Cristina Nita-Rotaru

7610: Distributed Systems

Lookup services. Chord. CAN. Pastry. Kademlia.

Required Reading

- I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, SIGCOMM 2001.
- A Scalable Content-Addressable Network S.a Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, SIGCOMM 2001
- A. Rowstron and P. Druschel. "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001
- Kademlia: A Peer-to-peer Information System Based on the XOR Metric. P. Maymounkov and D. Mazieres, IPTPS '02

1: Lookup services

Peer-to-Peer (P2P) Systems

- Applications that take advantage of resources (storage, cycles, content, human presence) available at the edges of the Internet.
- Characteristics:
    - System consists of clients connected through Internet and acting as peers
    - System is designed to work in the presence of variable connectivity
    - Nodes at the edges of the network have significant autonomy; no centralized control
    - Nodes are symmetric in function

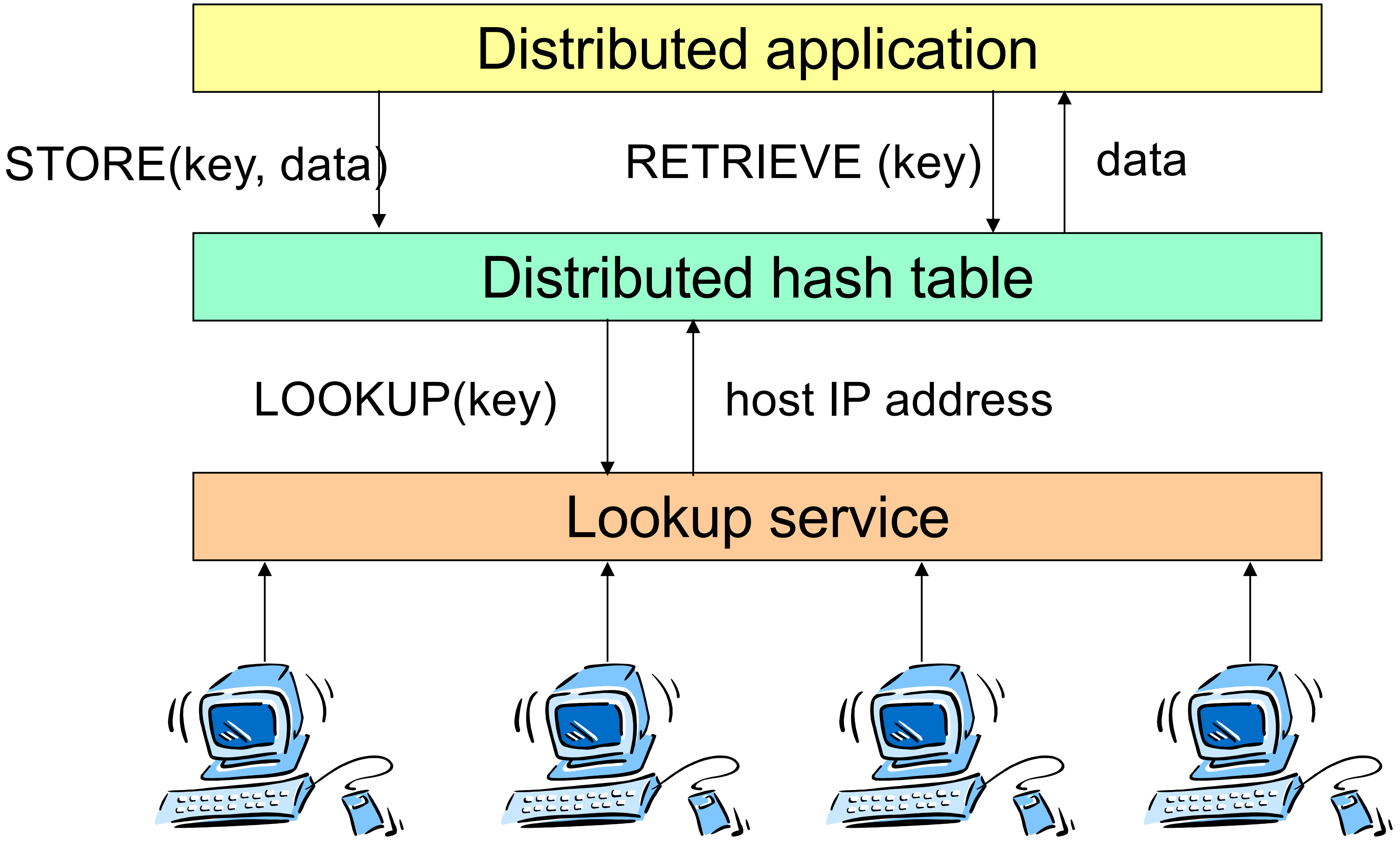
Benefits of P2P and Applications

- High capacity: all clients provide resources (bandwidth, storage space, and computing power). The capacity of the system increases as more nodes become part of the system.
- Increased reliability: achieved by replicating data over multiple peers, and by enabling peers to find the data without relying on a centralized index server.
- Applications:
  - File sharing: Napster, Gnutella, Freenet, BitTorrent
  - Distributed file systems: Ivy
  - Multicast overlays: ESM, NICE, AIML

Issues in P2P Systems Design

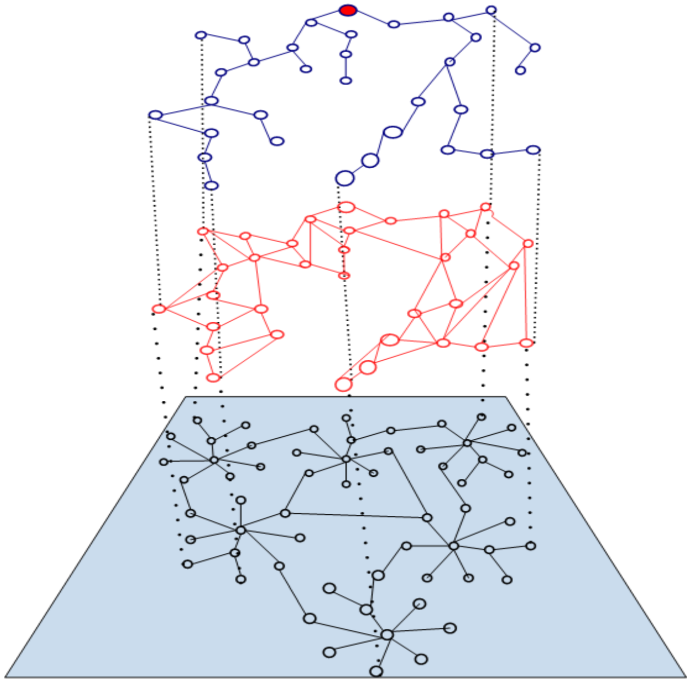
- How do nodes self-organize, what are appropriate structures?
- How to search efficiently or perform more complex queries?
- How to route efficiently on such structures?
- How to maintain performance in spite of crashes, transient failures?
- How to maintain availability in spite of failures and partitions?

Structure of P2P File Sharing Systems



Structure of P2P Multicast Systems

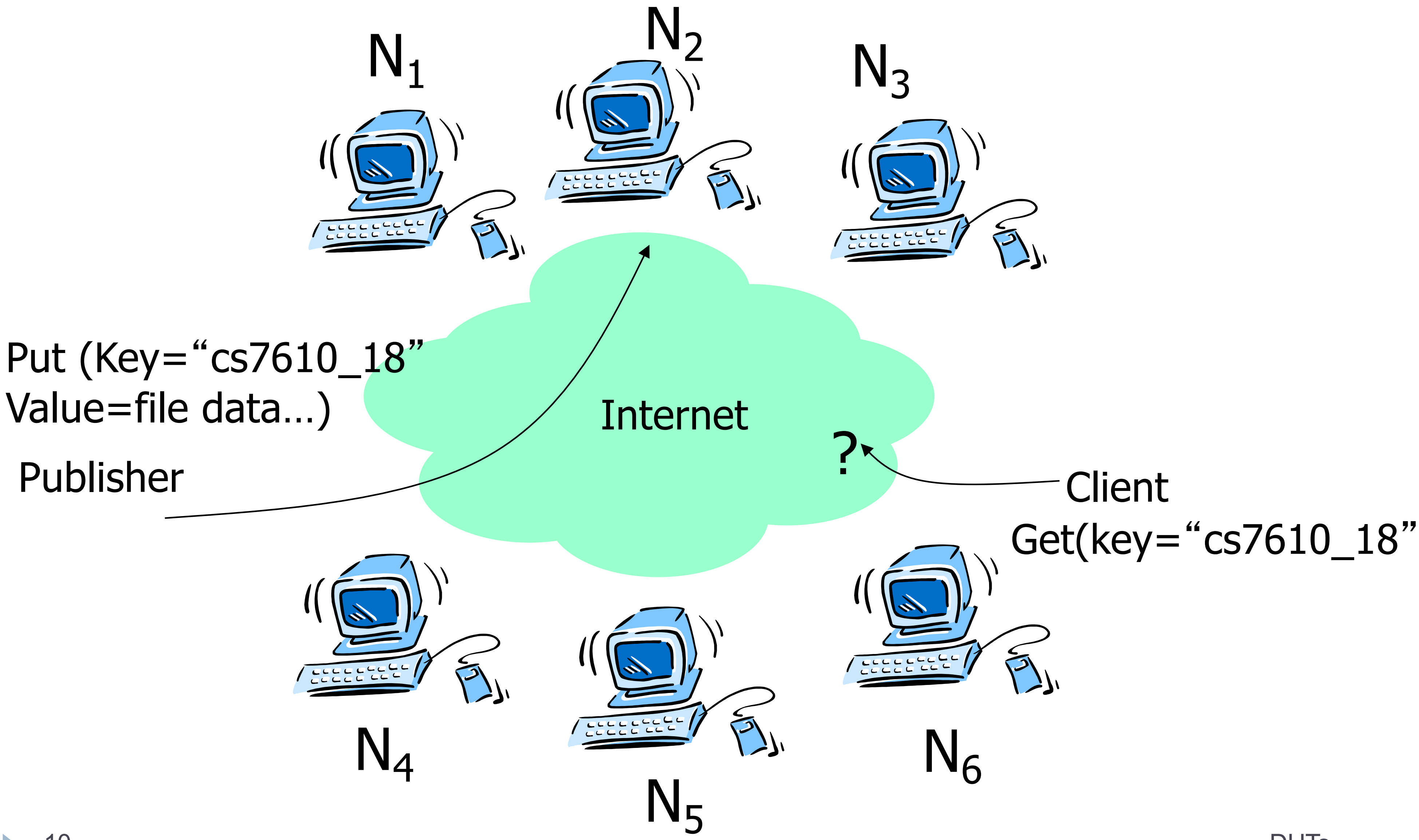
- Extend network functionality by providing multicast services
- Usually build a multicast tree that dynamically adapts to improve suboptimal overlay meshes.
- Overlay is unstructured and optimizations are done by using measurement-based heuristics
- ESM, Nice, Overcast, ALMI



Structured vs. Unstructured P2P

- Many file sharing P2P systems are structured:
  - A small subset of nodes meeting presubscribed conditions are eligible to become neighbors
  - The goal here is to bound the cost of locating objects and the number of network hops
- Many multicast/broadcast P2P systems are not structured:
  - The goal here is maximizing performance in terms of throughput and latency

Why Lookup Services



N_1

N_2

N_3

Internet

Put (Key=“cs7610_18”
Value=file data…)

Publisher

?

