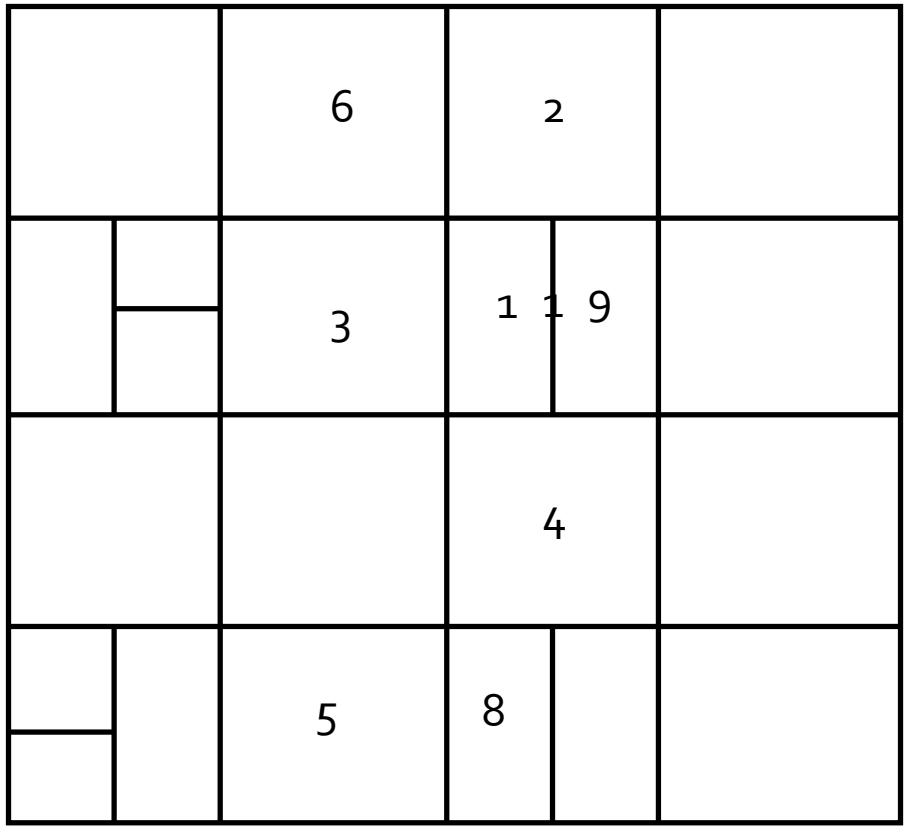
- Departure of a Node
- Single Node Failure

Departure of a Node

- The node that departs hands over the details to the one of its neighbor
- If the zone of one of the neighbors **can be merged** with the departing node's zone to produce a valid single zone, then this is done
- If not, then the zone is handed to the neighbor whose **current zone is smallest**, and that node will then temporarily handle both zones

