Naming

Slide set 4
Distributed Systems

Graduate Level

K. N. Toosi Institute of Technology
Dr. H. Khanmirza
h.khanmirza@kntu.ac.ir

Naming

- Names are used
  - to share resources,
  - to uniquely identify entities,
  - to refer to locations, and ⋯.
- Name resolution allows a process to access a named entity
- To resolve names, it is necessary to implement a *naming system* which resolves a name to the entity it refers to
- Naming System can be distributed on multiple machines
  - Method of distributing names on systems?
    - It determines degree of scalability or fault tolerancy.

Names

- Naming Examples
  - Registers → R5
  - Memory → 0xdeadbeef
  - Host names → srini.com
  - User names → sseshan
  - Email → srini@cmu.edu
  - File name → /usr/srini/foo.txt
  - URLs → http://www.srini.com/index.html
  - Ethernet → f8:e4:fb:bf:3d:a6

Naming Systems

- Flat Naming
  - Strings, random numbers, ···
  - Name do not reveal anything about the **location** of the entity
- Structured Naming
  - Names involves several sections like IP address

Flat Name Resolution Methods

- Broadcast
  - ARP in LAN
  - Not Scalable
- Multicast
  - Hosts join multicast groups
  - Groups are identified by multicast address (i.e. special class of IP addresses)
  - Examples
    - Replicated entities are assigned to a multicast group
    - Mobile computers are assigned to a multicast group

Flat Name Resolution Methods

- Forwarding Pointers
  - It is designed for moving objects
  - When object moves from A → B, it leaves a pointer behind which points to the new location
  - Very simple for implementation
  - Issues
    - Forwarding chain can be very long!
    - Needs all intermediate links participate in routing (they should support chaining as long as the object is needed)
    - If one of the intermediate nodes is lost, the object is also lost

Flat Name Resolution Methods

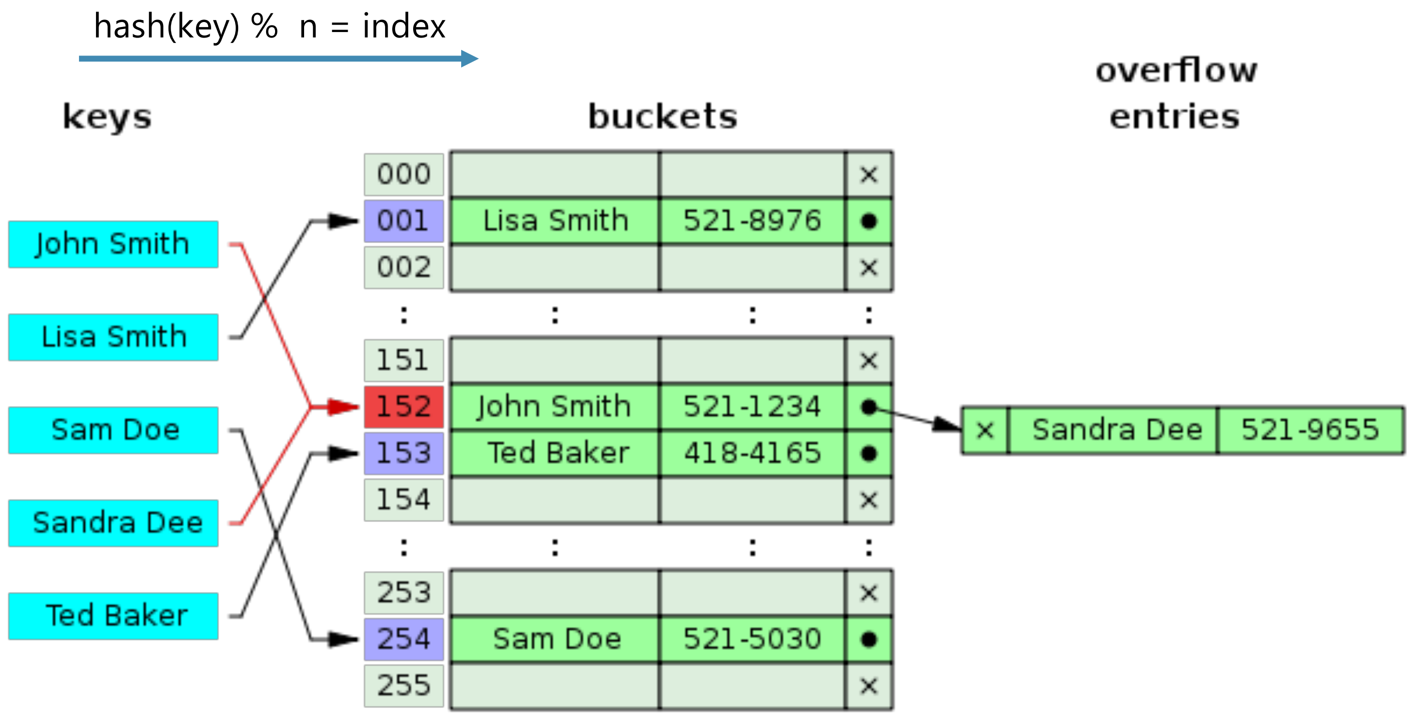
- Home-based approaches
  - Each entity has an ID, registered in fixed location named Home
  - When object is moved to another location, it requests a new temporary address
  - The address is reported back to the home-agent of the mobile object
  - To communicate with object, message is sent to the fixed ID of that object
  - Message is received by home-agent, then it is forwarded to the current location
  - Successive packets are sent to the current location of the object
  - Advantage: everything is hidden from the client
  - Drawback:
    - Needs a fixed home, must be available
    - Distance of current location and the home can impose large delays
  - Example: Mobile Phone System

Flat Name Resolution Methods

- Distributed Hash Tables (DHT)
- Before DHT, we have to study **Consistent Hashing**

Hashtable Data Structure

- Mapping a key of any type to a value of any type
- Sample: key = sum of ASCII codes of characters of a string



Consistent Hashing

- In normal hash tables
  - We have n buckets
  - Objects is mapped to a bucket number with a function:
  - $b = hash(key)\ \%\ n$

- Problem with normal hashing approach
  - Changing number of buckets needs re-mapping of *all* keys ☹

Consistent Hashing

- Example
  - Consider a datacenter with n cache servers storing web page contents
  - Contents are mapped to servers with hash of page address
    - $URL_j \rightarrow server_i \, , i = hash(URL_j) \; mod \; n$
  - Consider adding or failure (of) a cache server

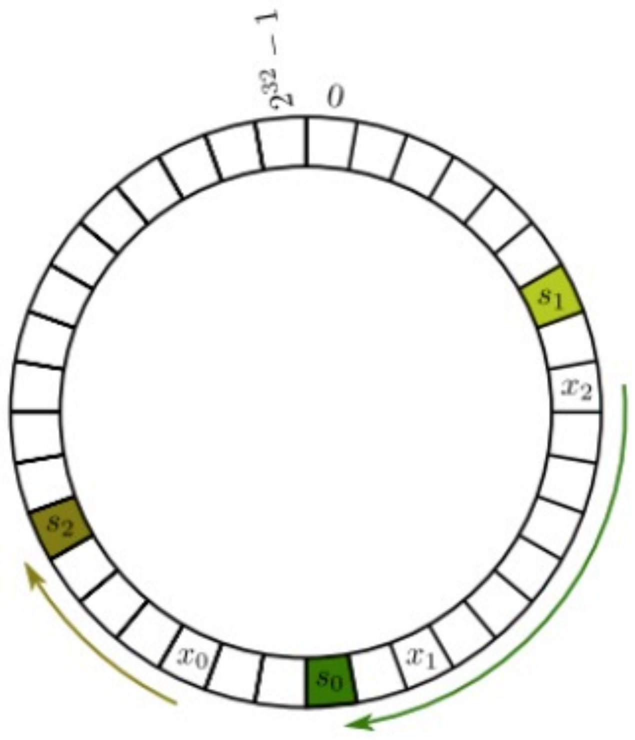
Consistent Hashing

- Having n buckets and k keys, changing the number of buckets needs re-mapping of only $\frac{k}{n}$ keys.

- Main Idea: Map servers to the same ID space of Hash of objects
  - Consider m bits for ID space
  - Assign ID to both servers and objects in range of $[0,2^m-1]$
  - Given an object x, compute h(x)=hash(x)
  - Scan buckets to the right of h(x) until we find a server $s \geq h(x)$
  - Such a system can be visualized as a circle

Consistent Hashing

- We have 3 servers s_0, s_1, s_2
- If $ID(s_1) = S_1$ & $ID(s_2) = S_2$ &$ID(s_0) = S_0$
- $S_1 < S_0 < S_2$
- $hash(obj_1) = x$
- If $S_0 < x \leq S_2 \Rightarrow$ save obj_1 in s_2



Consistent Hashing

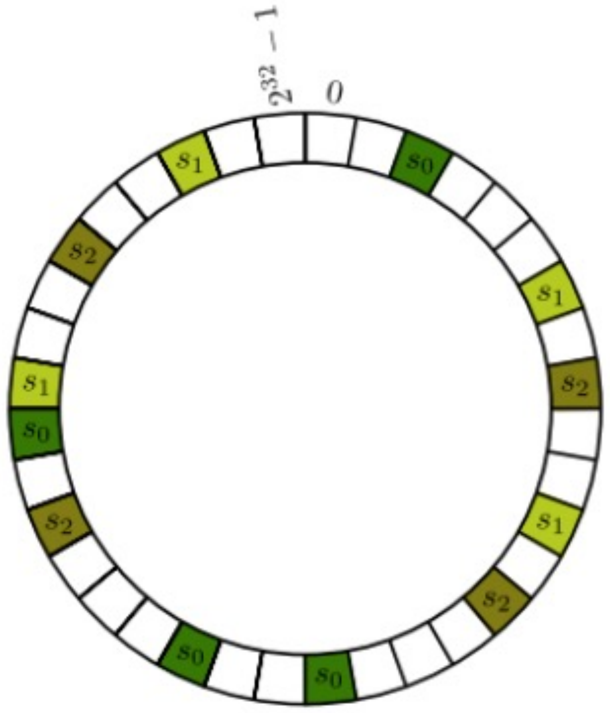
- With a good hash function and hopefully good hashes for objects, keys are distributed evenly, each server receives k/n of keys
- If a new server is added only k/n of keys that must be stored in the new server and need to be moved
- If a server is removed, only k/n of keys must be distributed to other nodes