Client
Get(key=“cs7610_18”

N_4

N_5

N_6

Challenges for Lookup Services

- Availability
- Scalability
- Complexity
- Exact-match searching vs. approximate marching
- General lookup vs specialized lookup

Architectures for Lookup Services: Centralized

- Central index server maintaining the list of files available in the system
- Upon joining, a node sends the list of files it stores locally, to the central index server
- When performing a search, a node contacts the central index server to find out the location of the file
- Vulnerable to single point of failures
- Maintains O(N) state, costly to maintain the state
- Example: Napster

Architectures for Lookup Services: Flooded Query

- There is no centralized index server
- Each node stores the list of the files it stores locally, no cost on join
- When performing a search, a node floods the query to every other machine in the network
- More robust than the centralized approach, avoids the single point of failure
- Inefficient, worst case O(N) messages per lookup
- Example: Gnutella

Architectures for Lookup Services: Rooted Query

- Completely distributed
- Use a more efficient key-based routing in order to bound the cost of lookup
- Less robust than flooded query approach, but more efficient
- Example: Chord, Pastry, Tapestry, Kademlia

Distributed Hash Tables

- Decentralized distributed systems that partition a set of keys among participating nodes
- Goal is to efficiently route messages to the unique owner of any given key
- Typically designed to scale to large numbers of nodes and to handle continual node arrivals and failures
- Examples: Chord, CAN, Pastry, Tapestry

DHT Design Goals

- Decentralized system:
  - One node needs to coordinate with a limited set of participants to find the location of a file; should work well in the presence of dynamic membership
- Scalability:
  - The system should function efficiently even with thousands or millions of nodes
- Fault tolerance:
  - The system should be reliable even with nodes continuously joining, leaving, and failing

DHT: Keys and Overlays

- Key space:
  - Ownership of keys is split among the nodes according to some partitioning scheme that maps nodes to keys
- Overlay network:
  - Nodes self organize in an overlay network; each node maintains a set of links to other nodes (its neighbors or routing table).
  - Overlay and routing information is used to locate an object based on the associated key

DHT: Storing an Object

- Compute key according to the object-key mapping method
- Send message store(k,data) to any node participating in the DHT
- Message is forwarded from node to node through the overlay network until it reaches the node S responsible for key k as specified by the keyspace partitioning method
- Store the pair (k,data) at node S (sometimes the object is stored at several nodes to deal with node failures)

DHT: Retrieving an Object

- Compute key according to the object-key mapping method
- Send a message to any DHT node to find the data associated with k with a message retrieve(k)
- Message is routed through the overlay to the node S responsible for k
- Retrieve object from node S

Key Partitioning

- Key partitioning: defines

what node “owns what keys” <=> ”stores what objects”

- Removal or addition of nodes should not result in entire remapping of key space since this will result in a high cost in moving the objects around
- Use consistent hashing to map keys to nodes. A function d(k1,k2) defines the distance between keys k1 to key k2. Each node is assigned an identifier (ID). A node with ID i owns all the keys for which i is the closest ID, measured according to distance function d.
- Consistent hashing has the property that removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs, and leaves all other nodes unaffected.

Overlay Networks and Routing

- Nodes self-organize in a logical network defined by the set of links to other nodes each node must maintain
- Routing:
  - Greedy algorithm, at each step, forward the message to the neighbor whose ID is closest to k.
  - When there is no such neighbor, then this is the closest node, which must be the owner of key k

2: Chord

CHORD

- Efficient lookup of a node which stores data items for a particular search key.
- Provides only one operation: given a key, it maps the key onto a node.
- Example applications:
  - Co-operative mirroring
  - Time-shared storage
  - Distributed indexes
  - Large-scale combinatorial search

Design Goals

- Load balance: distributed hash function, spreading keys evenly over nodes
- Decentralization: CHORD is fully distributed, nodes have symmetric functionality, improves robustness
- Scalability: logarithmic growth of lookup costs with number of nodes in network
- Availability: CHORD guarantees correctness, it automatically adjusts its internal tables to ensure that the node responsible for a key can always be found

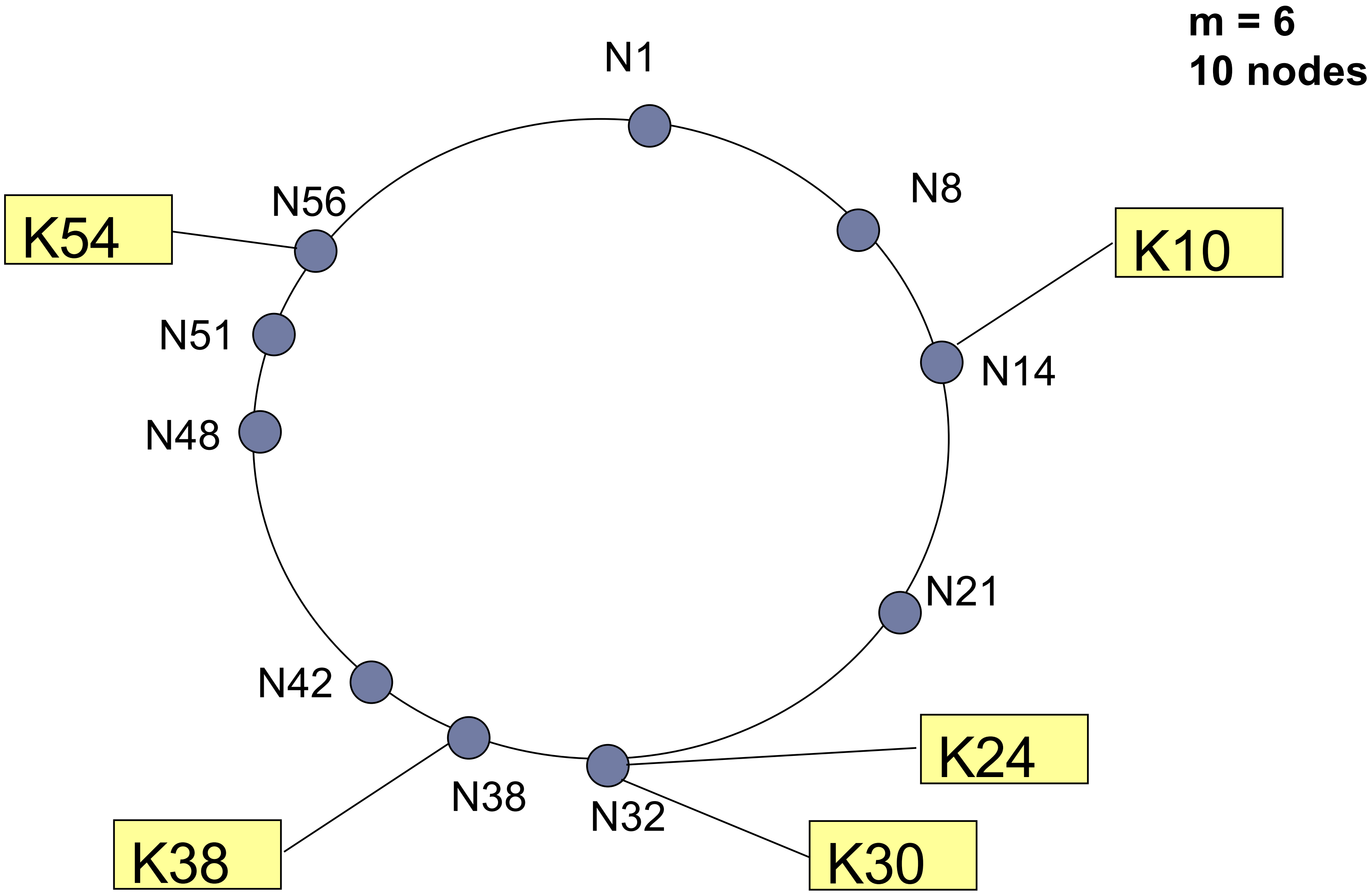
Assumptions

- Communication in underlying network is both symmetric and transitive
- Assigns keys to nodes with consistent hashing
- Hash function balances the load
- Participants are correct, nodes can join and leave at any time
- Nodes can fail

Chord Rings

- Key identifier = SHA-1(key)
- Node identifier = SHA-1(IP address)
- Consistent hashing function assigns each node and key an m-bit identifier using SHA-1
- Mapping key identifiers to node identifiers:
  - Identifiers are ordered on a circle modulo 2^m called a chord ring.
  - **The circle is split into contiguous segments whose endpoints are the node identifiers. If i1 and i2 are two adjacent IDs, then the node with ID greater identifier i2 owns all the keys that fall between i1 and i2.**