Example

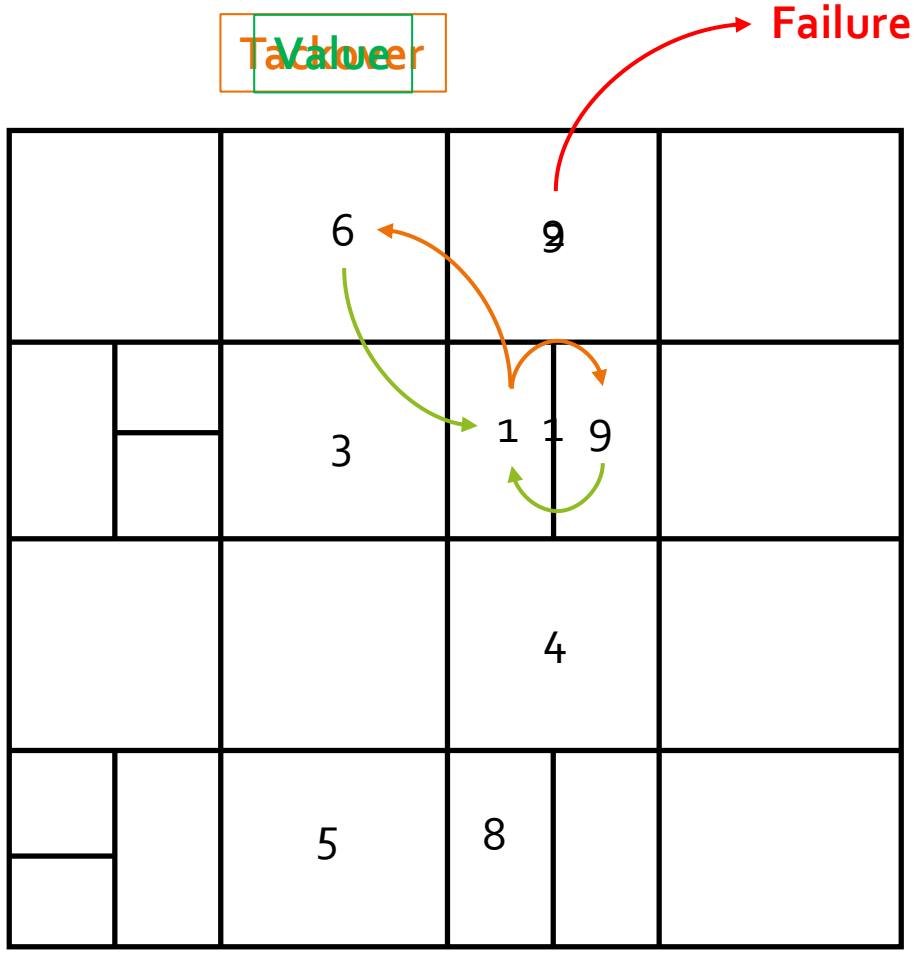
When node F fails, E will be merged with F



Single Node Failure

- Each node sends **periodic update** messages to each of its neighbors
- Crashed nodes are detected by neighbors by **a lack of periodic update** messages
- Neighbor nodes start **takeover timer**
- Send a takeover message to all of failed node's neighbors
- Neighboring nodes agree on node with smallest volume
- Smallest node takes over crashed node's zone