Consistent Hashing

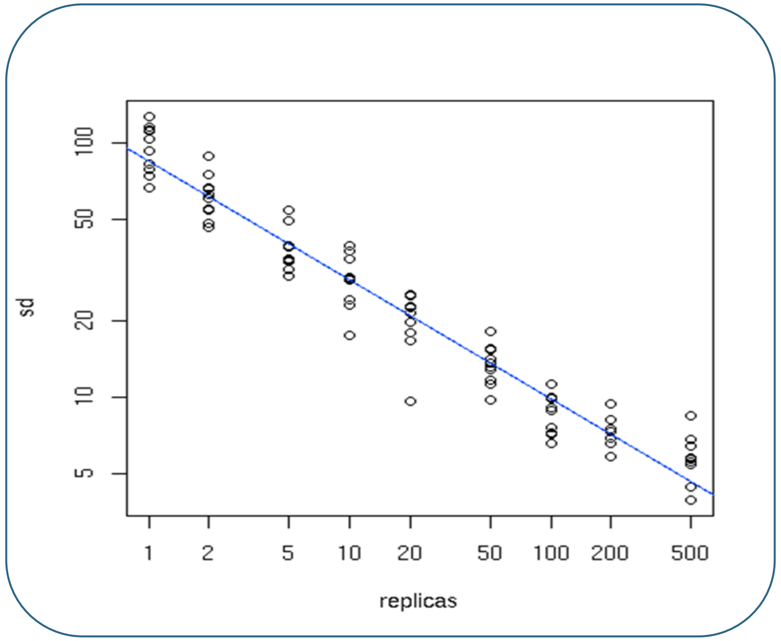
- What if servers receive imbalanced number of keys?
- Create 1-to-k mapping between machines and hash nodes
  - Assign k IDs to each server!
    - Multiple hash functions with the same mapping range
    - Multiple names for a server and one hash function: S1 is added with names: S1_0, S1_1, S1_2,
  - Add k "virtual nodes" to the circle for each server

- Do the searching and adding keys just as before
  - Adding object → hash the key → decide which server must keep the key as before!

- This approach reduces the variance in server loads, significantly



Consistent Hashing



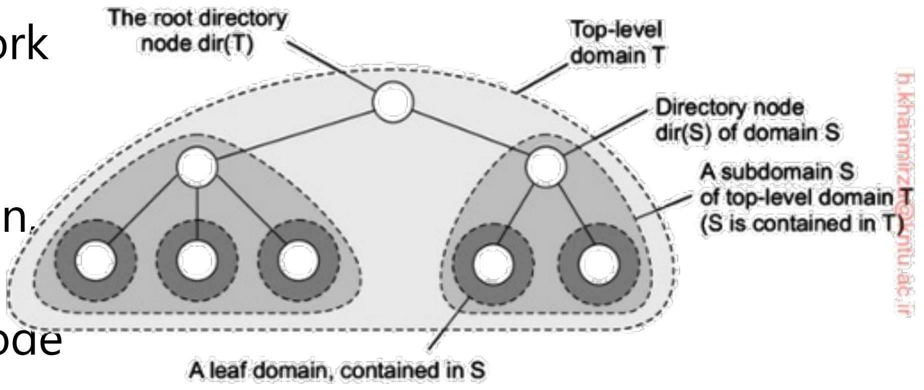
Standard deviation of number of mapped keys to servers

Consistent Hashing

- Consistent Hashing is used in several practical projects
  - DHT (Distributed Hash Table)
    - Consistent Hashing is a way of implementing DHT
  - Chord (P2P Network)
  - Amazon Dynamo (NoSql DB)
  - Cassandra (NoSql DB)
  - MemCached
  - …

Hierarchical Approaches

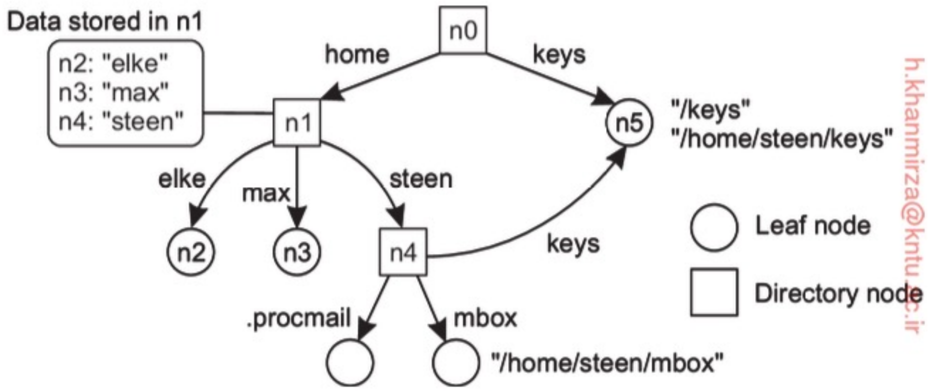
- A network is divided in to Domains
  - a top-level domain spans the entire network
  - The lowest level domain is called Leaf domain
  - The directory node of the top-level domain called the root (directory)
- Each domain D has an associated directory node dir(D) that keeps track of the entities in that domain.


The root directory node dir(T)
Top-level domain T
Directory node dir(S) of domain S
A subdomain S of top-level domain T (S is contained in T)
A leaf domain, contained in S

- Entities are located in directory nodes using location records
  - The actual entity location is kept in leaf domains
  - In higher domains location record is just a pointer
  - An entity may have several addresses (it can be reached from different domains)
- Look-up process
  - Client issues lookup request to the leaf domain directory node
  - If does not find it, passes it to the parent domain
- Insertion and deletion process works like lookup

Structured Naming

- Composed from simple, human-readable names like File system
- Names are commonly organized into a namespace
- Namespaces represented as a labeled, directed graph with two types of nodes
  - Leaf nodes: the entity (file, host,···)
  - Directory Node: contains one incoming but several outgoing edges
    - Keeps a directory table
  - Root node has no incoming edge
  - Path name: sequence of edge labels from a node
    - N3=N0:[home,max]
- Relative vs Absolute paths

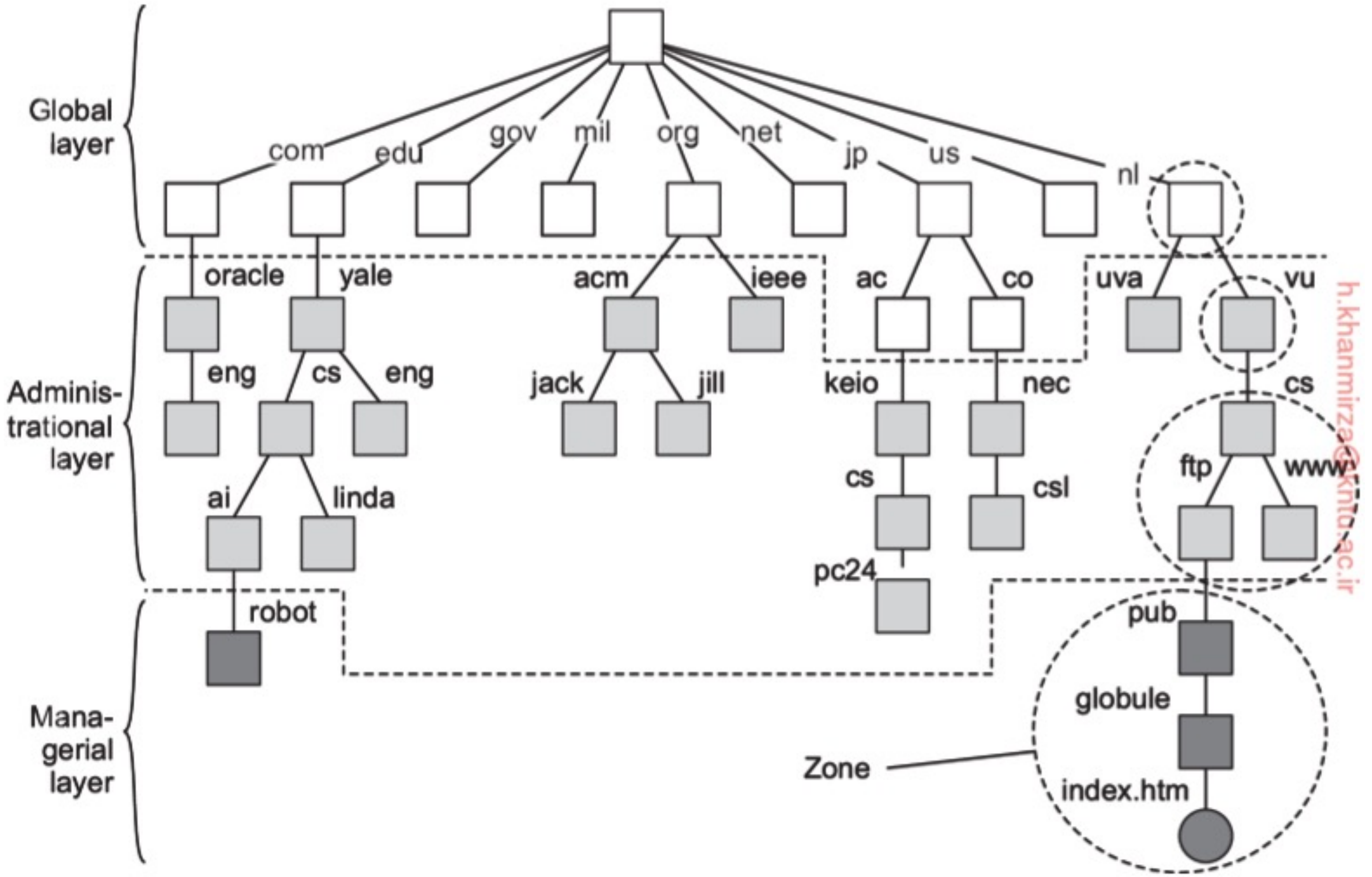


Distributing Namespace

- Generally composed of three layers
  - Global layer
    - highest-level nodes with very rare change in their directory table
  - Administrational layer
    - Directory nodes that together are managed within a single organization
    - Changes are more than global layer, but still few
  - Managerial layer
    - Have regular changes
    - Managed by users not admins of a system
    - Consists
      - Hosts in a LAN
      - User-defined files or directories