Example of Key Partitioning in Chord

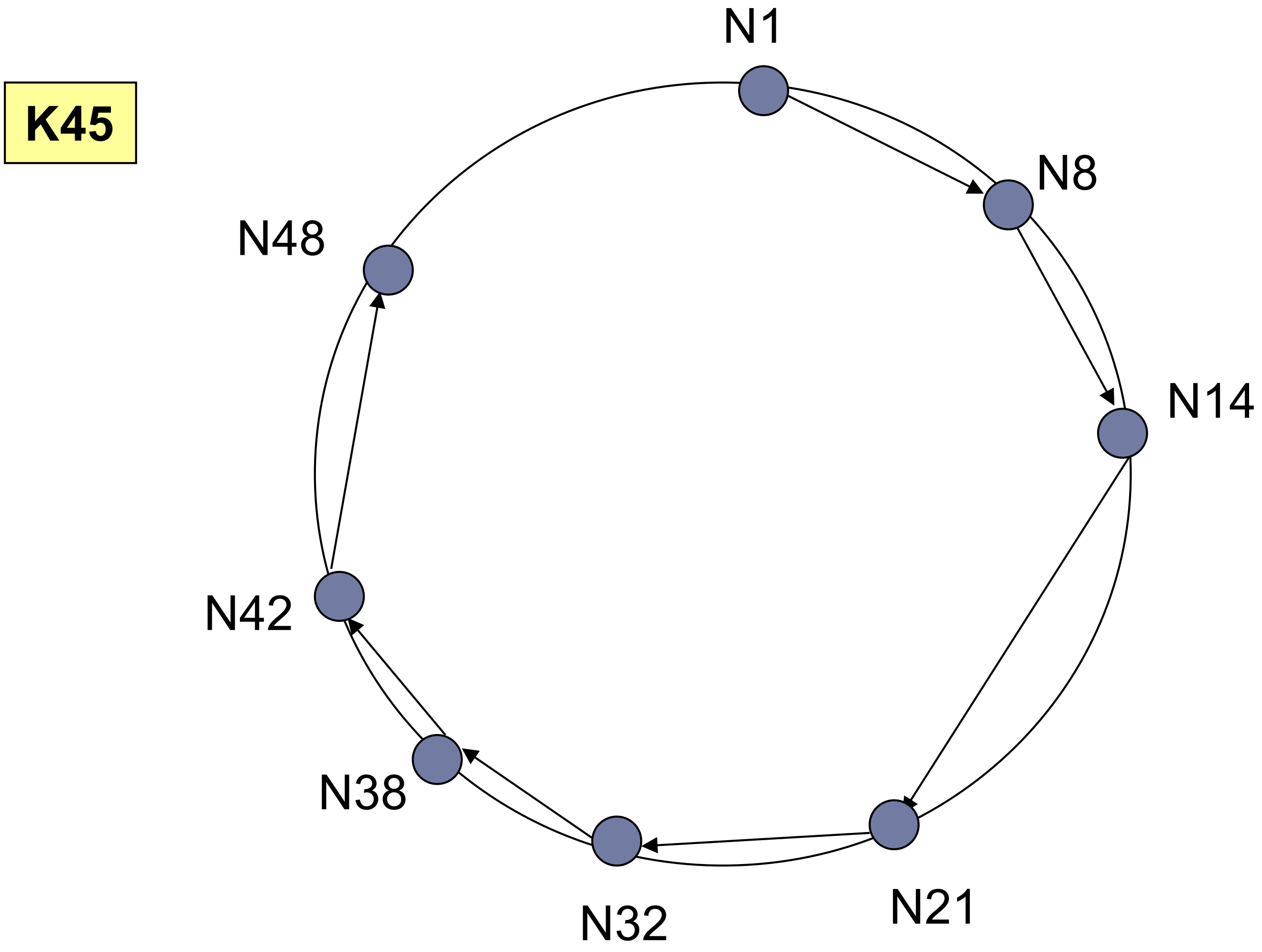


How to Perform Key Lookup

- Assume that each node knows only how to contact its current successor node on the identifier circle, then all node can be visited in linear order.
- When performing a search, the query for a given identifier could be passed around the circle via these successor pointers until they encounter the node that contains the key corresponding to the search.

Example of Key Lookup Scheme

succesor(k) = first node whose ID is >= ID of k in identifier space

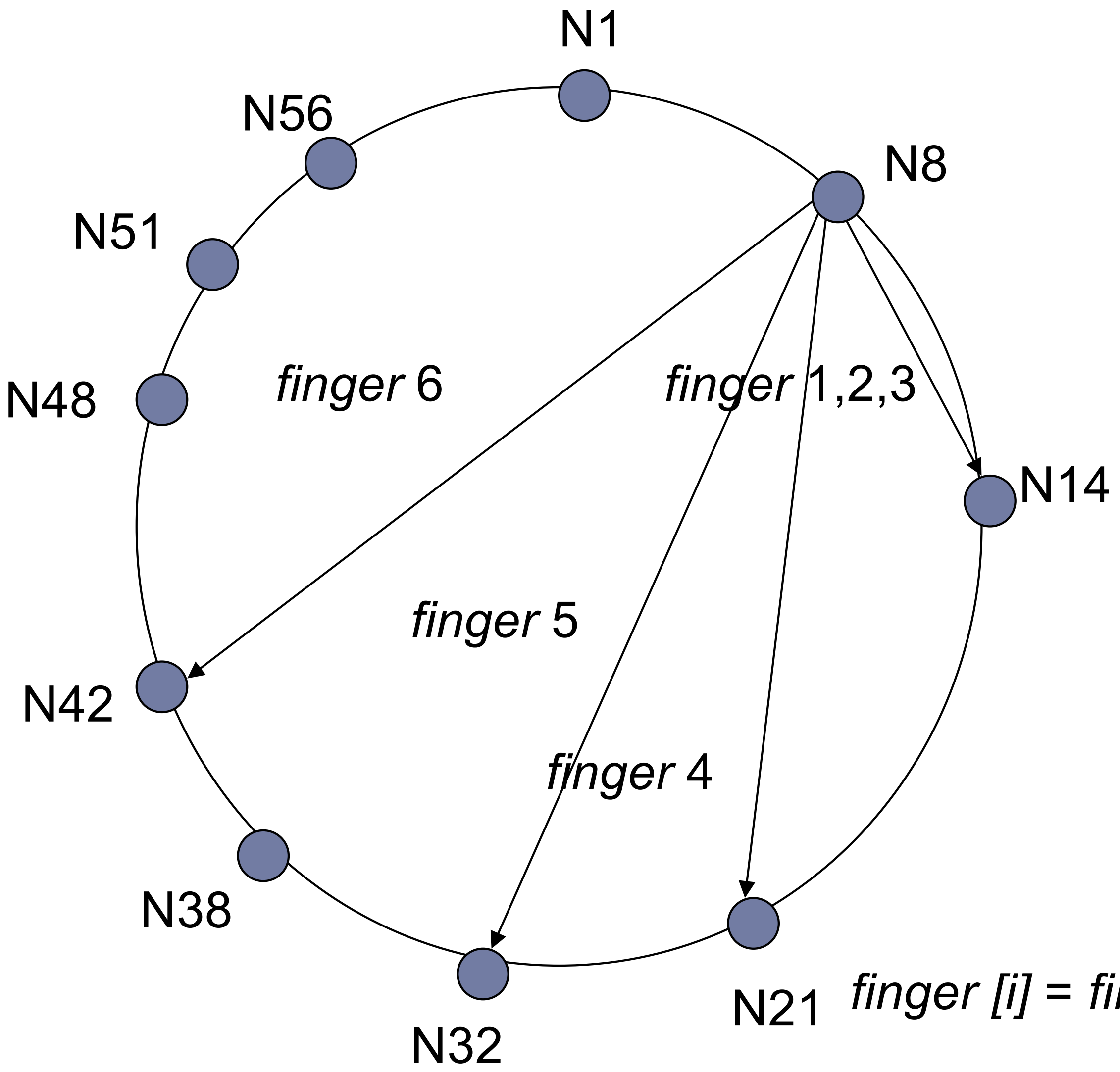


Scalable Key Location

- To accelerate lookups, Chord maintains additional routing information (m entries): finger table
- The ith entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle.
- s = successor($n+2^{i-1}$).
- s is called the ith finger of node n

Scalable Lookup Scheme

m = 6



Finger Table for N8

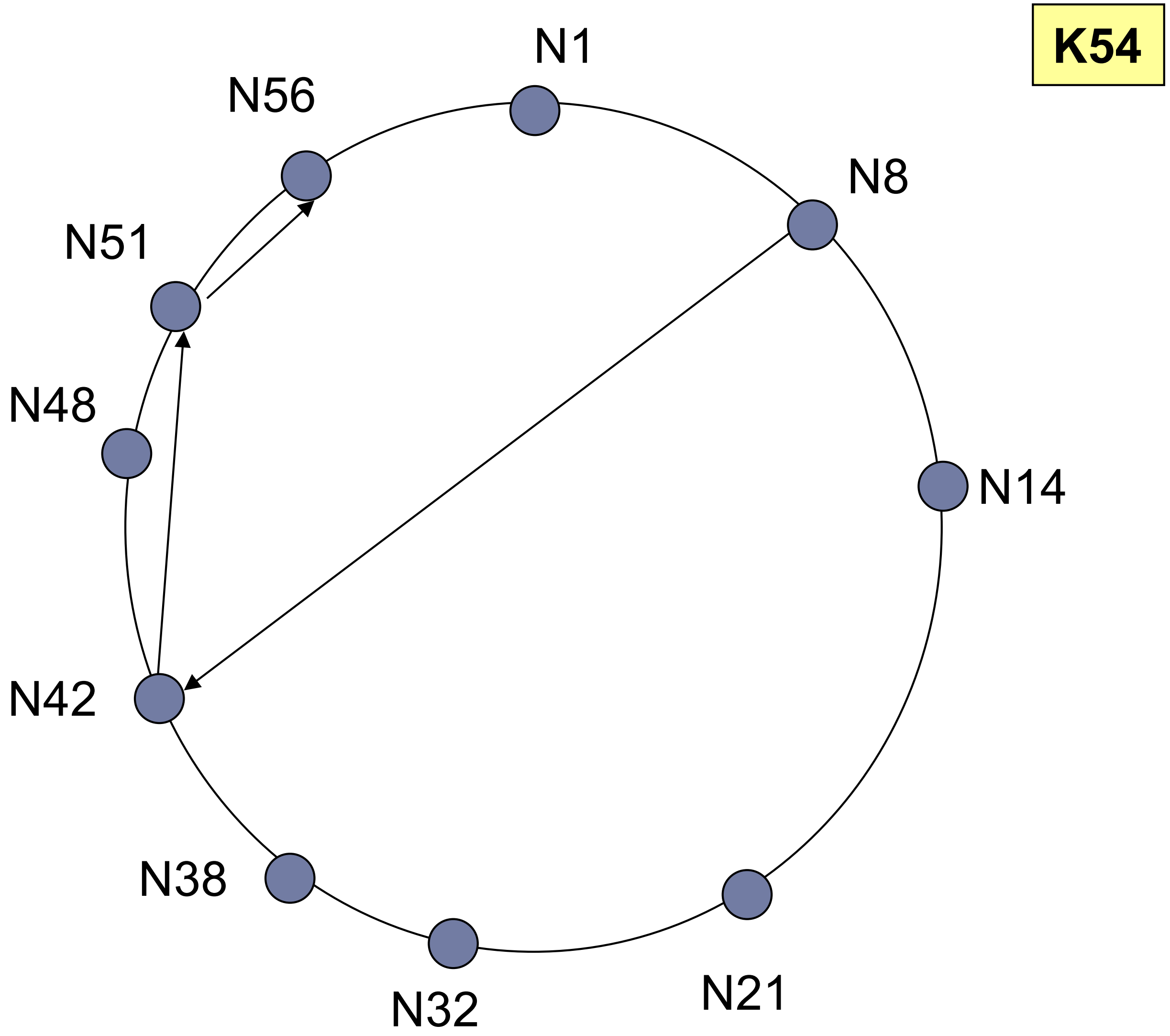
N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

finger [i] = first node that succeeds $(n+2^{i-1})\text{mod}2^{m}$

Scalable Lookup

- Each node has finger entries at power of two intervals around the identifier circle
- Each node can forward a query at least halfway along the remaining distance between the node and the target identifier.

Lookup Using Finger Table



Node Joins and Failures/Leaves

- When a node N joins the network, some of the keys previously assigned to N' s successor should become assigned to N.
- When node N leaves the network, all of its assigned keys should be reassigned to N' s successor.
- How to deal with these cases?