Example

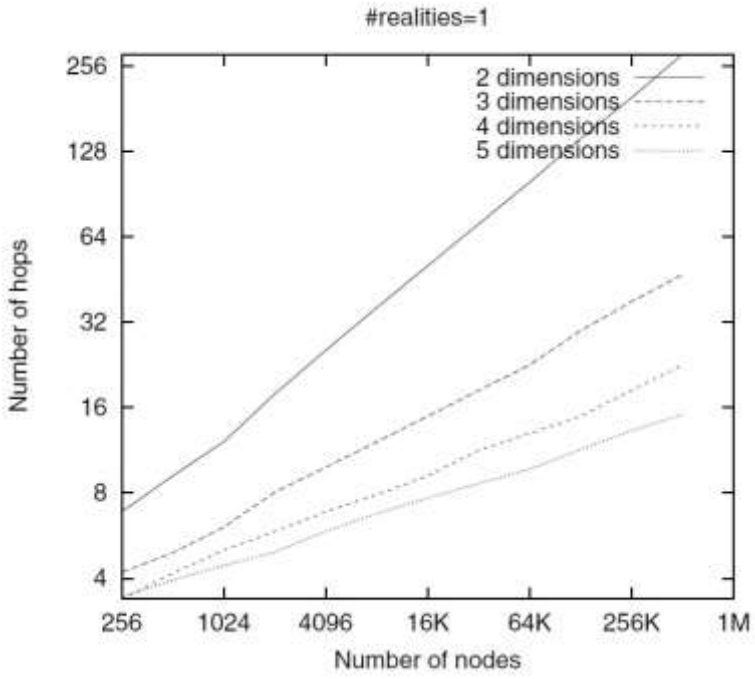


Design Improvements

- Multiple Dimensions
- Multiple Realities
- Multiple Hash functions

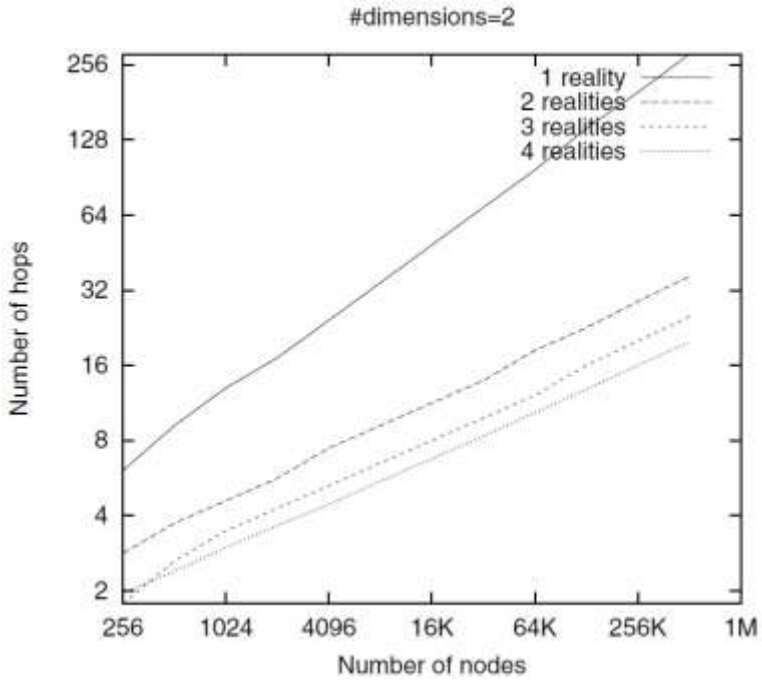
Multiple Dimensions

- Increase number of dimensions
  - Reduce average **path length**
  - Reduce **path latency**
  - Increases routing tablesize due to greater number of neighbors
  - Improves routing fault-tolerance
  - Simulated path lengths follow $O(dn^{1/d})$



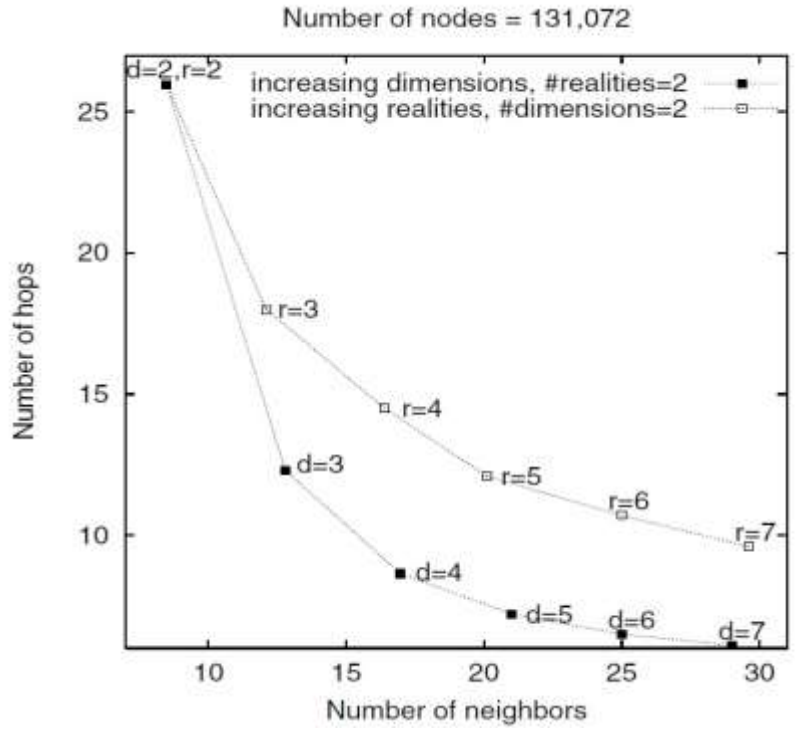
Multiple Realities

- Increase number of Realities
  - Nodes can maintain multiple independent coordinate spaces
  - For a CAN with r realities: a single node is assigned *r* zones and holds *r* independent neighbor sets
  - Contents of the hash table are replicated for each reality