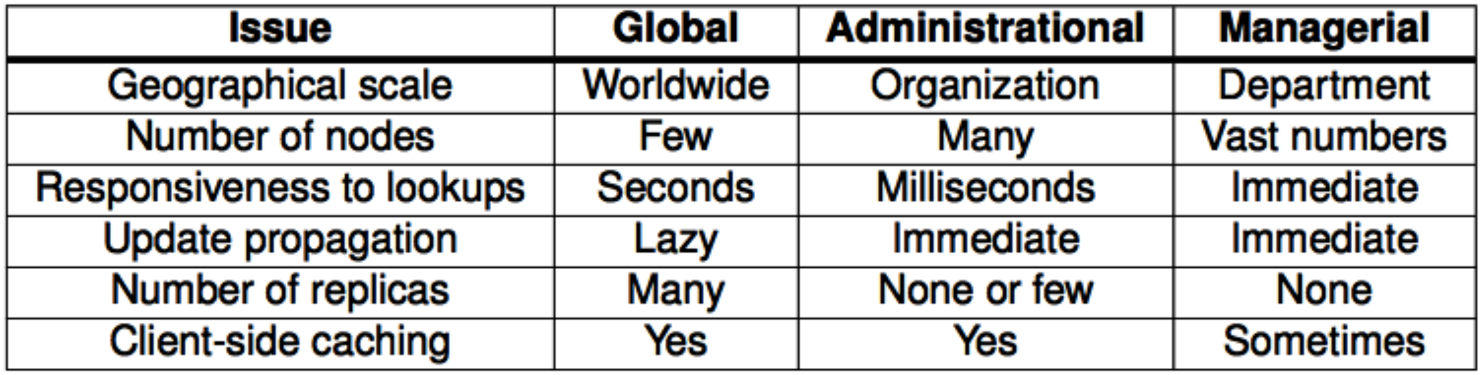
- Real-world Example: DNS

Distributing Namespace



Distributing Namespace

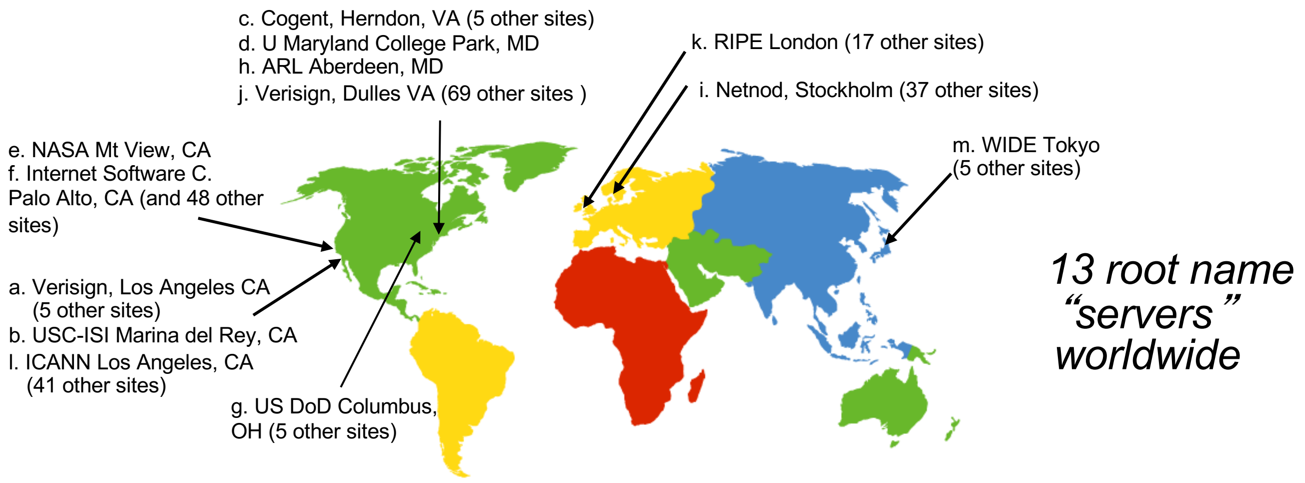
- Requirements of name servers in each layer



Issue	Global	Administrational	Managerial
Geographical scale	Worldwide	Organization	Department
Number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Client-side caching	Yes	Yes	Sometimes

DNS

- Root name servers
  - Root DNS servers are labeled from A through M (13 main sites, they are not single servers)
  - They all keep the same data (replicated data)


c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
h. ARL Aberdeen, MD
j. Verisign, Dulles VA (69 other sites)
k. RIPE London (17 other sites)
i. Netnod, Stockholm (37 other sites)
e. NASA Mt View, CA
f. Internet Software C. Palo Alto, CA (and 48 other sites)
m. WIDE Tokyo (5 other sites)
a. Verisign, Los Angeles CA (5 other sites)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA (41 other sites)
g. US DoD Columbus, OH (5 other sites)
13 root name "servers" worldwide

DNS

- Top-level domain (TLD) servers:
  - Responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
    - Verisign Global Registry Services maintains servers for .com TLD
    - Educause for .edu TLD

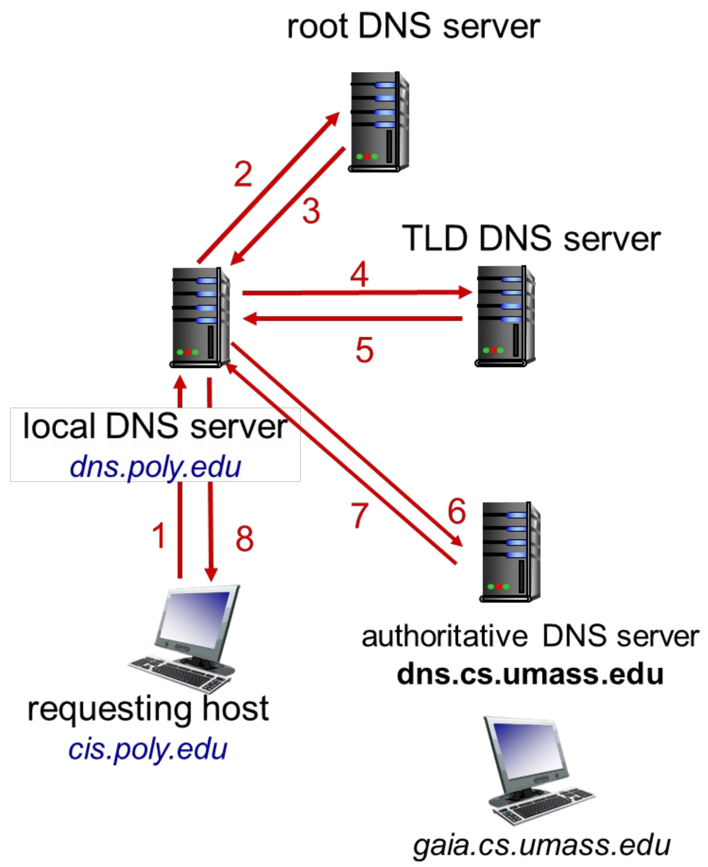
- Authoritative DNS servers:
  - Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization' s named hosts
  - It is maintained by organization or service provider

DNS

- Local DNS name server
  - Does not strictly belong to hierarchy
  - Each ISP (residential ISP, company, university) has one
  - When host makes DNS query, query is sent to its local DNS server
    - Has local cache of recent name-to-address translation pairs
    - Acts as proxy, forwards query into hierarchy

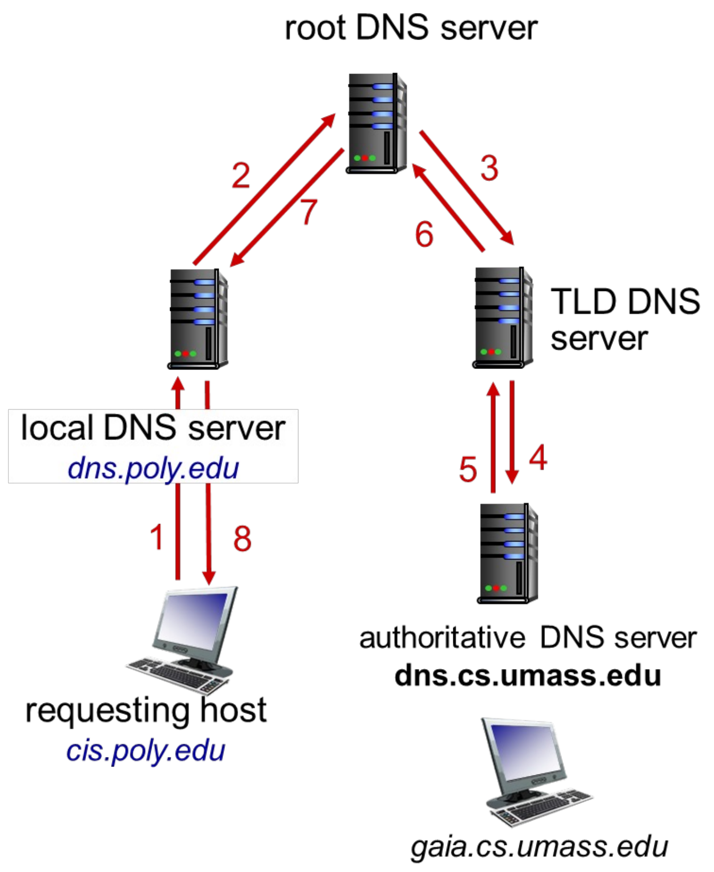
Name Resolution Methods

- Iterative Name Resolution



Name Resolution Methods

- Recursive Name Resolution



- Comparison
  - Recursive model
    - More efficient from the bandwidth view
    - Simpler implementation
    - Caching is efficient
      - If A requests a name resolution it is cached in A, only.
  - Iterative model
    - Less pressure on DNS servers

Attribute-Based

- Search an entity using its properties
  - LDAP
  - Microsoft Active Directory
- Read yourself!

Case Studies

CDN: Content Distribution Networks

CDN

- Typical Workload of Web (= Web Pages)
  - Multiple (typically small) objects per page
  - File sizes are heavy-tailed (files with small sizes are more)
  - This ruins the performance. Why?
    - Lots of small objects & TCP: 3-way handshake, Lots of slow starts, connection states, ...

- Popular websites have millions of simultaneous users from all over the world,
  - Is it enough to have large datacenters in USA?
  - Video Streaming has been popular, How we can stream video with high quality to millions of the users throughout the globe?