Node Joins and Stabilizations

- Everything relies on successor pointer.
- Up to date successor pointer is sufficient to guarantee correctness of lookups
- Idea: run a "stabilization" protocol periodically in the background to update successor pointer and finger table.

Stabilization Protocol

- Guarantees to add nodes in a fashion to preserve reachability
- Does not address the cases when a Chord system thas split into multiple disjoint cycles, or a single cycle that loops multiple times around the identifier space

Stabilization Protocol (cont.)

- Each time node N runs stabilize protocol, it asks its successor for its predecessor p, and decides whether p should be N's successor instead.
- Stabilize protocol notifies node N's successor of N's existence, giving the successor the chance to change its predecessor to N.
- The successor does this only if it knows of no closer predecessor than N.

Impact of Node Joins on Lookups

- If finger table entries are current then lookup finds the correct successor in O(log N) steps
- If successor pointers are correct but finger tables are incorrect, correct lookup but slower
- If incorrect successor pointers, then lookup may fail

Voluntary Node Departures

- Leaving node may transfers all its keys to its successor
- Leaving node may notify its predecessor and successor about each other so that they can update their links

Node Failures

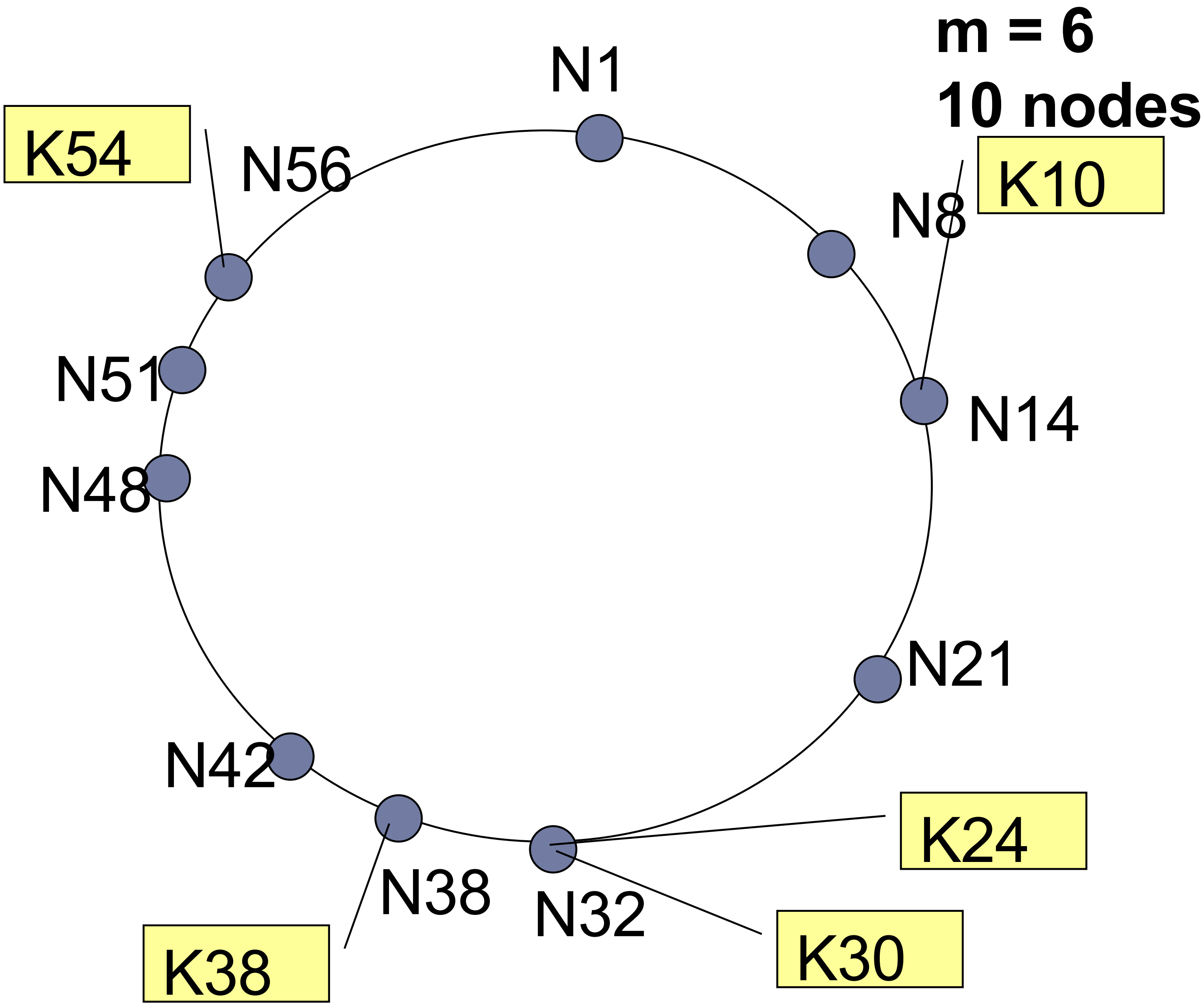
- Stabilize successor lists:
  - Node N reconciles its list with its successor S by copying S’ s successor list, removing its last entry, and prepending S to it.
  - If node N notices that its successor has failed, it replaces it with the first live entry in its successor list and reconciles its successor list with its new successor.

CHORD Summary

- Efficient location of the node that stores a desired data item is a fundamental problem in P2P networks
- Separates correctness (successor) from performance (finger table)
- Chord protocol solves it in a efficient decentralized manner
  - Routing information: O(log N) nodes
  - Lookup: O(log N) nodes
  - Update: O(log2 N) messages
- It also adapts dynamically to the topology changes introduced during the run

Node ID Assignment Implications

- Node ID determines where will the node be placed in the structured overlay
- Determines who the neighbors are going to be
- Determines what objects a node will hold



Attacks Based on Node ID Assignment

- What if attacker can choose the ID of a node?
  - Surround a victim node
  - Partition a p2p network
  - Can control what object will be a replica for
  - Holding objects allow an attacker to delete, corrupt or deny access to objects
- Can an attacker choose the ID of a node?
  - In some systems ID is randomly generated
  - In some systems ID is the hash of the IP address

Sybil Attack

- Attack particular to peer networks, a malicious attacker takes/forges multiple identities
- Result: attacker controls a significant part of the system while correct nodes are not aware of this, they see different identities
  - destroy cohesion of the overlay
  - observe network status
  - slow down, destroy overlay
  - DoS
- How to ensure/validate distinct identities refer to distinct entities?

Evaluating Identity

- Straightforward form of identity: secure hash of a public key
- How to evaluate/learn the identity of other entities
  - Use a trusted agency (learn from trusted source)
  - A node has a direct way of validating other nodes - direct validation (learn directly)
  - Using other untrusted agencies - indirect validation (learn from others)
- Which one is best?

Direct and Indirect Validation (Untrusted Sources)

- Utilize computational tasks to validate distinctness;
  - validate distinctness of two entities by getting them to perform some task (for example a computational puzzle) that a single entity could not
  - can not assume homogeneous resources, only minimum; faulty entity could have more than minimum
  - The goal is to make it practical impossible for an adversary to have challenges issued simultaneously, limit the number of identities he can forge

Direct Validation Limitations

- Even with severely resource constrains, a faulty entity can counterfeit a constant number of identities
- Each correct entity must simultaneously validate all the identities it is presented otherwise a faulty entity can counterfeit an unbounded number of identities

Indirect Validation

- A sufficiently large set of faulty entities can counterfeit an unbounded number of identities
- All entities in the system must perform their identity validations concurrently, otherwise a faulty identity can counterfeit a constant number of multiple identities

Certified (Secure) NodeID Assignment

- Delegate ID generation to trusted CAs
- Bind IPs with nodeIds such that colluding attackers can not exchange certificates
- Nodes must pay for certificates to prevent attackers from buying many "correct" certificates
- Works for static IP addresses
- Does not solve all problems: what happens if the IP changes?
- What happens if the trusted CA is not available or can not be reached?

Certified (Secure) NodeID Assignment

- How about distributed ID generation with periodic renewal of distributed IDs
  - Addresses single point of failure
  - Requires techniques to moderate the rate at which attackers can acquire node IDs

Routing Table Maintenance

- Routing table contains information about where to ‘look next’
- Table is updated based on information from other nodes

m = 6

Finger Table for N8

N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

finger [k] = first node that succeeds $(n+2^{k-1})\text{mod}2^m$

Attack Against Maintaining Routing Table

- Attackers can easily supply 'malicious' updates or can return incorrect lookup
  - point to faulty or non-existent nodes
  - fake the closest node
  - lie about next hop
- Result: lookup will fail (denial of information to a node) or the lookup algorithm will have sub-optimal performance

Secure Routing Table Maintenance

- Constrained Routing Tables: Identify invariants in the system and look for violations of the invariants
- Maintain two routing tables - one that uses proximity information and one that constrains entries to "specific" values
  - Proximity routing used in normal operation
  - Constrained routing used when failures occur:
- Other proposed solutions involve anonymous auditing

Secure Bootstrapping

- How to securely bootstrap the routing table?
  - A new node, n, picks a subset of bootstrap nodes to query and join the network
  - n uses the bootstrap information to initialize its constrained routing table

Attacks on Forwarding

- Simply ignore forwarding messages, route to the wrong node
  - Failed if ANY one in routing is faulty
  - Probability of routing successfully to a replica root is $(1\text{-}f)^{h-1}$
  - h is the number of average hops for delivering a message
  - h depends on the overlay

Secure Message Forwarding

- Ensure that with high probability, at least on copy of a message reaches every correct replica root
- Collect the prospective set of replica roots from the prospective root node
- Apply **routing failure test** to determine if routing worked correctly, If no, use ***redundant*** **and/or** ***iterative routing.***

Testing Routing

- Route Failure Test:
  - Average density of nodeIds per unit of "volume" in the id space is greater than the average density of faulty nodes
  - Compares density of nodeIds in the neighbor set of the sender with the density of nodeIds close to the replica roots of the destination key
  - Have sender contact all prospective roots
  - Timeout to detect ignoring routing msgs, selecting the appropriate threshold not easy
- Use redundant routing when test fails
  - Neighbor set anycast - sends copies of message towards destination until they reach a node with the key in its neighbor set.
- How about false positives and false negatives when performing the routing failure test?
- Redundant routing has high overhead?

Iterative Routing

- Alternative to redundant routing
- Every lookup answer goes back to the requester that can verify that the next hop gets him closer (using the distance function) to the node hosting the object associated with the requested key
- Iterative routing is more secure, but more expensive

What Does Secure Routing Buy Us?

