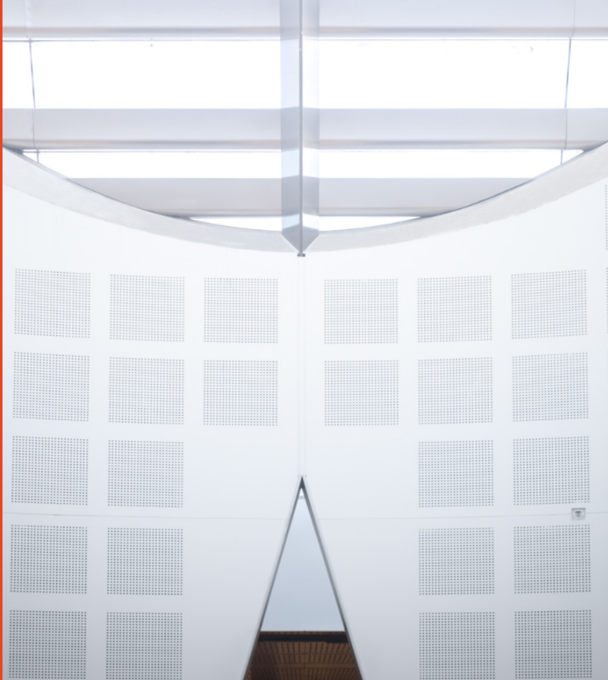
COMP5313/COMP4313 - Large Scale Networks

Week 11: Introduction to Peer-to-Peer Systems

Lijun Chang

May 15, 2025

THE UNIVERSITY OF SYDNEY



Introduction

- Last week we studied a model for constructing a network by using randomness, such that it is small-world and is efficiently searchable.
  - Besides regular links, we had to create random links to nodes with probability that is inversely proportional to the number of closer nodes
- We will now use this observation to create peer-to-peer file sharing systems, such that files can be easily found in peer-to-peer systems
  - The network structure will be constructed in a deterministic way

Outline

Peer-to-Peer System

The Lookup Problem in P2P Systems

The Chord Distributed Hash Table

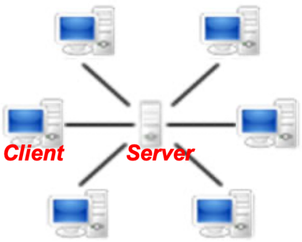
The CAN Distributed Hash Table

What is a Peer-to-Peer (P2P) system?

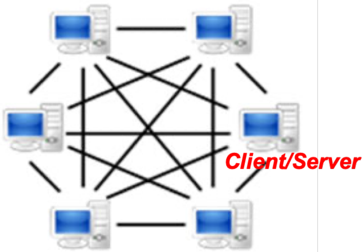
- A network of computers (e.g., at homes, schools, businesses) connected through a real network (e.g., cable modem, DSL links, WiFi).
  - Each computer usually is referred to as a node or a peer.
- A large-scale network
  - Thousands of simultaneously active participants
  - Million of machines over a week-long period
    - 3 millions machines in Gnutella as of January 2007
- A decentralized system
  - It does not need any centralized computer to work
  - Every computer acts both as a client and as a server

P2P V.S. Client-Server

- In the traditional client-server model, the server supplies while the client only consumes.
- In P2P networks, computers are both suppliers and consumers.



A client-server-based network



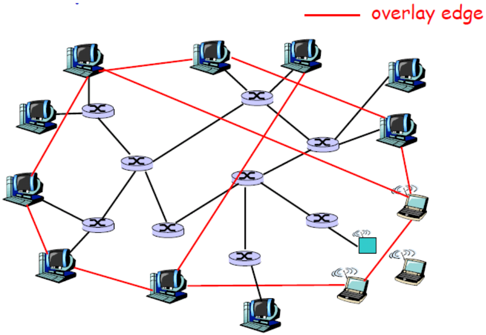
A P2P network

Benefits of P2P Systems

- The barrier to starting and growing such systems is low
  - No special administrative/financial arrangements
- It aggregates and makes use of the tremendous computation and storage resources on computers across the Internet
  - Otherwise left idle on the Internet
- It is robust to faults or intentional attacks
  - Good for long-term storage or lengthy computation