CDN

- How distribute contents around the globe without delay?
  - esp. Streaming video

- How distribute popular contents to millions of people
  - Real-time videos (i.e. a football match)

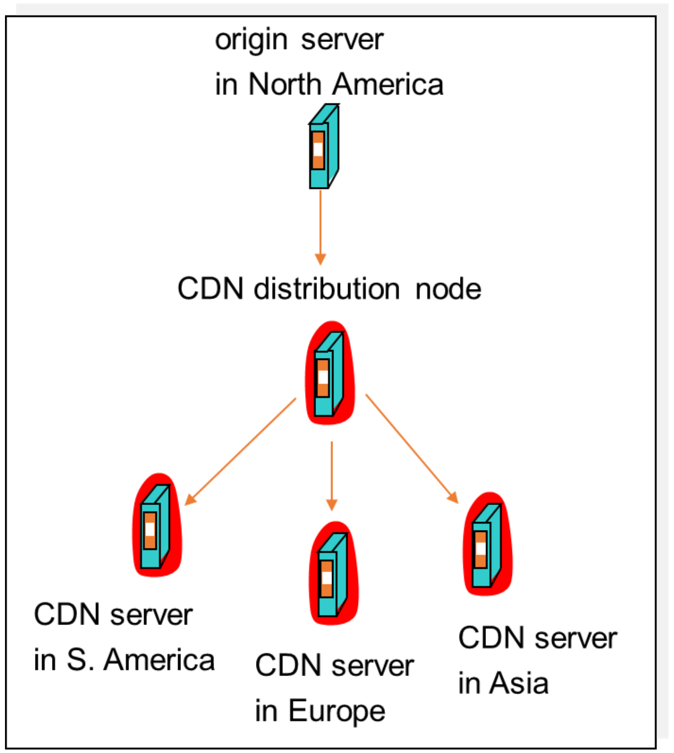
CDN

- CDN company installs hundreds of content servers throughout Internet
  - Geographically close to users

- CDN replicates content in CDN servers
  - Content providers are the CDN customers
  - When provider updates content, CDN updates servers

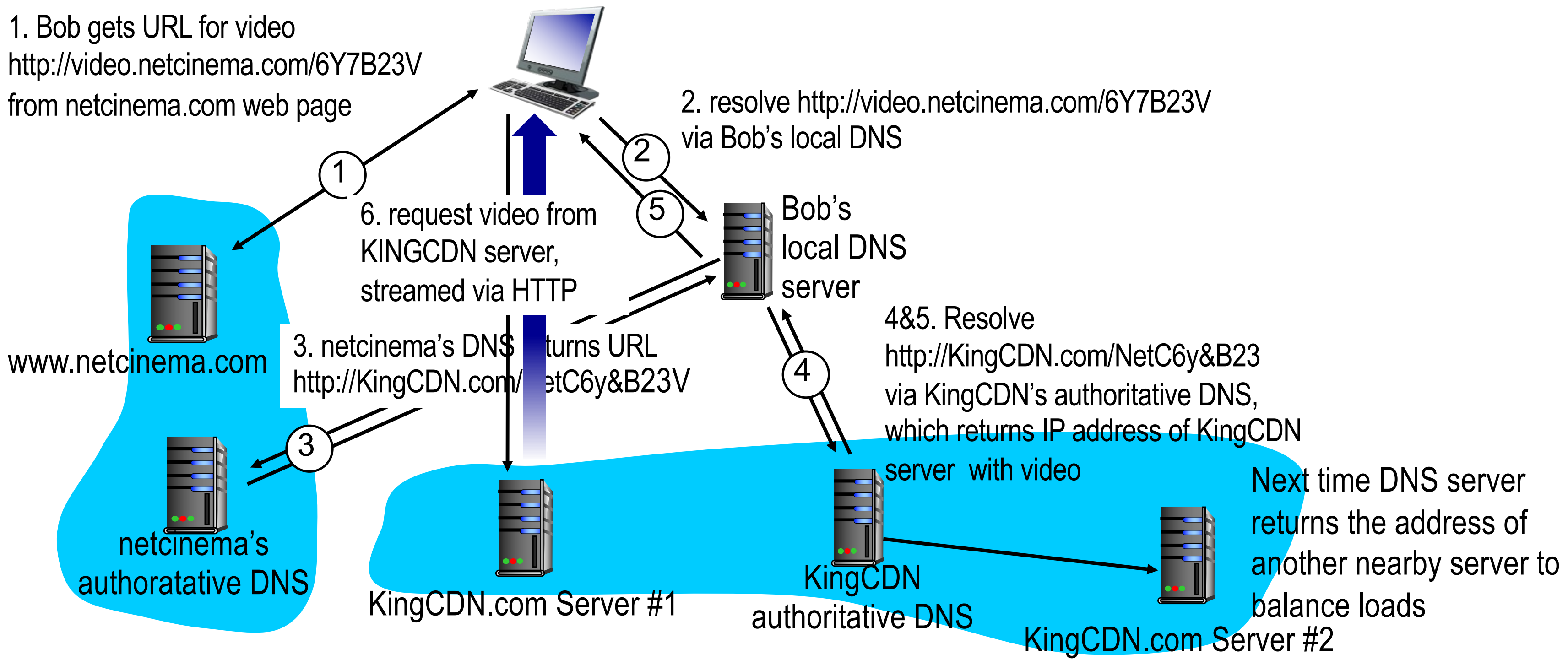
- Subscriber requests content from CDN
  - Directed to nearby copy, retrieves content
  - May choose different copy if network path congested

- One of the first commercial CDN networks is Akamai a company based in Japan



Content Access Scenario

- Client requests video http://video.netcinema.com/6Y7B23V
  - video stored in CDN at http://KingCDN.com/NetC6y&B23V


1. Bob gets URL for video http://video.netcinema.com/6Y7B23V from netcinema.com web page
1
2. resolve http://video.netcinema.com/6Y7B23V via Bob's local DNS
2
5
6. request video from KINGCDN server, streamed via HTTP
Bob's local DNS server
www.netcinema.com
3. netcinema's DNS returns URL http://KingCDN.com/NetC6y&B23V
3
4
4&5. Resolve http://KingCDN.com/NetC6y&B23 via KingCDN's authoritative DNS, which returns IP address of KingCDN server with video
netcinema's authoratative DNS
KingCDN.com Server #1
KingCDN authoritative DNS
Next time DNS server returns the address of another nearby server to balance loads
KingCDN.com Server #2

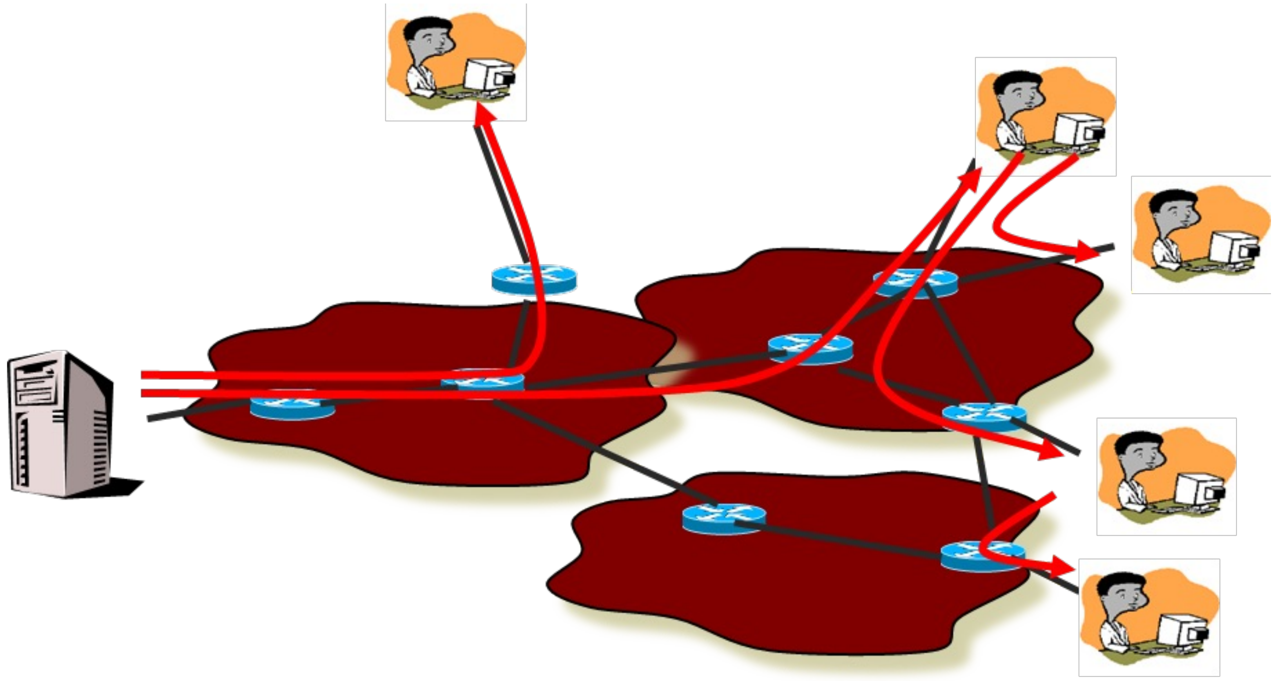
CDN

- CDN research challenges
  - How to replicate content
  - Where to replicate content
  - How to find replicated content
  - How to choose among known replicas
  - How to direct clients towards replica (we’ve answered already)

Peer-to-Peer Systems

Peer-To-Peer Networks (P2P)

- The aim is to deliver a service that is fully decentralized and self-organizing, dynamically balancing the storage and processing loads between all the participating computers as computers join and leave the service.



Peer-to-Peer Networks

- Typically each member stores/provides access to some content
- Basically a replication system for files
  - A tradeoff between possible location of files and searching difficulty
  - Peer-to-peer allow files to be anywhere, then searching is the challenge
  - Dynamic member list makes it more difficult

P2P characteristics

- All users must contribute resources to the system
- All nodes have the same functional capabilities and responsibilities
- Correct operation of the system does not depend on the existence of any centrally administered systems
- They may offer a limited degree of anonymity to the providers and users of resources.

P2P characteristics

- Efficiency
- Choice of an algorithm for placement of data across many hosts and access to it
  - Needs a method for balancing the workload
  - Needs a method for ensuring availability without adding excessive overheads.

Searching

- Needles in Haystacks
  - Searching for top 40, or an obscure track from 1981 that nobody’s heard of?
- Search expressiveness
  - Whole word? Regular expressions? File names? Attributes? Whole-text search
- Common Primitives
  - Join: how to begin participating?
  - Publish: how do advertise owing file?
  - Search: how to find a file?
  - Fetch: how to retrieve a file?