- Prevents attacks at join time: secure nodeID assignment and bootstrapping
- Ensure that when a correct node sends a message for a particular key, the message reaches all correct replica roots for the key with very high probability.
- What about the data? We need other mechanisms, for example self-certifying data

Self-Certifying Data

- Client can check data and only needs to rely on routing when certification check fails.
- Reduces the reliance on the redundant, secure routing primitive (you still need secure forwarding otherwise there is no data to verify in the first place)
- Uses concepts like proactive signature sharing or group keys/signatures.
- Self-certifying data can eliminate the overhead of secure routing in common cases

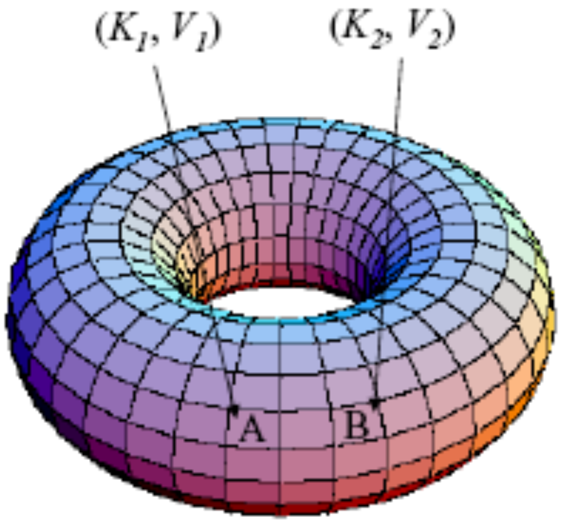
3: CAN

Content Addressable Network

- Each node stores a chunk (zone) of the hash table
- Each node stores state information about neighbor zones
- Requests (insert, lookup, or delete) for a key are routed by intermediate nodes using a greedy routing algorithm
- Completely distributed
- Nodes can crash
- Keys mapped to a d dimensional space

Design of CAN

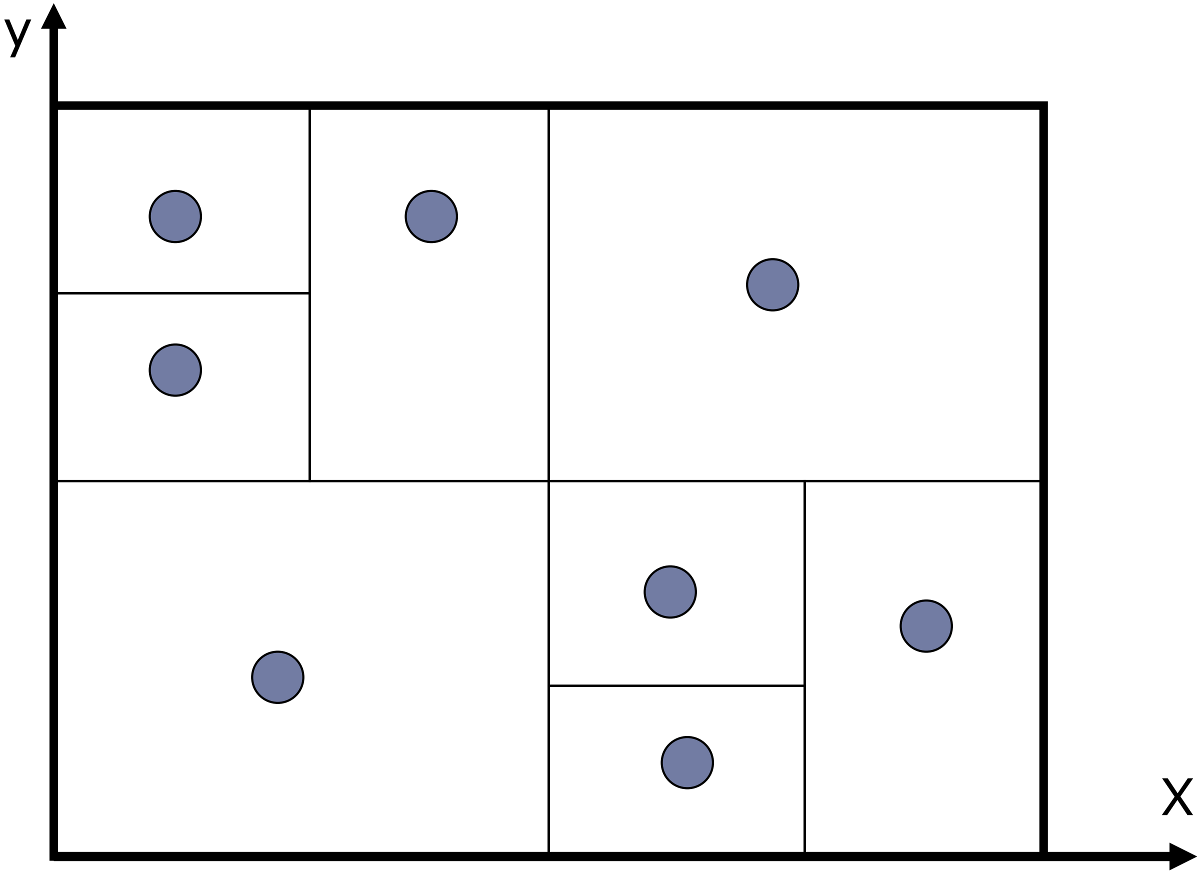
- d-dimensional Cartesian coordinate space (d-torus)
- Each node owns a zone on the toru
- To store key value pair (K1,V1),
  - K1 mapped to point P1 using uniform
  - (K1,V1) stored at the node N that ow P1



2-torus

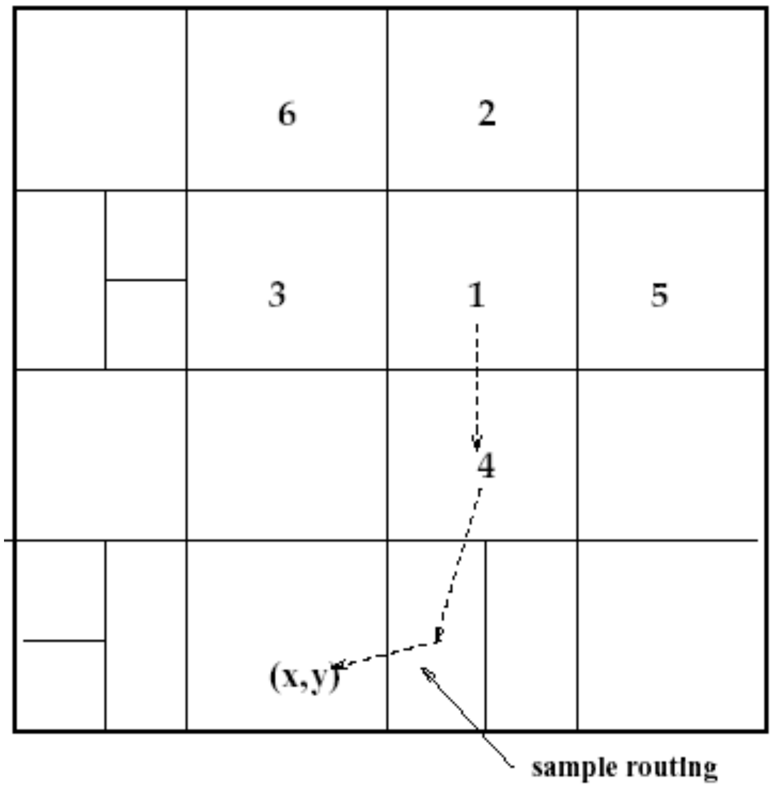
Key Partitioning

- Virtual d-dimensional Cartesian coordinate dynamically partitioned among all nodes
- Key partitioning using a uniform hash function



Routing in CAN

- Each node maintains a table of the IP address and virtual coordinate zone of each local neighbor
- Use greedy routing to neighbor closest to destination

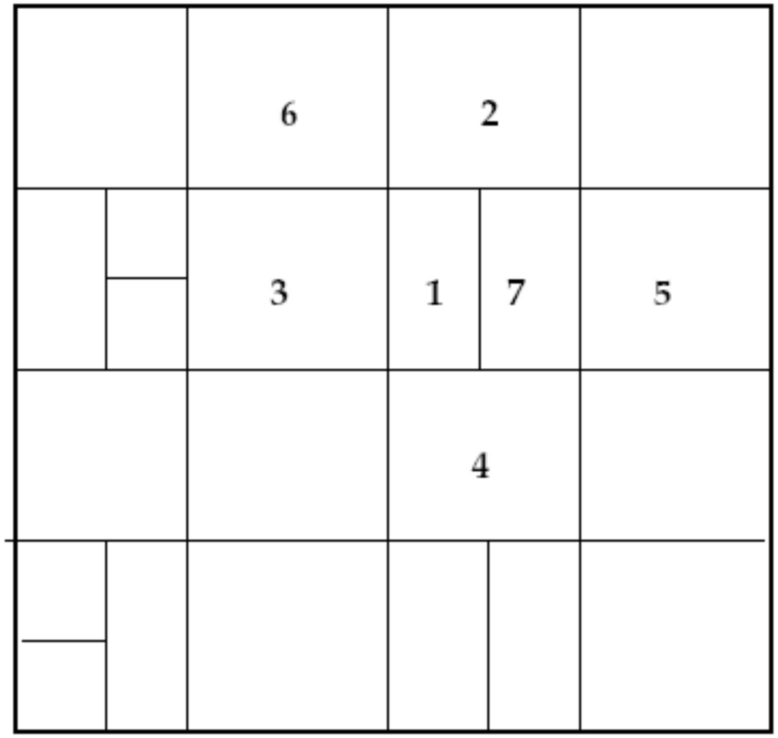


sample routing path from node 1 to point (x,y)

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

Node Join

- Joining node locates a bootstrap node B (random member) using the CAN DNS entry
- Node in zone containing B splits the zone and allocates "half" to joining node
- Keys and objects are transferred to new node
- Joining node and previous node zone update their neighbor set



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

Node Voluntarily Leave and Failure

- Leave: node transfers its zone and the pairs of keys and data to a neighbor
- Failure: unreachable node(s) trigger an immediate takeover algorithm that allocate failed node' s zone to a neighbor
- Multiple concentrated failures: neighbors may need to perform an expanding ring search to build sufficient neighbor state prior to initiating the takeover algorithm
- Background zone-reassignment algorithm to prevent space fragmentation
  - Over time nodes may takeover zones that cannot be merged with its own zone

4: Pastry

Pastry

- Each node in Pastry has a unique, uniform random identifier (nodeID) in a circular 128-bit identifier space (as in Chord)
- Each object has a unique, uniform random identifier (objectID)
- Node with numerically closest nodeID maintains the object
- Routing is done to the numerically closest to the key that is searched
- Network of N nodes
  - Expected number of forwarding steps in the Pastry overlay network is O(log N)
  - Size of the routing table maintained at each node is O(log N)