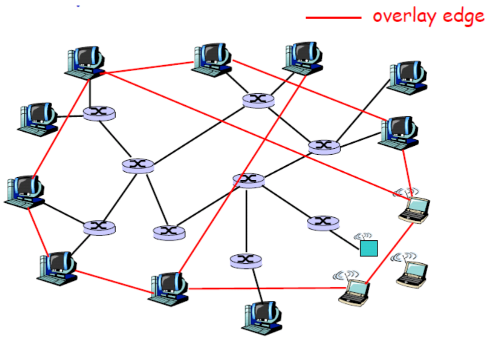
Overlay Network of P2P Systems

- A P2P system organizes computers/nodes into a logical overlay network
  - At the application layer, on top of native or physical (TCP/IP) network
  - The nodes in the overlay network is a subset of the nodes in the physical network



Overlay Network of P2P Systems

- Data is still exchanged over the underlying physical network, but at the application layer nodes directly communicate with each other via the logical overlay links
  - Each overlay link corresponds to a path through the underlying physical network



Overlay Network of P2P Systems

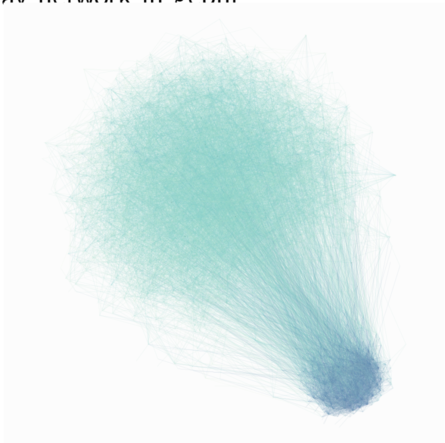
- Overlay networks make the P2P system independent from the physical network topology.
- Overlay networks are used for indexing and peer discovery
  - Each node can directly send messages only to its connected neighbors in the overlay network

Classification of P2P Systems

- Based on how the nodes are linked to each other within the overlay network and how contents (e.g., files) are indexed and located, we can classify P2P systems as unstructured or structured.
- Unstructured P2P systems
  - Do not impose any particular structure on the overlay network by design
  - Nodes randomly form connections to each other
  - Gnutella, Kazaa
- Structured P2P systems
  - The overlay network is organized into a specific topology
  - There is a protocol ensuring that any node can efficiently search the network for a content, even if the content is extremely rare.
  - BitTorrent

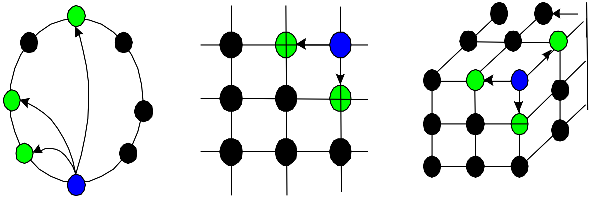
Example Unstructured P2P

- Example of a representation of Gnutella overlay network in gephi
- 6301 nodes, 20777 edges
- Dark blue: high in-degree and
  Light-blue: low in-degree
- Dataset:
  https://snap.stanford.edu/data/p2p-Gnutella08.html



Example Structured P2P

- Structured P2P connects the nodes smartly
  - The degrees of nodes are similar to each other



1-dimensional 2-dimensional 3-dimensional

Pros and Cons of Unstructured P2P

- Unstructured P2P is easy to build and allow for localized optimizations to different regions of the overlay
  - No structure globally imposed upon them
- Unstructured P2P is highly robust in the face of high rates of "churn" (i.e., nodes joining and leaving the network)
- It is difficult to search for rare content in unstructured P2P

Pros and Cons of Structured P2P

- Structured P2P is less robust for a high rate of churn
  - In order to route traffic efficiently through the network, nodes in a structured overlay must maintain lists of neighbors that satisfy specific criteria
- It is easy to find any content in structured P2P

Outline

The Lookup Problem

- The lookup problem is to find a given data item in a large P2P system in a scalable manner
  - Without any centralized server
- Some publisher inserts an item X, say a file, in the system
  - At a later point in time, some consumer wants to retrieve X