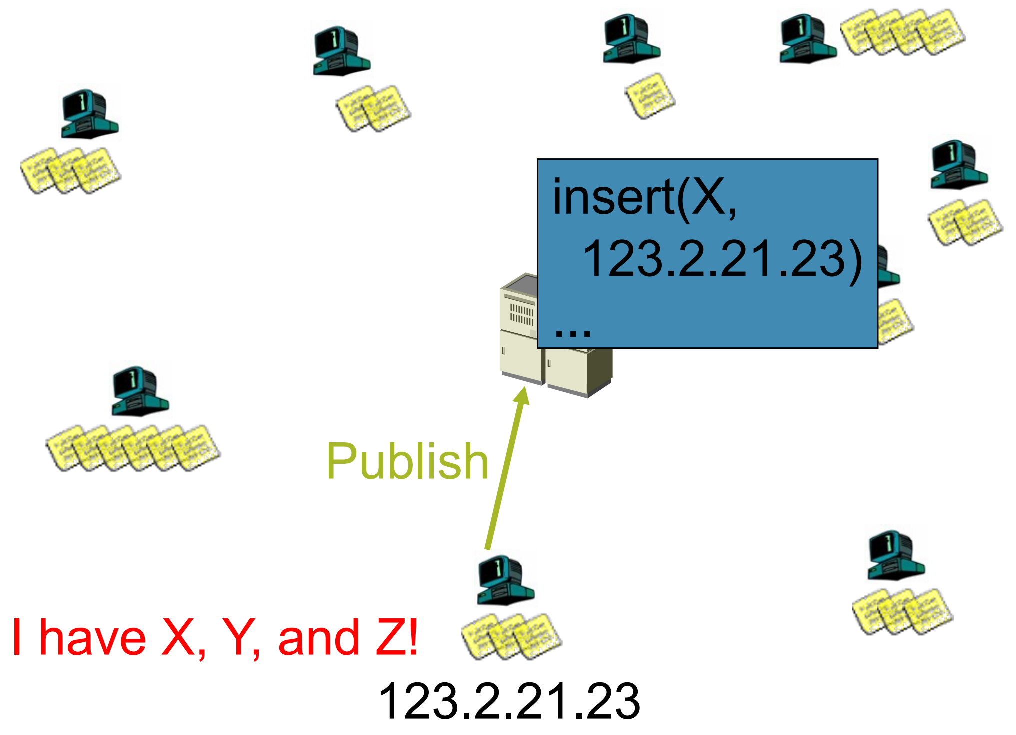
Generations

- First: initiated by Napster, a music sharing platform
- Second: having greater scalability, anonymity and fault tolerance like Gnutella, Kazaa, BitTorrent
- Third:
  - Characterized by the emergence of middleware layers for the application-independent management of distributed resources on a global scale.
  - Provides guarantees of delivery for requests in a bounded number of network hops
  - Exmaples: Chord, Pastry, Tapestry, CAN

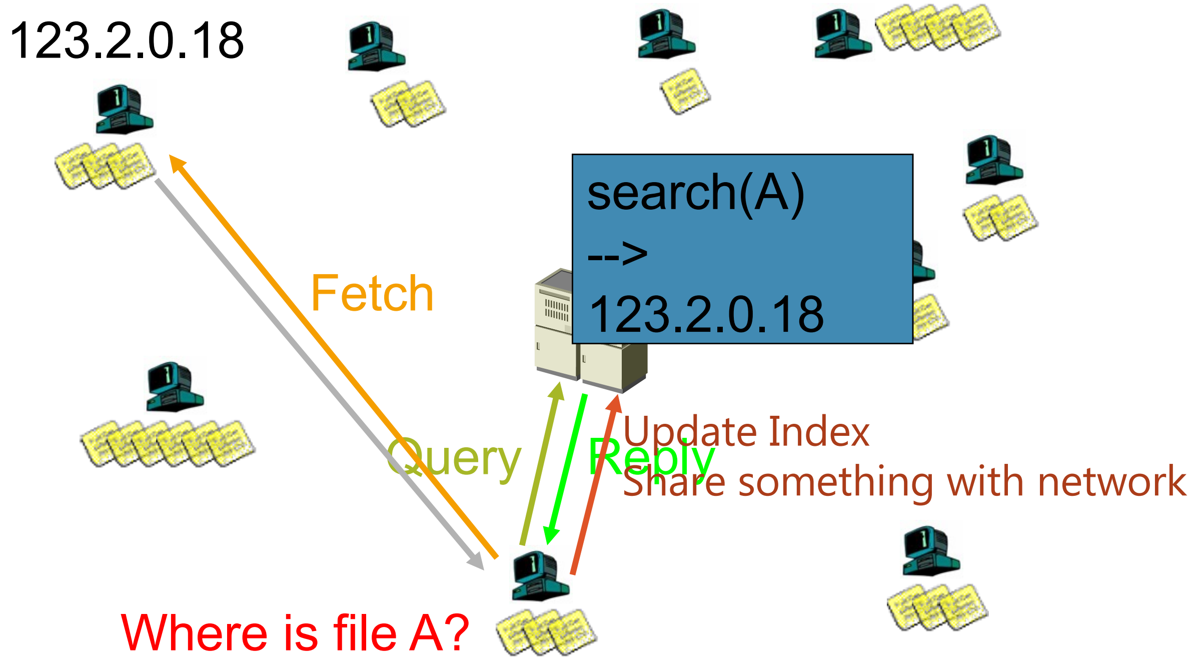
Napster

- A music sharing system
- Centralized Database:
  - Join: on startup, client contacts central server
  - Publish: reports list of files to central server
  - Search: query the server => return someone that stores the requested file
  - Fetch: get the file directly from peer

Publish



Search



search(A)
-->
123.2.0.18

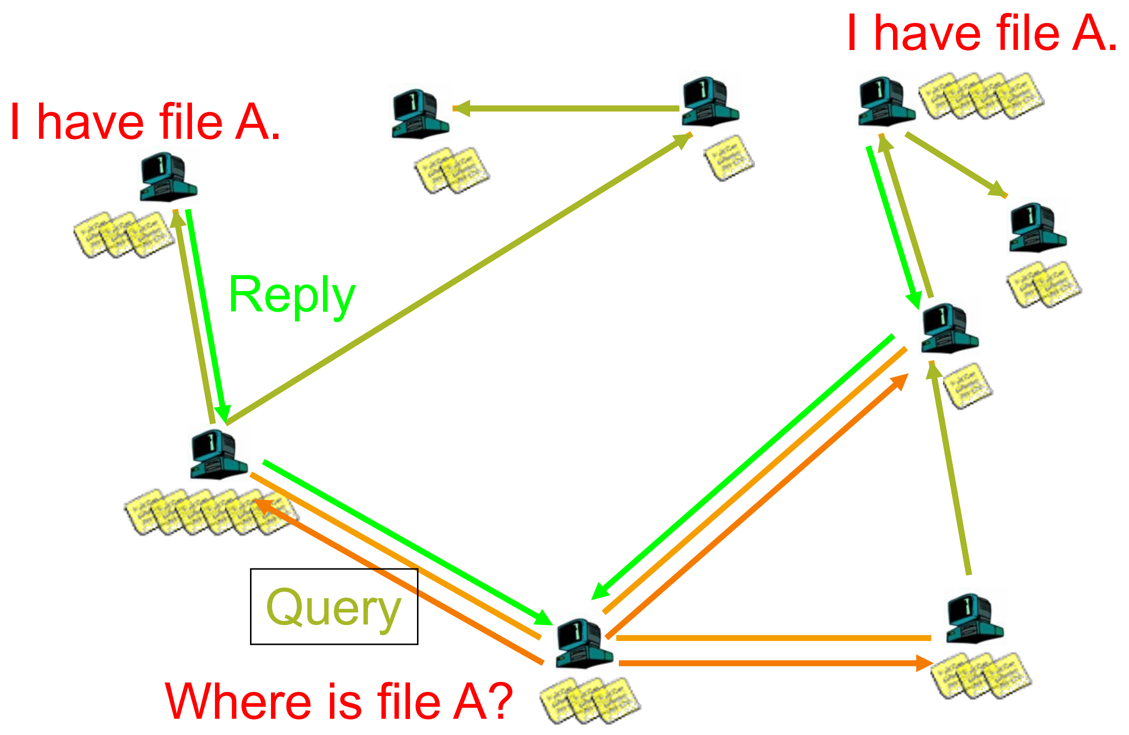
Napster

- Pros
  - Simple
  - Search scope is O(1)
  - Replicated index servers
  - Taking locality into account when choosing the file location for a client
    - No. of hops between client and server
    - Provides load distribution
- Cons:
  - Server maintains O(N) State
  - Server does all processing

"Old" Gnutella

- Query Flooding:
  - Join: on startup, client contacts a few other nodes; these become its "neighbors"
  - Publish: no need
  - Search: ask neighbors, who ask their neighbors, and so on... when/if found, reply to sender.
    - TTL limits propagation
  - Fetch: get the file directly from peer

Old Gnutella Search



Old Gnutella

- Pros:
  - Fully de-centralized
  - Search cost distributed
  - Processing at each node permits powerful search semantics
- Cons:
  - Search scope is O(N)
  - Search time is O(???)
  - A non-popular file may not be found, forever!
  - Nodes leave often, network unstable
- TTL-limited search
  - For scalability, does NOT search every node. May have to re-issue query later

Kazaa

- Modifies the Gnutella protocol into two-level hierarchy
  - Hybrid of Gnutella and Napster
  - Super-nodes
    - Nodes that have better connection to Internet
    - Act as temporary indexing servers for other nodes
    - Help improve the stability of the network
  - Standard nodes
    - Connect to super-nodes and report list of files
    - Allows slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search across super-nodes
- Disadvantages
  - Kept a centralized registration → allowed for law suits ☹

BitTorrent

- Bram Cohen, Incentives Build Robustness in BitTorrent , 2003
- The principal design feature is splitting of files into fixed-sized chunks
- Clients can download a number of chunks in parallel from different sites.