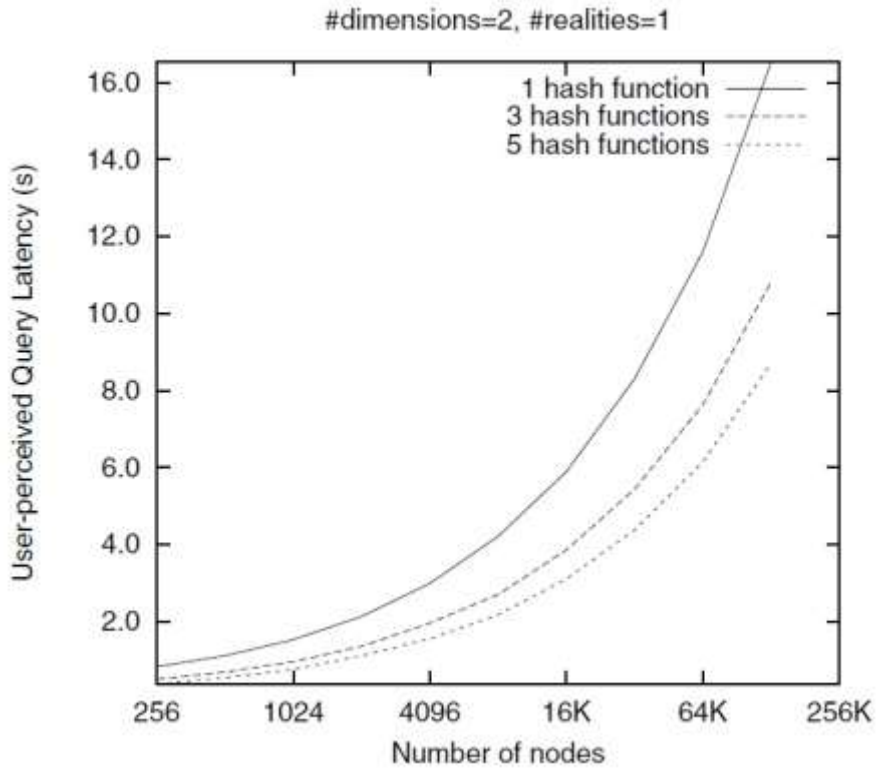
Dimensions vs. Realities

- Increasing the number of dimensions and/or realities **decreases path length** and increases per-node state
- More dimensions has greater effect on path length
- More realities provides stronger **fault-tolerance** and increased **data availability**



Multiple Hash Functions

- Increases data availability
- Reduces query latency
- (K, V) only unavailable when all nodes crash
- Parallel querying of k nodes with k hash functions can reduce lookup latency



Summary of the Complexity

P2P system	Overlay structure	Lookup protocol	Network parameter	Routing table size	Routing complexity	Join/leave overhead
Chord	1-dimensional, circular-ID space	Matching key and NodeID	n = number of nodes in the network	$O(\log(n))$	$O(\log(n))$	$O((\log(n))^2)$
Pastry	Plaxton style mesh structure	Matching key and prefix in NodeID	n = number of nodes in the network, b = base of the identifier	$O(\log_b(n))$	$O(b \log_b(n)+b)$	$O(\log(n))$
CAN	d-dimensional ID space	key, value pairs map to a point P in the d-dimen-sional space	n = number of nodes in the network, d = number of dimensions	$O(2\ d)$	$O(d\ n^{1/d})$	$O(2\ d)$
Tapestry	Plaxton style mesh structure	Matching suffix in NodeID	n = number of nodes in the network, b = base of the identifier	$O(\log_b(n))$	$O(b \log_b(n)+b)$	$O(\log(n))$

Conclusion

- Addresses two key problems in the design of Content-Addressable Networks:
  - Scalable routing
  - Indexing
- Simulation results validate the scalability of our overall design
  - for a CAN with over 260,000 nodes, we can route with a latency that is less than twice the IP path latency
- Future works
  - Secure CAN
  - Key word searching