Lookup in Napster

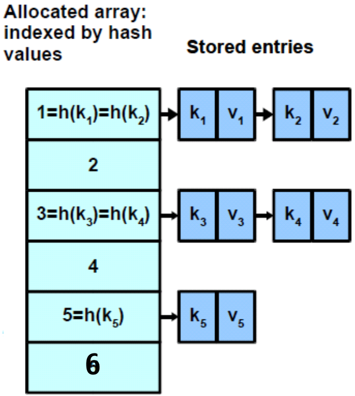
- Napster is hybrid between P2P and a centralized network (Lookup is centralized, but files are copied in P2P manner)
  - It maintains a central database that maps a file name to the locations of peers that store the file
  - Peers get from the server the locations of the peers that store the file
  - Peers download directly from other peer(s)
- Limited scalability
  - All peers query the central server for addresses
- Limited fault tolerance
  - Single point of failure, e.g., once lawyers got the central server to shut down, the network fell apart
- Napster existed from June 1999 to July 2001

Lookup in Gnutella

- Gnutella is a pure P2P system
- Gnutella used flooding to lookup a data item
  - The client sends a lookup request to each actively connected node
  - When a node receives such a request, it checks its local database
    - If it contains the data item, it responds with the item.
    - Otherwise, it forwards the request to all its actively connected nodes
  - The request package keeps propagating until it reaches a predetermined number of hops from the sender (e.g., maximum 7)
- This approach is not scalable
  - Large bandwidth consumed by broadcast messages
  - Many compute cycles consumed by the nodes that must handle the messages

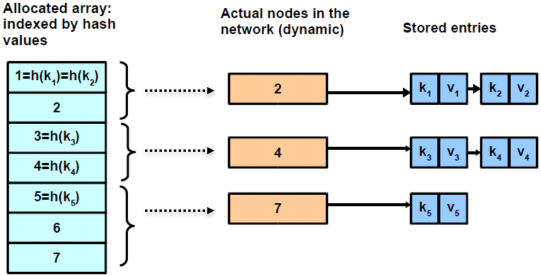
Lookup by Distributed Hash Table

- A hash table stores arbitrary keys and satellite data (value)
  - put(key, value)
  - value = get(key)
- This interface is an attractive foundation for a distributed lookup algorithm because it places very few constraints on the structure of keys or the data they name



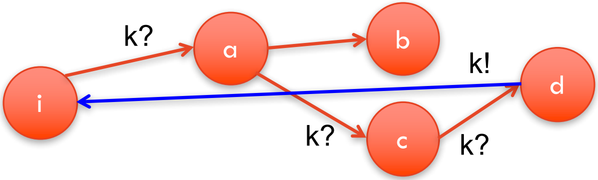
Lookup by Distributed Hash Table

- A Distributed Hash Table (DHT) partitions a hash table across a set of nodes: each node is responsible for a subset of the keys
  - It implements one central operation lookup(key) which yields the identity (e.g., IP address) of the node currently responsible for the given key
  - Someone, who wants to publish a file under a name, converts this name to a key using an ordinary hash function such as SHA-1 and calls lookup(key); the publisher would send the file to be stored at the resulting node

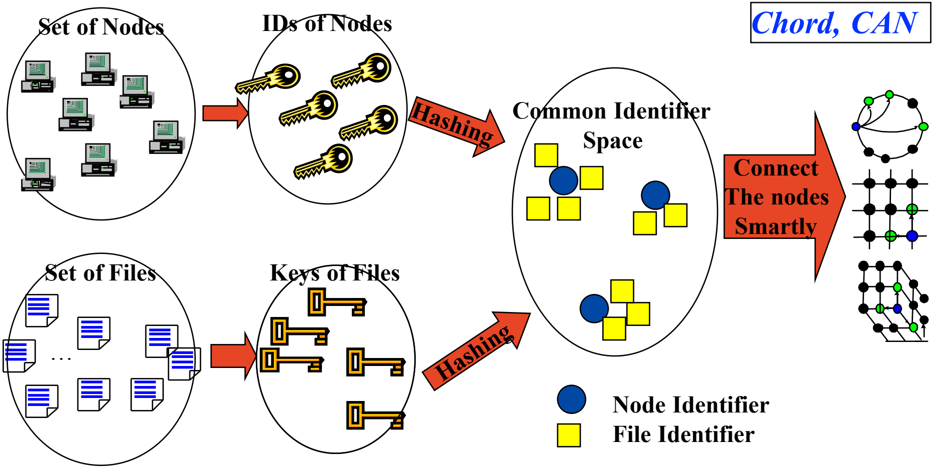


Lookup by Distributed Hash Table

- The goal of lookup(key) is to provide a distributed lookup service returning the host of a content
  - The content is hashed into a key, and the DHT returns the host of that key