Terminology

- Seeder**:** A peer that holds a complete copy of a file (consisting of all its chunks)

- Leecher: A peer involved in downloading a file and holds only a portion of its chunks

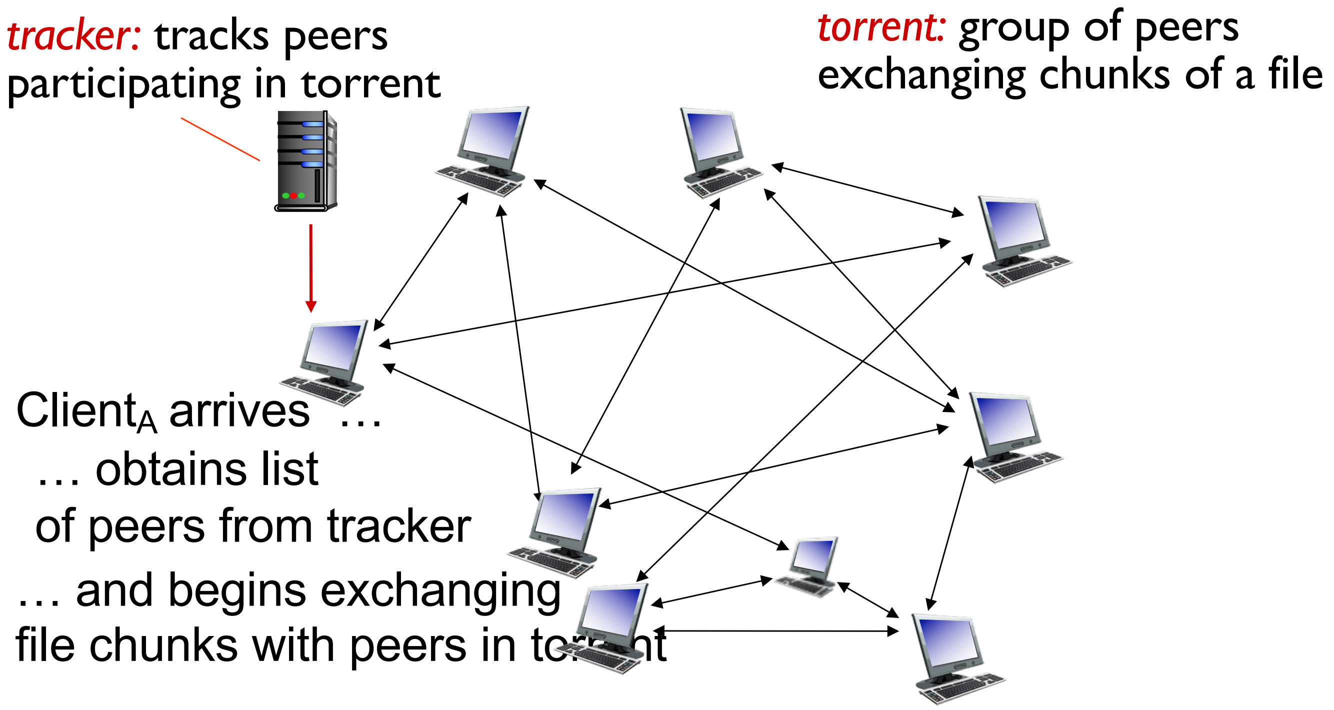
- .torrent file: A file that maintains metadata about an available file
  - Files are typically published on websites or $\cdots$, and registered with at least one tracker

Terminology

- Tracker
  - Responsible for keeping track of download status of a particular file
  - Trackers should be obtained from google!! or email
  - Tracker server keeps track of
    - Where file copies reside on peer machines
    - Which ones are available at time of the client request
    - Helps coordinate efficient transmission and reassembly of the copied file
- Clients communicate with a tracker periodically to negotiate faster file transfer with new peers, and provide network performance statistics
  - After the initial download, peer-to-peer communication can continue without the connection to a tracker
  - Tracker-less torrent networks use DHT

BitTorrent

- A file is divided into 256Kb chunks
- Peers in torrent send/receive file chunks

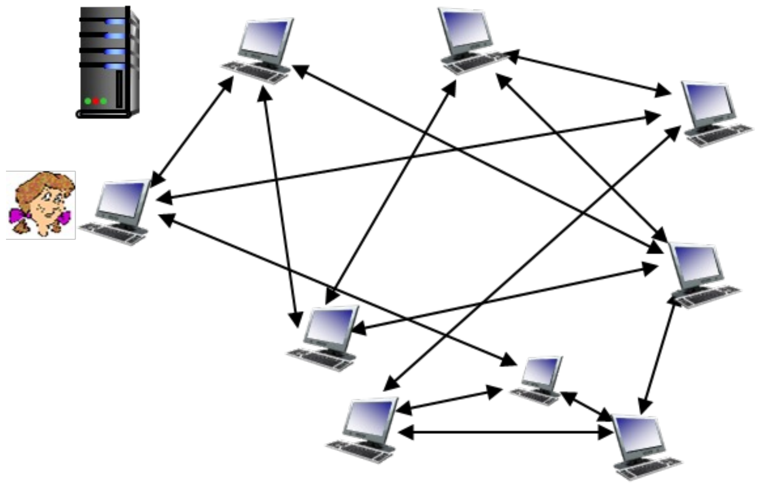

tracker: tracks peers participating in torrent
torrent: group of peers exchanging chunks of a file
Client_A arrives …
… obtains list of peers from tracker
… and begins exchanging file chunks with peers in torrent

BitTorrent

- Peer joining torrent:
  - Has no chunks, but will accumulate them over time from other peers
  - Registers with tracker to get list of peers, connects to subset of peers (“neighbors”)

- While downloading, peer uploads chunks to other peers
  - Peer may change peers with whom it exchanges chunks
  - Once peer has the entire file, it may (selfishly) lea   or remain in torrent

- Churn: peers may come and go



BitTorrent

- Requesting chunks
  - At any given time, different peers have different subsets of file chunks
  - Periodically, client asks each peer for list of chunks that they have
  - Client requests missing chunks from peers, rarest first

- Tit-for-Tat Mechanism
  - A client sends chunks to those 4 peers currently sending chunks at highest rate
    - Other peers are choked by the client (do not receive chunks from this client)
    - Re-evaluate top 4 every 10 secs
  - Every 30 secs: randomly select another peer, start sending chunks
    - "Optimistically unchoke" this peer
    - Newly chosen peer may join top 4

BitTorrent

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Central tracker server is needed to bootstrap the swarm
    - Out-of-band searching
    - Alternate tracker designs exist (e.g. DHT based)
  - Recent Seeders can leave after completing their needs

Chord

- Stoica, A scalable peer-to-peer lookup service for internet applications, 2001
- A tracker-less P2P
- Uses a modified version of DHT
  - Assigns a randomly-chosen m-bit id for nodes (m=128 to 160 bit)
  - Keys to objects are assigned from the same m-bit space, generated by a hash function

Chord

- An object with key k is assigned to a node with the first id greater than k $(id \geq k)$
  - Node is called Successor of k $(succ(k) = id)$

- Problem: Having key k how to find $succ(k)$, efficiently?

- Simple Approach:
  - Each node p knows its successor and predecessor
  - When receives a key, if it does not have the object forwards the request to its (overlay) neighbor
  - In the worst case this is a linear search O(n), very inefficient for large networks

Chord

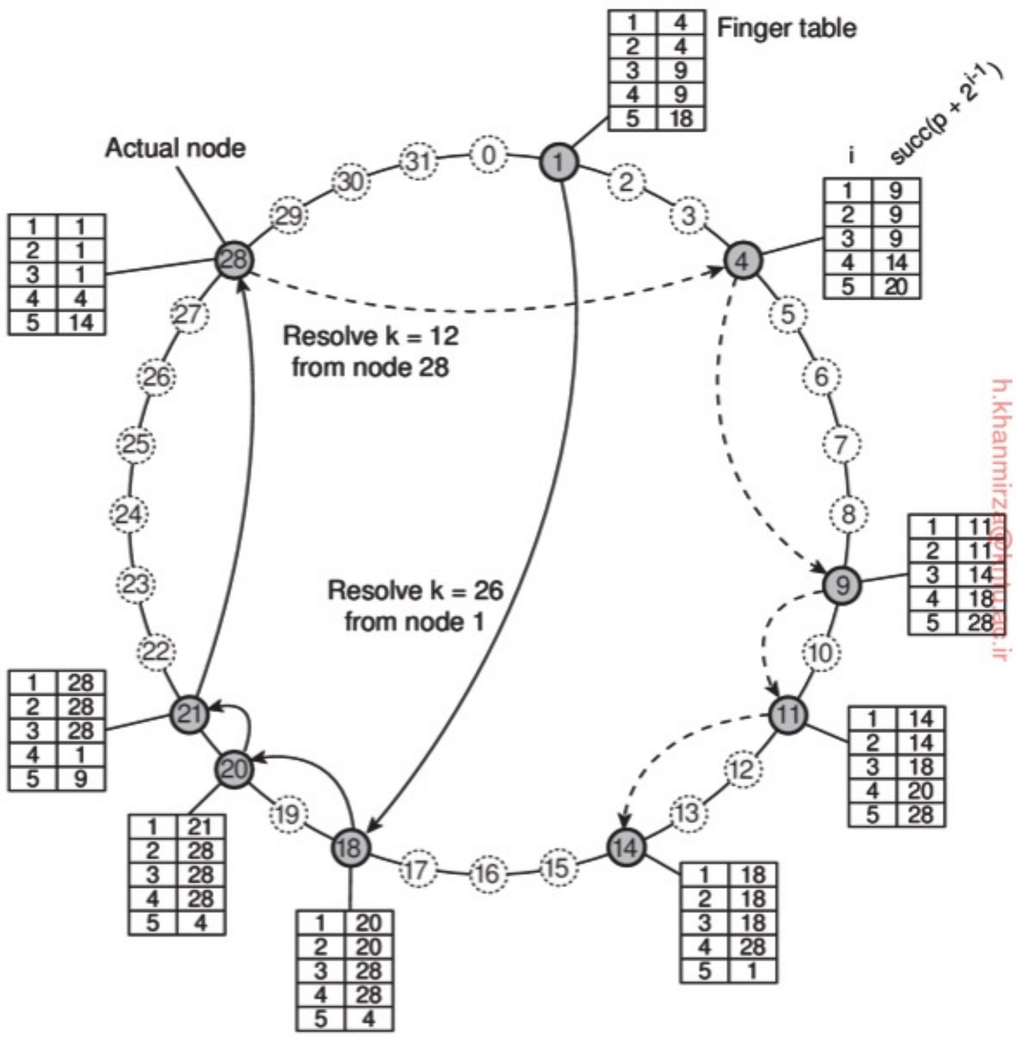
- Having key k how to find $succ(k)$, *more* efficiently?
  - Use shortcut links in the circle overlay!