Routing

- Address blocks: 128-bit key is divided up into digits, each digit being b bits long
  - level 0 represents a zero-digit common prefix between two addresses, level 1 a one-digit common prefix, and so on.
- Leaf set: IP addresses of the nodes with the L/2 numerically closest peers by nodeID in each direction around the circle
- Neighbor list: M closest peers with respect to the routing metric.
- Routing table: address of the closest known peer for each possible digit at each address level, except for the digit that belongs to the peer itself at that particular level.
- Storage of 2b – 1 contacts per level, number of levels scaling as (log 2N) / b, b=4, L = 2b and M=2b are typical settings

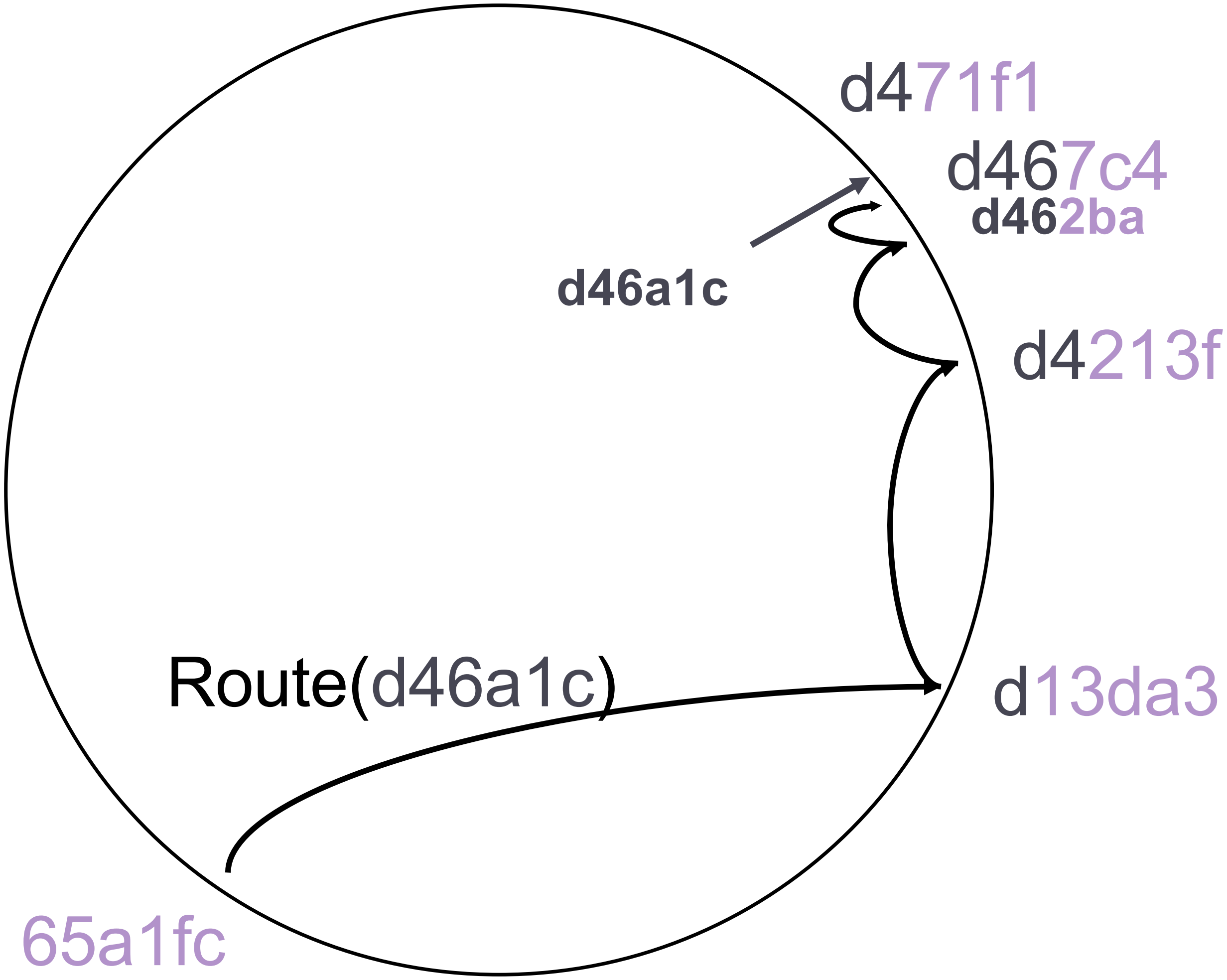
Routing Table of Node 65a1fc$x$

Row 0	*0x*	*1x*	*2x*	*3x*	*4x*	*5x*		*7x*	*8x*	*9x*	*ax*	*bx*	*cx*	*dx*	*ex*	*fx*
Row 1	*60x*	*61x*	*62x*	*63x*	*64x*		*66x*	*67x*	*68x*	*69x*	*6ax*	*6bx*	*6cx*	*6dx*	*6ex*	*6fx*
Row 2	*650x*	*651x*	*652x*	*653x*	*654x*	*655x*	*656x*	*657x*	*658x*	*659x*		*65bx*	*65cx*	*65dx*	*65ex*	*65fx*
Row 3	*65a0x*		*65a2x*	*65a3x*	*65a4x*	*65a5x*	*65a6x*	*65a7x*	*65a8x*	*65a9x*	*65aax*	*65abx*	*65acx*	*65adx*	*65aex*	*65afx*

$\log_{16}$ N rows

From www.cs.rice.edu/~druschel/comp413/lectures/Pastry.ppt

Routing intuition



Properties

- $\log_{16}$ N steps
- O(*log N*) state

From www.cs.rice.edu/~druschel/comp413/lectures/Pastry.ppt

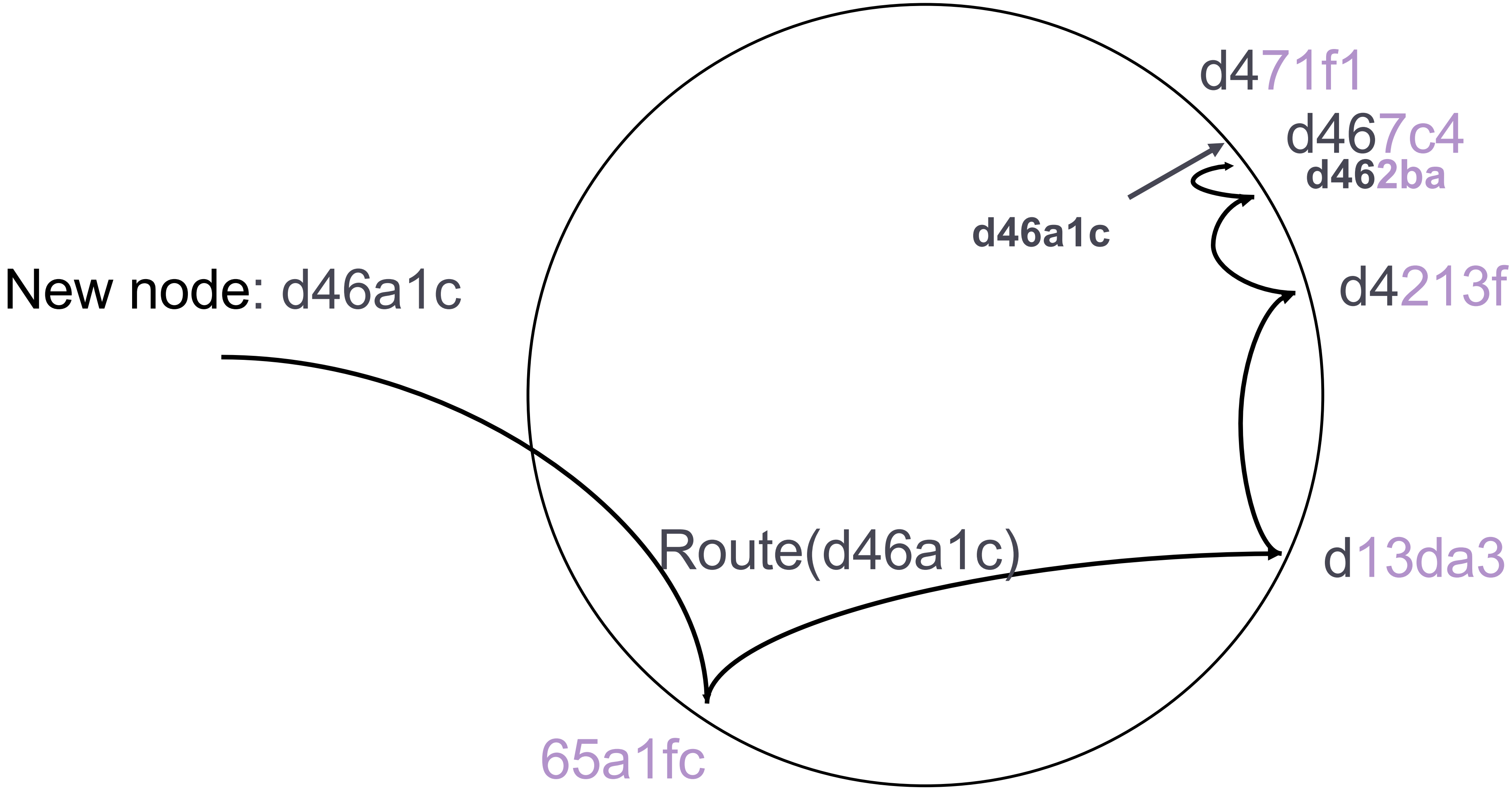
Mapping Objects to Nodes

- Objects are assigned unique, uniform random identifiers (objIDs) and mapped to the k nodes with nodeIDs numerically closest to the objId.
- Inserting objects : the message reaches a node with one of the k closest nodeIDs to the objID, that node replicates the object among the other k-1 nodes with closest nodeIDs (which are, by definition, in the same leaf set for k <= L/2).
- Searching for objects: Application-specific objects can be looked up, contacted, or retrieved by routing a Pastry message, using the objID as the key. By definition, the message is guaranteed to reach a node that maintains a replica of the requested object unless all k nodes with nodeIDs closest to the objID have failed.

Routing

- A peer first examines its leaf set and routes directly to the correct node if one is found.
- If this fails, the peer checks its routing table to find the address of a node which shares a longer prefix with the destination address than the peer itself.
- If the peer does not have any contacts with a longer prefix or the contact has died it will pick a peer from its contact list with the same length prefix whose node ID is numerically closer to the destination and send the packet to that peer.