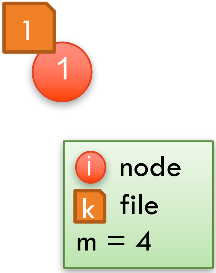
Overview of Distributed Hash Tables



Outline

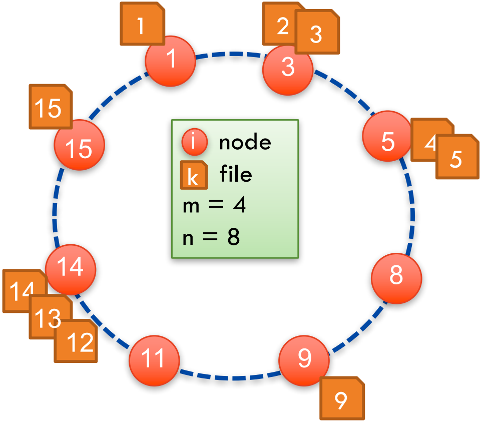
The Chord Distributed Hash Table

- **Files and nodes:**
  - Each node has a unique id, each file has a key, both coded in m bits
  - A file of key k is stored on the node whose id i equals $\min\{\text{node.id} : \text{node.id} \geq k\}$
    - This node is said to be responsible for the key k



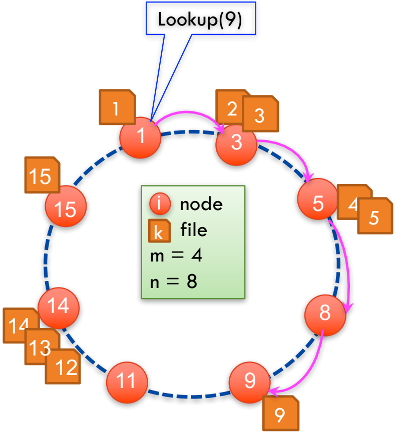
The Chord Distributed Hash Table

- **Files and nodes:**
  - Each node has a unique id, each file has a key, both coded in m bits
  - A file of key k is stored on the node whose id i equals $\min\{\text{node.id} : \text{node.id} \geq k\}$
    - This node is said to be responsible for the key k
- **Ring structure:**
  - n nodes are organized into a ring of $2^m \geq n$ slots with ascending id in the clockwise direction



The Chord Distributed Hash Table

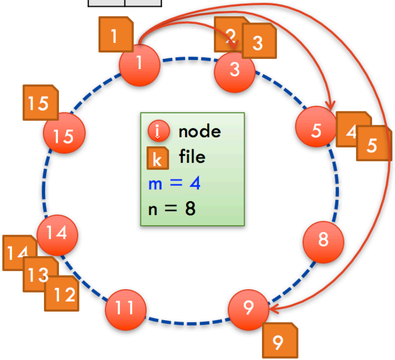
- **A naive lookup algorithm:**
  - Each node maintains its successor
  - To find the file with key k, query successor nodes until node.id $\geq k$
- **Running time:**
  - $O(n)$



The Chord Distributed Hash Table

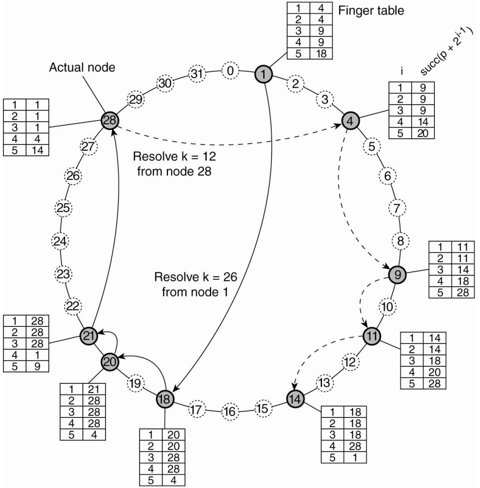
- To reduce lookup time, each node i maintains a finger table of m entries
  - j^{th} entry with address of the node responsible for key $(i + 2^{j-1}) \mod 2^m$