Chord

- Each node has a Finger Table (FT) with $s \leq m$ rows
  - $FT_p[i] = succ(p + 2^{i-1})$
    - i th row of the table for the node p have the address of node with the computed id
    - $FT_p[1] = succ(p),$
    - $FT_p[2] = succ(p + 2)$
    - $FT_p[3] = succ(p + 4)$
  - When node p receives a query k sends it to $q = FT_p[j] \leq k < FT_p[j + 1]$
    - The highest finger table *id* which is smaller than *k*
  - Needs $O(logN)$ steps to find the appropriate node

Chord

- Finger Table Size = 5
- Resolving $k = 26$ from node 1
- Node1 send to $FT_1[5] = 18$
- Node18 send to $FT_{18}[2] = 20$
- Node20 send to $FT_{20}[1] = 21$
- Node21 send to $FT_{21}[1] = 28$



1	4
2	4
3	9
4	9
5	18

1	9
2	9
3	9
4	14
5	20

1	11
2	11
3	14
4	18
5	28

1	14
2	14
3	18
4	20
5	28

1	18
2	18
3	18
4	28
5	1

1	20
2	20
3	28
4	28
5	4

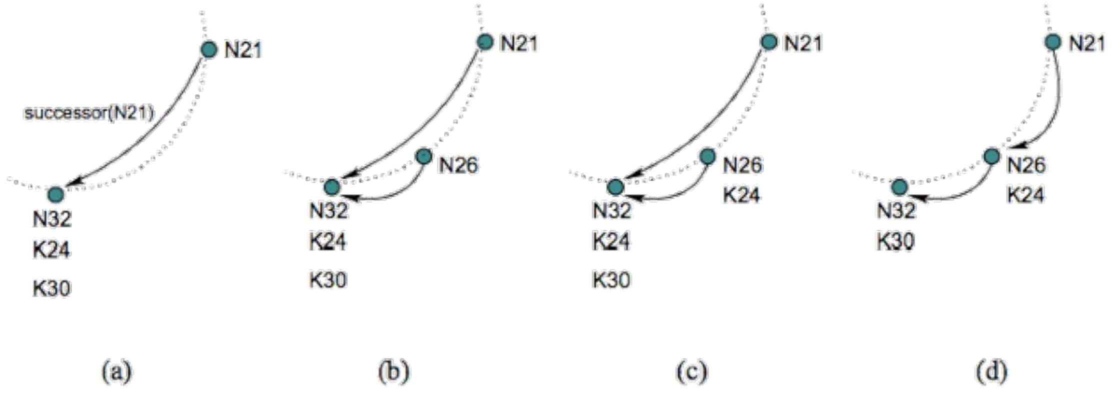
1	21
2	28
3	28
4	28
5	4

1	28
2	28
3	28
4	1
5	9

1	1
2	1
3	1
4	4
5	14

Chord

- Nodes may fail or join/leave, how to keep FT up-to-date?
- Joining:
  - Node is assigned an ID, say p
  - Resolves $succ(p+1)$
    - This means: If we had an object with key=p+1 who is the responsible
  - After finding, it inserts it self between pred(p+1) & p+1
  - Copies all keys from $succ(p+1)$ in range (p-1,p]
- Leaving
  - A similar approach



Chord

- Keeping the FT up-to-date, Stabilizing procedure
  - Periodically, p issues a request to *succ(p+1)* to resolve id=pred(succ(p+1))
    - If id=p, OK
    - If no, some new nodes have been joined in between
    - Sends the same request to the id
  - This periodic request is made for all rows of the table
  - Node *p* also checks the aliveness of *pred(p)*
    - If finds it failed, sets the pred(p)=unknown and waits for the periodic stabilizing procedure

Chord

- Other optimizations
  - Topology-based Id assignment
    - Geographically close nodes get closer ids
  - Proximity Routing (Proximity = Closeness)
    - Having multiple destinations for each FT entry
    - Leaving of a nodes does not immediately lead to look-up failure
    - Can choose the best DEST for a query

Chord

- Structured Overlay Routing (like Chord)
  - Join: On startup, contact a "bootstrap" node and integrate yourself into the distributed data structure; get a node id
  - Publish: Route publication for file id toward a close node id along the data structure
  - Search: Route a query for file id toward a close node id. Data structure guarantees that query will meet the publication.
  - Fetch: Two options:
    - Publication contains actual file ➔ fetch from where query stops
    - Publication says "I have file X" ➔ query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3

Chord

- Routing time O(log N) hops, N=rows of finger tables
- It provides a guaranteed lookup time/steps
- Finding files with complex queries is a problem
  - File starting with “string”
  - File name with “string1” + “string2”

Pastry

- Rowstron, Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems”, 2001
- Another DHT implementation
- Assign 128-bit SHA-1–generated GUIDs to nodes and objects
- For routing, these IDs are treated of as sequences of 2^b
  - For b=4, GUIDs are viewed as Hex strings
- IDs are arranged in the same circular structure
- Content is copied in k nodes with the closest GUID to the content GUID, numerically.

Pastry

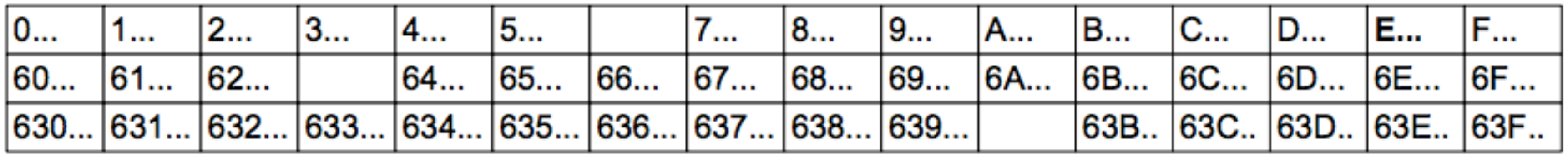
- Each nodes keeps three tables:
  - Table of leaf nodes with length l, $\frac{l}{2}$ nodes numerically before and after its own GUID

  - A Routing Table
    - Table has $\lceil \log_{2^b} N \rceil$ rows (for b=4, $\log_{16} N$) and 2^b columns (for b=4, 16 cols)
    - Row j of table (starting j=0) refers to a node which has j −digit common prefix with the current node GUID

  - Each node maintains information about nodes that are close in terms of network physical locality
    - Based on RTT or distance based on hops

Pastry

(1) Routing table of node 63AB



0...	1...	2...	3...	4...	5...		7...	8...	9...	A...	B...	C...	D...	**E...**	F...
60...	61...	62...		64...	65...	66...	67...	68...	69...	6A...	6B...	6C...	6D...	6E...	6F...
630...	631...	632...	633...	634...	635...	636...	637...	638...	639...		63B..	63C..	63D..	63E..	63F..

- If N = 10000 (total nodes in the network) and b=4
- $[\log_{16} 10000] = 4$ rows in routing table

Pastry

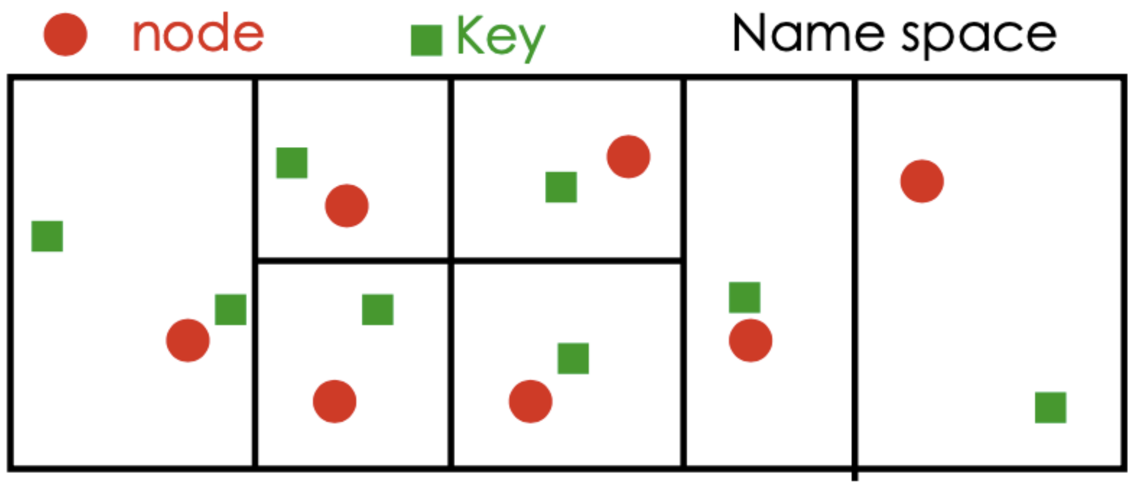
- Routing Algorithm
  1. Given a message with its key, the node first checks its leaf set
     - If there is a leaf node whose ID is closest to the key, the message is forwarded directly to the node
  2. Find a node with the longest common prefix with query ID
  3. Send query to a node in Leaf-set or routing table or neighbor-set which is numerically closest to the query
     - This condition occurs rarely (due to failure, $\cdots$)
- Search takes $O(\log_{2^b} N)$ steps

Pastry

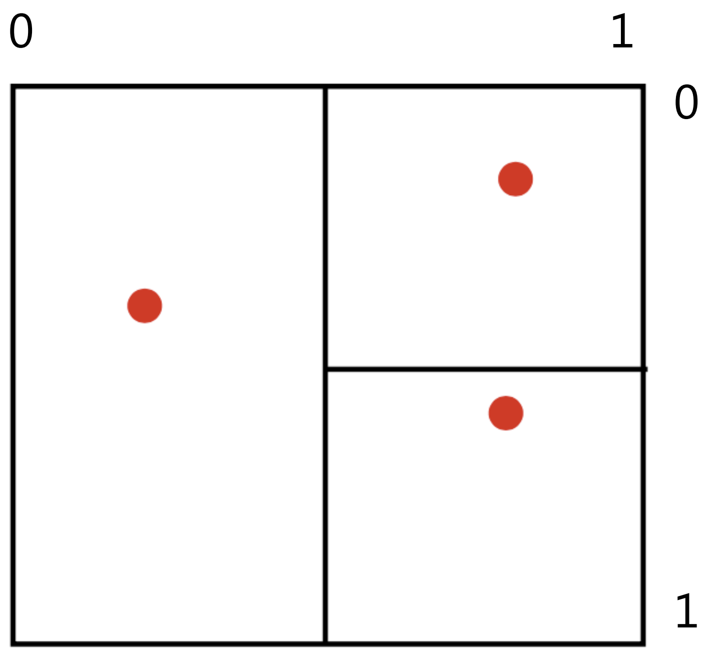
- Join mechanism
  - Joining node must know of at least a node already in the system
  - Generates an ID for itself, and sends a join request to the known node
  - The request will be routed to the node whose ID is numerically closest to the new node ID
  - All the nodes encountered on route to the destination will send their state tables (routing table, leaf set, and neighborhood set) to the new node
  - The new node will initialize its own state tables, and it will inform appropriate nodes of its presence
  - Leaf-set and routing table is maintained by sending keep-alive messages