Node Addition



From www.cs.rice.edu/~druschel/comp413/lectures/Pastry.ppt

Node departure (failure)

- Leaf set members exchange keep-alive messages

- Leaf set repair (eager): request set from farthest live node in set
- Routing table repair (lazy): get table from peers in the same row, then higher rows

From www.cs.rice.edu/~druschel/comp413/lectures/Pastry.ppt

5: Kademlia

Kademlia in a Nutshell

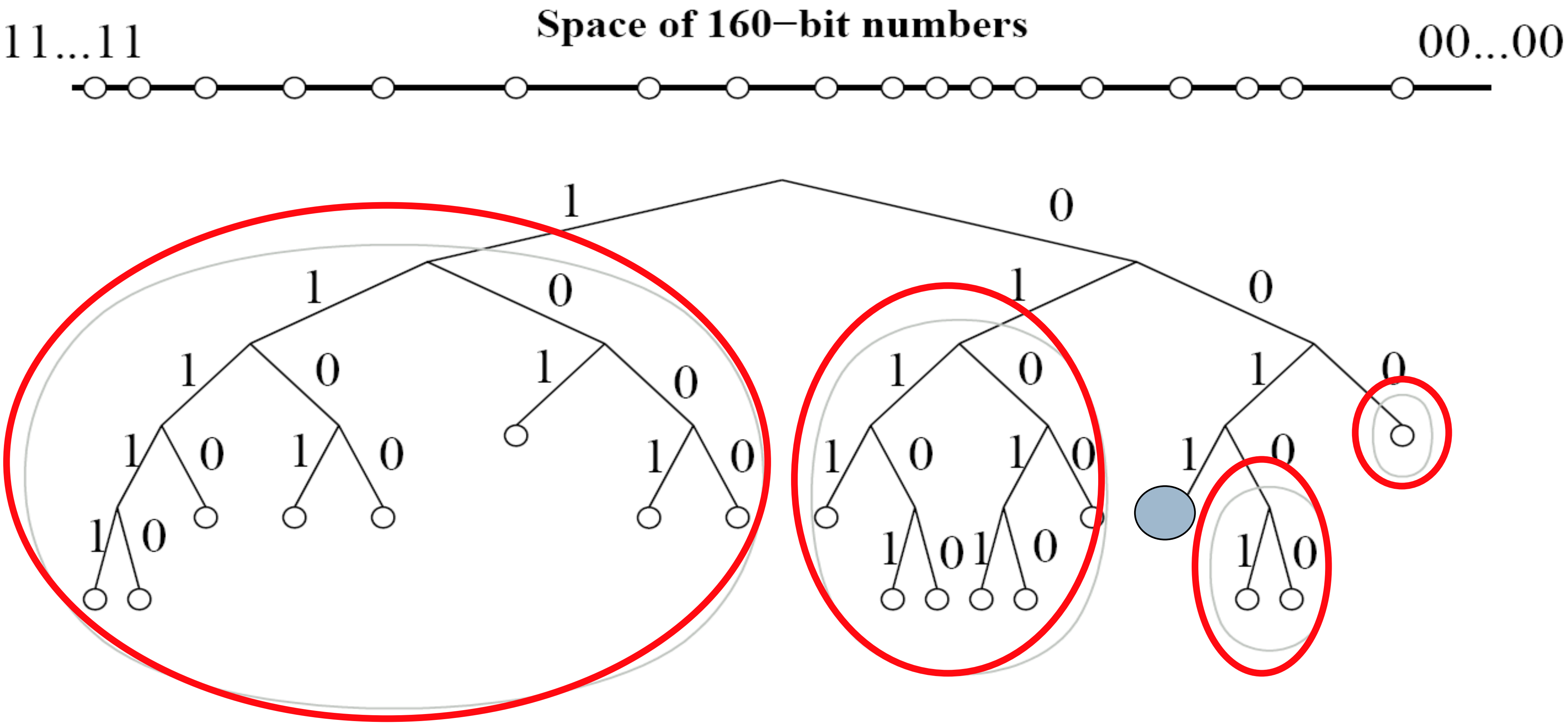
- Similar with other services, IDs based on SHA-1 hash into a 160 bits space.
- Closeness between two objects measured as their bitwise XOR interpreted as an integer.
- distance(a, b) = a XOR b
- Distance is symmetric, dist (a,b) = dist (b,a)
- Uses parallel asynchronous queries to avoid timeout delays of the failed nodes. Routes are selected based on latency
- Kademlia uses tree-based routing

Kademlia Binary Tree

- Start from root, for any given node, dividing the binary tree into a series of successively lower subtrees that do not contain the node and each correspond to a k-bucket
- Every node keeps track of at least one node from each of its subtrees.
- Every node keeps a list of (IP, Port, NodeID) triples, and (key, value) tuples for further exchanging information with others.

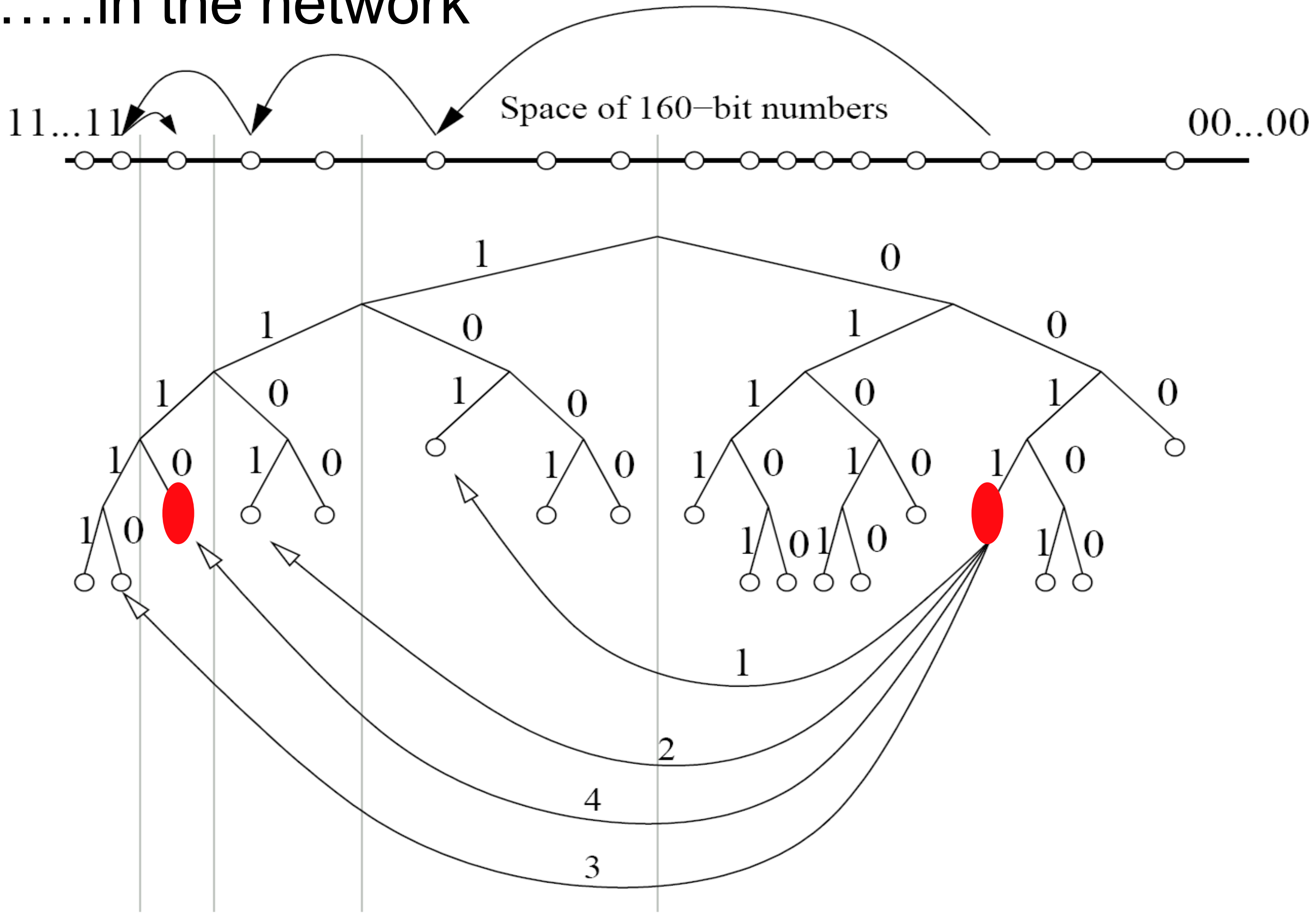
Kademlia Binary Tree

Subtrees for a node 0011……



Kademlia Search

An example of lookup: node 0011 is searching for 1110……in the network



Kademlia Lookup

- Locate the k closest nodes to a given nodeID.
- Uses a recursive algorithm for node lookups.
  - The lookup initiator starts by picking a node from its closest non-empty k-bucket.
  - The initiator then sends parallel, asynchronous FIND_NODE to the α nodes it has chosen.
  - The initiator resends the FIND_NODE to nodes it has learned about from previous requests.
  - If a round of FIND_NODES fails to return a node any closer than the closest already seen, the initiator resends the FIND_NODE to all of the k closest nodes it has not already queried.

Kademlia Keys Store

- To store a (key,value) pair, a participant locates the k closest nodes to the key.
- Additionally, each node re-publishes (key,value) pairs as necessary to keep them alive
- Kademlia's current application (file sharing), requires the original publisher of a (key,value) pair to republish it every 24 hours. Otherwise, (key,value) pairs expire 24 hours after publication.

Kademlia Cost

- Operation cost
  - As low as other popular protocols
  - Look up, O(logN)
  - Join or leave, O(log2N)
- Fault tolerance and concurrent change
  - Handles well, for the use of k-buckets
- Proximity routing
  - Kademlia can choose from α nodes that has lower latency

6: Pub-sub

Slides by Yu-Ling Chang

Publish/Subscribe System

- Distributed Pub/Sub System is a communication paradigm that allows freedom in the distributed system by the decoupling of communication entities in terms of time, space and synchronization.
- An event service system that is asynchronous, anonymous and loosely-coupled.
- Ability to quickly adapt in a dynamic environment.

Key components

- Publishers : Publishers generate event data and publishes them.
- Subscribers : Subscribers submit their subscriptions and process the events received
- P/S service: It's the mediator/broker that filters and routes events from publishers to interested subscribers.