1	3
2	3
3	5
4	9



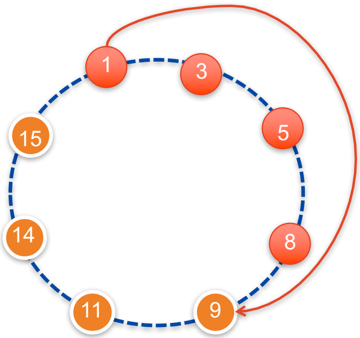
The Chord Distributed Hash Table

- **Routing with the finger table:** go as far as possible but never pass the target
  - If key between me and my succ then ask succ
  - Otherwise, ask the node in the finger table with the closest id that is no larger than the key
- Resolving key 26 from node 1
  - 1's finger table indicates to visit the closest lower id 18 to route to 26
  - 18's finger table indicates to visit 20
  - 20 tells to go to 21
  - 21 tells to go to its succ, 28.
- Resolving key 12 from node 28
  - 28's finger table indicates to visit the closest lower id 4 to route to 12
  - 4 tells to go to 9, 9 to 11.
  - 11 knows the succ 14 should have it



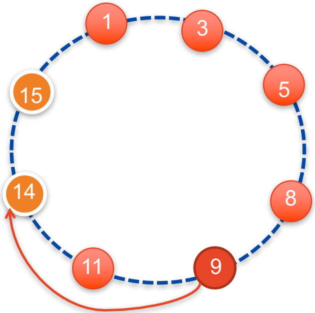
The Chord Distributed Hash Table

- **Time complexity:**
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1^{st} step: finds the half of the ring in which key is



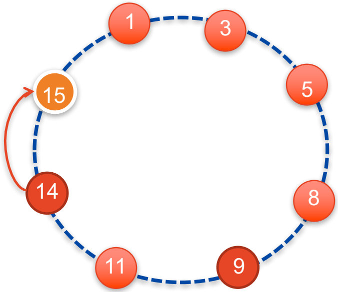
The Chord Distributed Hash Table

- **Time complexity:**
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1^{st} step: finds the half of the ring in which key is
    - 2^{nd} step: finds the quarter of the ring



The Chord Distributed Hash Table

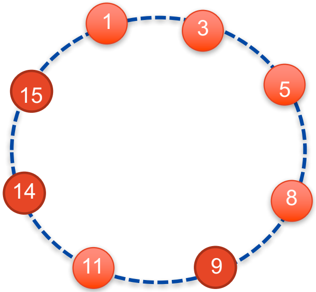
- **Time complexity:**
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1^{st} step: finds the half of the ring in which key is
    - 2^{nd} step: finds the quarter of the ring
    - 3^{rd} step: find the node that hosts the file



The Chord Distributed Hash Table

- **Time complexity:**
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1^{st} step: finds the half of the ring in which key is
    - 2^{nd} step: finds the quarter of the ring
    - 3^{rd} step: find the node that hosts the file
  - Generally speaking:
    - Start with n possible host nodes
    - After step i, only $\frac{n}{2^i}$ possible nodes remain

$\Longrightarrow$ $O(\log n)$

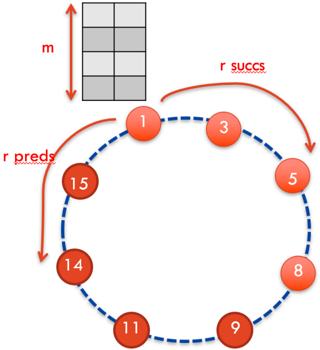


The Chord Distributed Hash Table

- **Space complexity:**
  - To cope with node fail and node leaving
    - Each node maintains the ids of its r successors and the ids of its r predecessors on the ring
  - Information maintained per node:
    - size of finger table, $m = O(\log n)$

      $+$
    - #predecessors and successors, $2r = O(1)$

$\Longrightarrow \quad O(\log n)$

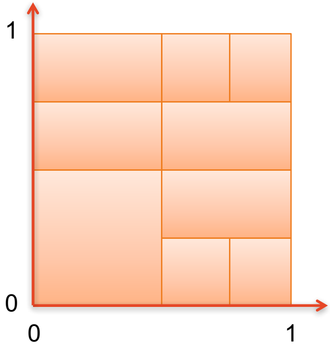


Outline

The CAN Distributed Hash Table

Content Addressable Network (CAN)

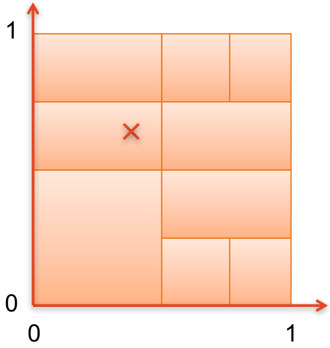
- **d-dimensional structure:**
  - Each node has a zone in the $(0, 1) \times (0, 1)$ 2-dimensional space $(d = 2)$
  - Each node is responsible for the files whose keys fall in its region



The CAN Distributed Hash Table

Content Addressable Network (CAN)

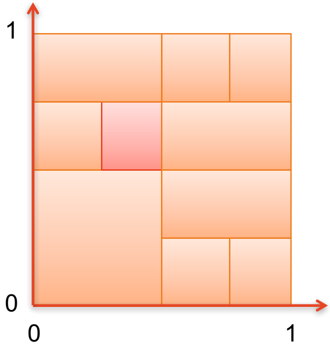
- **d-dimensional structure:**
  - Each node has a zone in the $(0,1) \times (0,1)$ 2-dimensional space $(d = 2)$
  - Each node is responsible for the files whose keys fall in its region
  - Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible for half of its zone



The CAN Distributed Hash Table

Content Addressable Network (CAN)

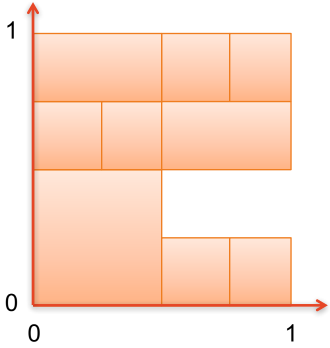
- **d-dimensional structure:**
  - Each node has a zone in the $(0,1) \times (0,1)$ 2-dimensional space $(d = 2)$
  - Each node is responsible for the files whose keys fall in its region
  - Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible for half of its zone



The CAN Distributed Hash Table

Content Addressable Network (CAN)

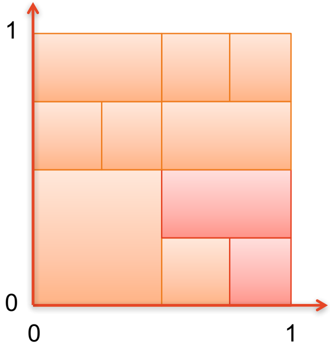
- **d-dimensional structure:**
  - Each node has a zone in the $(0,1) \times (0,1)$ 2-dimensional space $(d = 2)$
  - Each node is responsible for the files whose keys fall in its region
  - Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible for half of its zone
  - A neighboring node takes care of being responsible for the zone of the leaving node



The CAN Distributed Hash Table

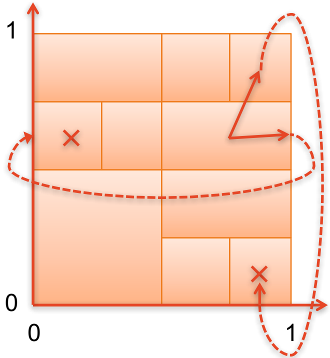
Content Addressable Network (CAN)

- **d-dimensional structure:**
  - Each node has a zone in the $(0,1) \times (0,1)$ 2-dimensional space $(d = 2)$
  - Each node is responsible for the files whose keys fall in its region
  - Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible for half of its zone
  - A neighboring node takes care of being responsible for the zone of the leaving node