Publish/Subscribe System

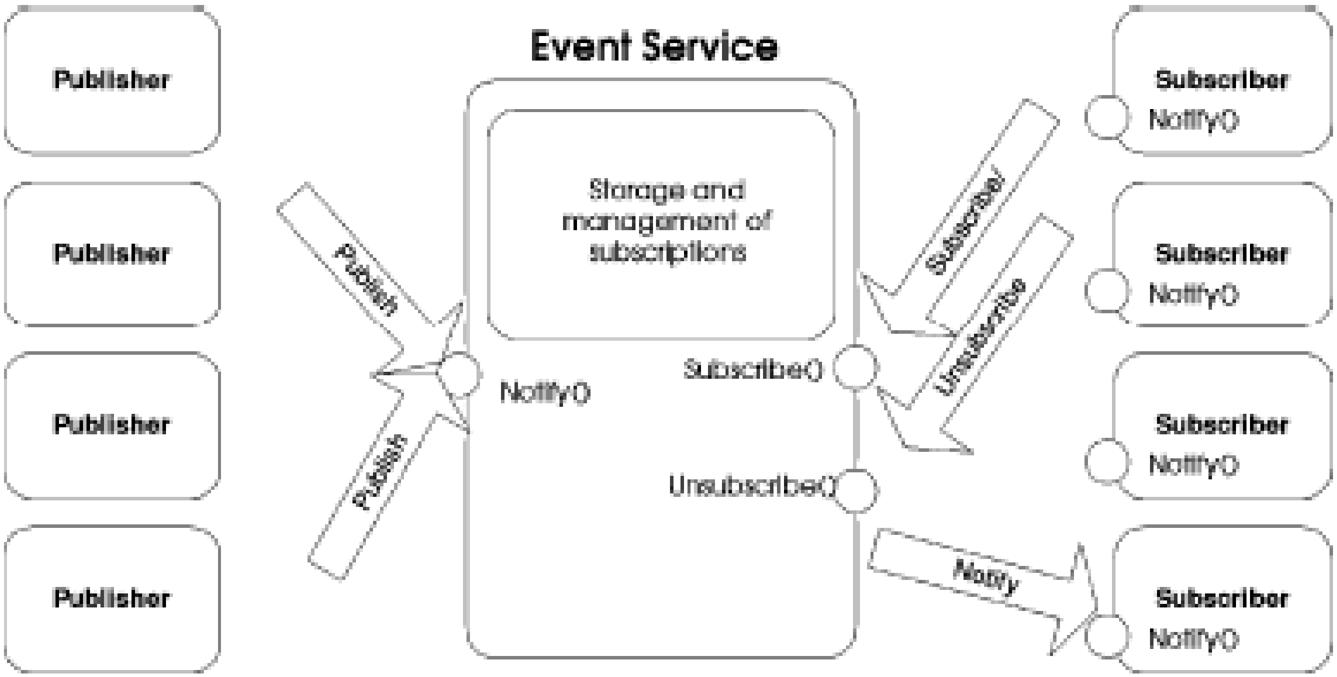
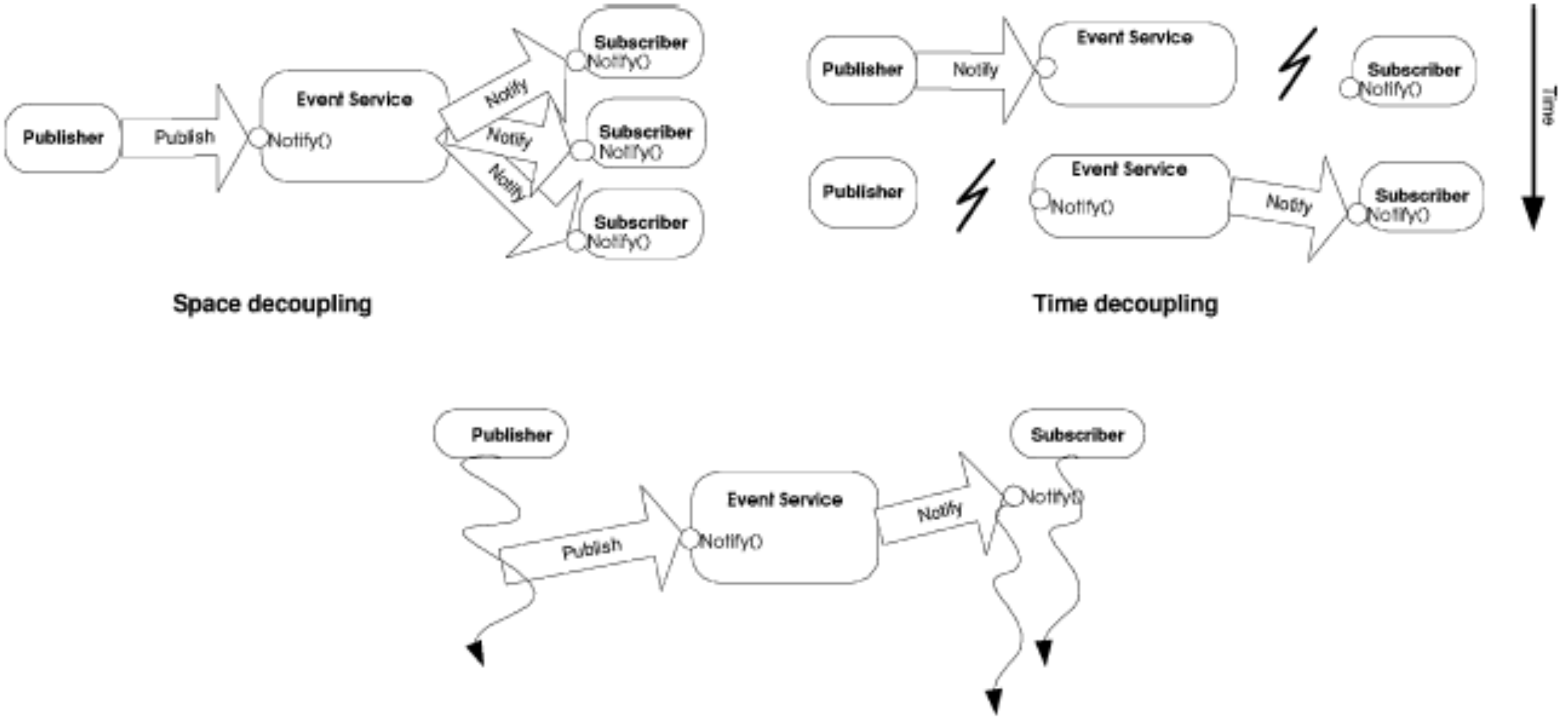


Fig. 1. A simple object-based publish/subscribe system.

Time, space and synchronization

- Provides decoupling in time, space and synchronization.



Space decoupling

Time decoupling

Synchronization decoupling

Architectures

- **Centralized Broker model**
  - Consists of multiple publishers and multiple subscribers and centralized broker/brokers (an overlay network of brokers interacting with each other).
  - Subscribers/Publishers will contact 1 broker, and does not need to have knowledge about others.
  - E.g. CORBA event services, JMS, JEDI etc…

Architectures

- **Peer-to-Peer model**
  - Each node can be publisher, subscriber or broker.
  - Subscribers subscribe to publishers directly and publishers notify subscribers directly. Therefore they must maintain knowledge of each other.
  - Complex in nature, mechanisms such as DHT and CHORD are employed to locate nodes in the network.
  - E.g. Java distributed event service

Middleware service

- Event filtering (event selection)
  - The process which selects the set of subscribers that have shown interest in a given event. Subscriptions are stored in memory and searched when a publisher publishes a new event.
- Event routing (event delivery)
  - The process of routing the published events from the publisher to all interested subscribers

Event Filtering (Subscription Model)
Topic based VS Content based

- Topic based
  - Generally also known as topic based, group based or channel based event filtering.
  - Each event is published to one of these channels by its publisher.
  - Subscribers subscribes to a particular channel and will receive ALL events published to the subscribed channel.

Topic-based subscription

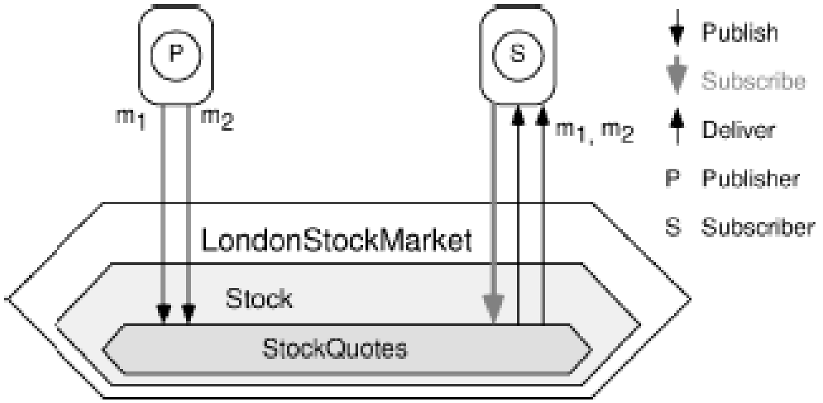


Fig. 12. Topic-based publish/subscribe interactions.

- Simple process for matching an event to subscriptions. However, limited expressiveness.
- Event filtering is easy, event routing is difficult (Heavy load on the network). The challenge is to multicast event effectively to subscribers.

Event Filtering- Subscription Model
Topic based VS Content based

- Content based
  - More flexibility and power to subscribers, by allowing more expression in arbitrary/customized query over the contents of the event.
  - Event publication by a key/value attribute pair, and subscriptions specify filters using a explicit subscription language.
  - E.g. Notify me of all stock quotes of IBM from New York stock exchange if the price is greater than 150

Content-based Subscription

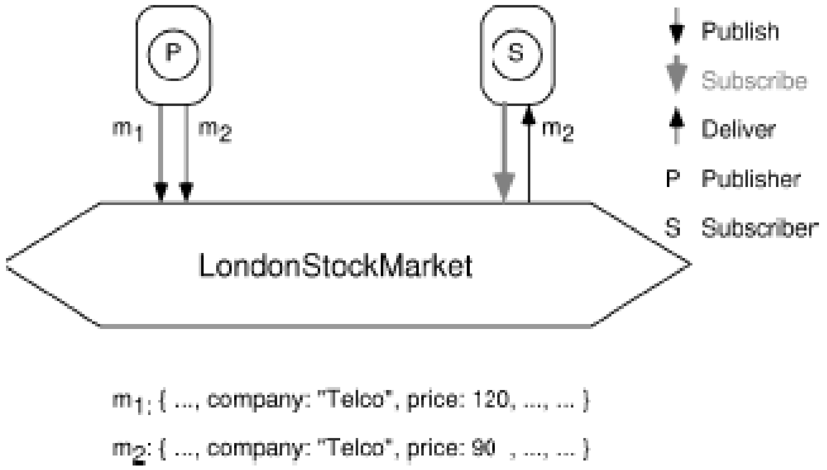

P
S
m1
m2
m2
Publish
Subscribe
Deliver
P Publisher
S Subscriber
LondonStockMarket
m1: { ..., company: "Telco", price: 120, ..., ... }
m2: { ..., company: "Telco", price: 90 , ..., ... }

Fig. 14. Content-based publish/subscribe interactions.

- Added complexity in matching an event to subscriptions. (Implementation: Subscription arranged in a matching tree, where each node is a partial condition.
- However, more precision is provided and event routing is easier

Event Routing

- After filtering the events, the broker/brokers must route the events to the corresponding subscribers.
- Can be done in the following ways:
  - Unicast
  - Multicast
  - Server push/ client pull

Event Routing

- The broker makes the decision: how to route the message to the subscriber.
- Several optimization schemes are available.
  - Profile forwarding scheme – brokers only forward the event to their neighbor broker which fulfill their subscription
  - Filtering the total covering of the subscription of the system – accept publisher events only if a subscriber has subscribed this event.