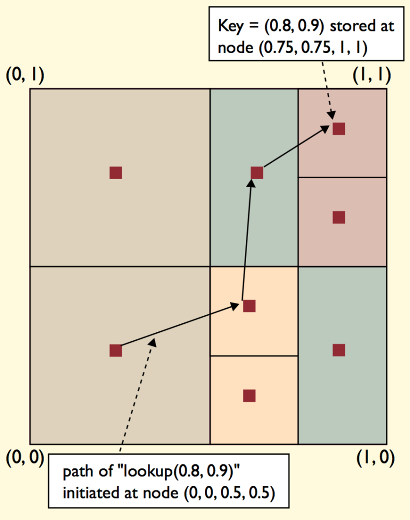
The CAN Distributed Hash Table

- **Routing: through abutting zones**
  - Each node maintains the addresses of its neighbours, that are nodes responsible for zones abutting his
  - The structure is actually a torus, so that topmost zones are considered abutting bottommost zones
  - To route towards the destination, each node chooses its closest neighbour to the destination in terms of euclidean distances



The CAN Distributed Hash Table

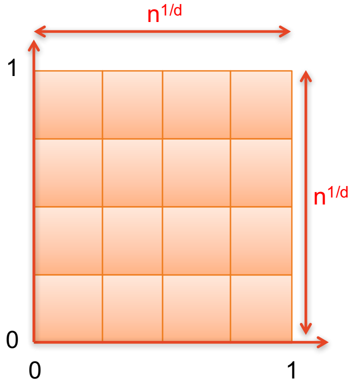
- **Routing: through abutting zones**
  - Each node maintains the addresses of its neighbours, that are nodes responsible for zones abutting his
  - The structure is actually a torus, so that topmost zones are considered abutting bottommost zones
  - To route towards the destination, each node chooses its closest neighbour to the destination in terms of euclidean distances



The CAN Distributed Hash Table

- **Complexity**
  - Route of lookup (for $d = 2$)
    - Assume id uniformly distributed in the range
    - Worst-case: has to route through half of the height of the zone + half of the width of the zone

    $\Longrightarrow$ $O(n^{\frac{1}{2}})$
  - Generally speaking $O(\frac{1}{2} \times d \times n^{\frac{1}{d}}) = O(d \times n^{\frac{1}{d}})$

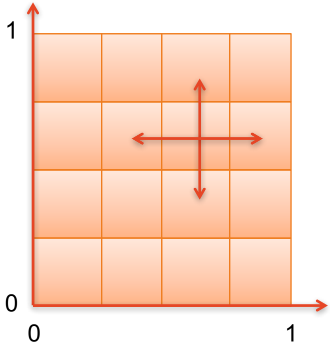


The CAN Distributed Hash Table

- **Complexity**
  - Route of lookup (for $d = 2$)
    - Assume id uniformly distributed in the range
    - Worst-case: has to route through half of the height of the zone + half of the width of the zone

  $\Longrightarrow$ $O(n^{\frac{1}{2}})$
  - Generally speaking $O(\frac{1}{2} \times d \times n^{\frac{1}{d}}) = O(d \times n^{\frac{1}{d}})$
  - Information maintained per node:
    - One in each direction (top, bottom, left, right)

  $\Longrightarrow$ $2d = O(d)$



Summary of Chord and CAN Distributed Hash Tables

General characteristics

- Fault tolerance:
  - Fully distributed, no central point of control
  - Each node knows only a small amount of neighbours:
    - In Chord: $O(m) = O(\log n)$ logarithmic in n
    - In CAN: $O(d)$ constant independent from n

$\Longrightarrow$ Small number of changes needed to adapt to churn (i.e., node leave/join)

- Scalable to a large number n of participants:
  - Each node knows the right nodes to route to:
    - In Chord: it takes $O(\log n)$ to lookup a key
    - In CAN: it takes $O(d \times n^{\frac{1}{d}})$ to lookup a key

$\Longrightarrow$ The routes are short compared to n

Conclusion

- Peer-to-peer networks are fully distributed networks where peers are connected through a network of wires (e.g., the Internet)
- Searching for a file stored in one of these peers is similar to decentralized searching within a social network:
  - Peers can collaborate by using their local information about the network
  - If they can properly exchange information, then they can search effectively
- The underlying idea of the Chord distributed hash table is consistent hashing, which has been used in many applicaitons. https://en.wikipedia.org/wiki/Consistent_hashing

Reading

- Reading for this week
  - Hari Balakrishnan, M. Frans Kaashoek, David R. Karger, Robert Tappan Morris, Ion Stoica: Looking up data in P2P systems. Commun. ACM 46(2): 43-48 (2003) https://dl.acm.org/doi/10.1145/606272.606299