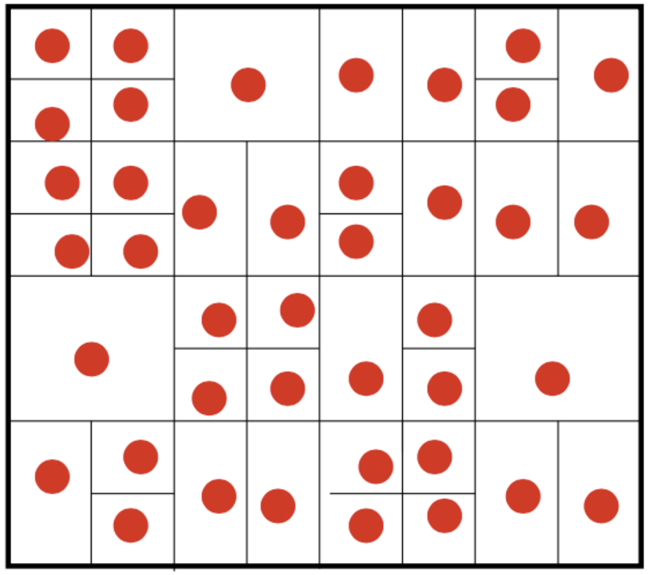
CAN

- Ratnasamy, “A Scalable Content-Addressable Network”, 2001
- Distribute nodes and objects in a d-dimension space



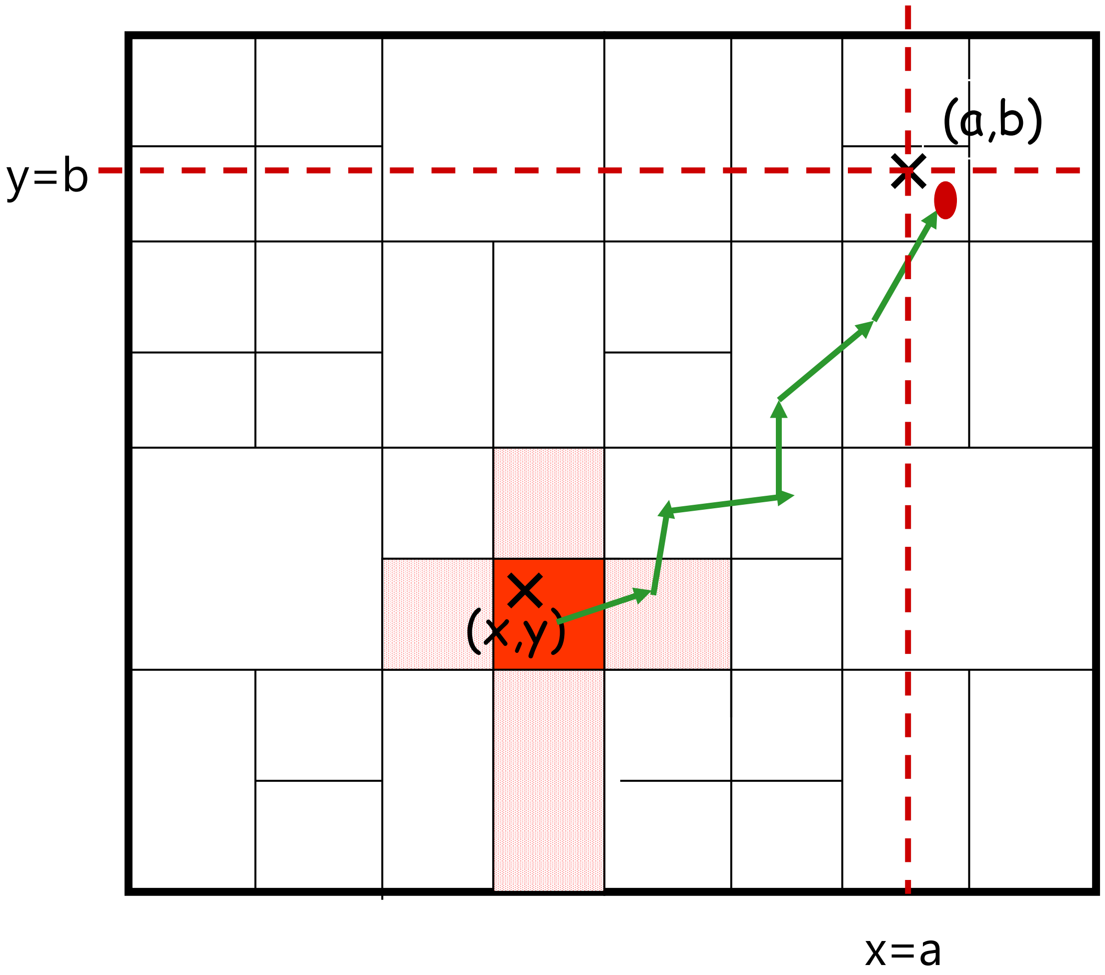
CAN





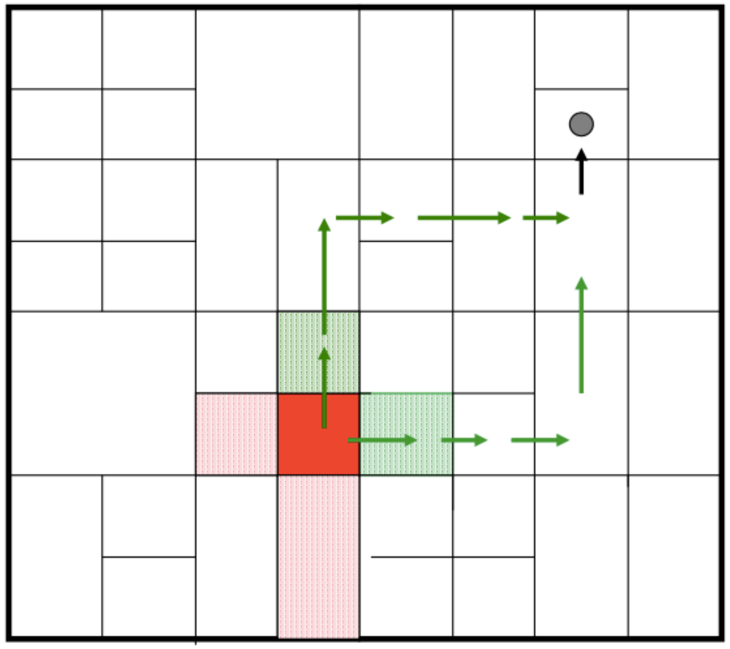
CAN

- In node X: put(K,V)
- $a = h_x(K)$
- $b = h_y(K)$
- Find route to the node in region (a,b)
- Store (K,V) in node



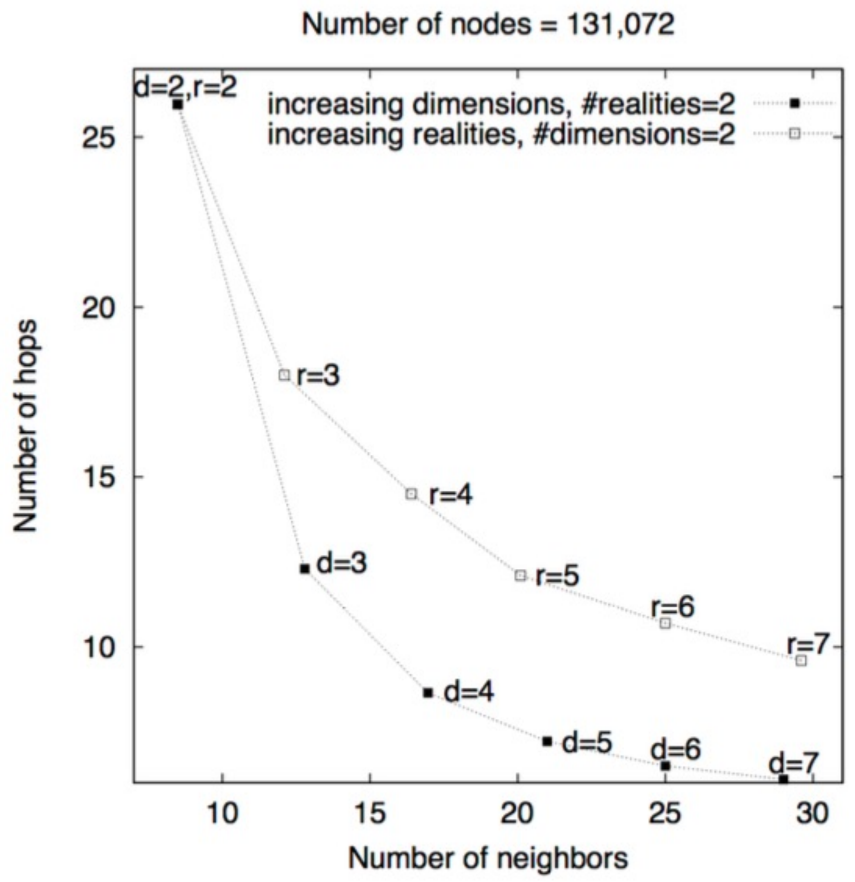
CAN

- Each node maintains list of its neighbor nodes
  - In 2D, neighbors have a common edge
- Number of hops in a d-dimension space: $O(dN^{1/d})$
- In case of failure, select alternative neighbor



CAN

- Improvements
  - Increase dimension
  - Increase Realities (Multiple independent coordinate spaces)
  - Replicate objects in several nodes

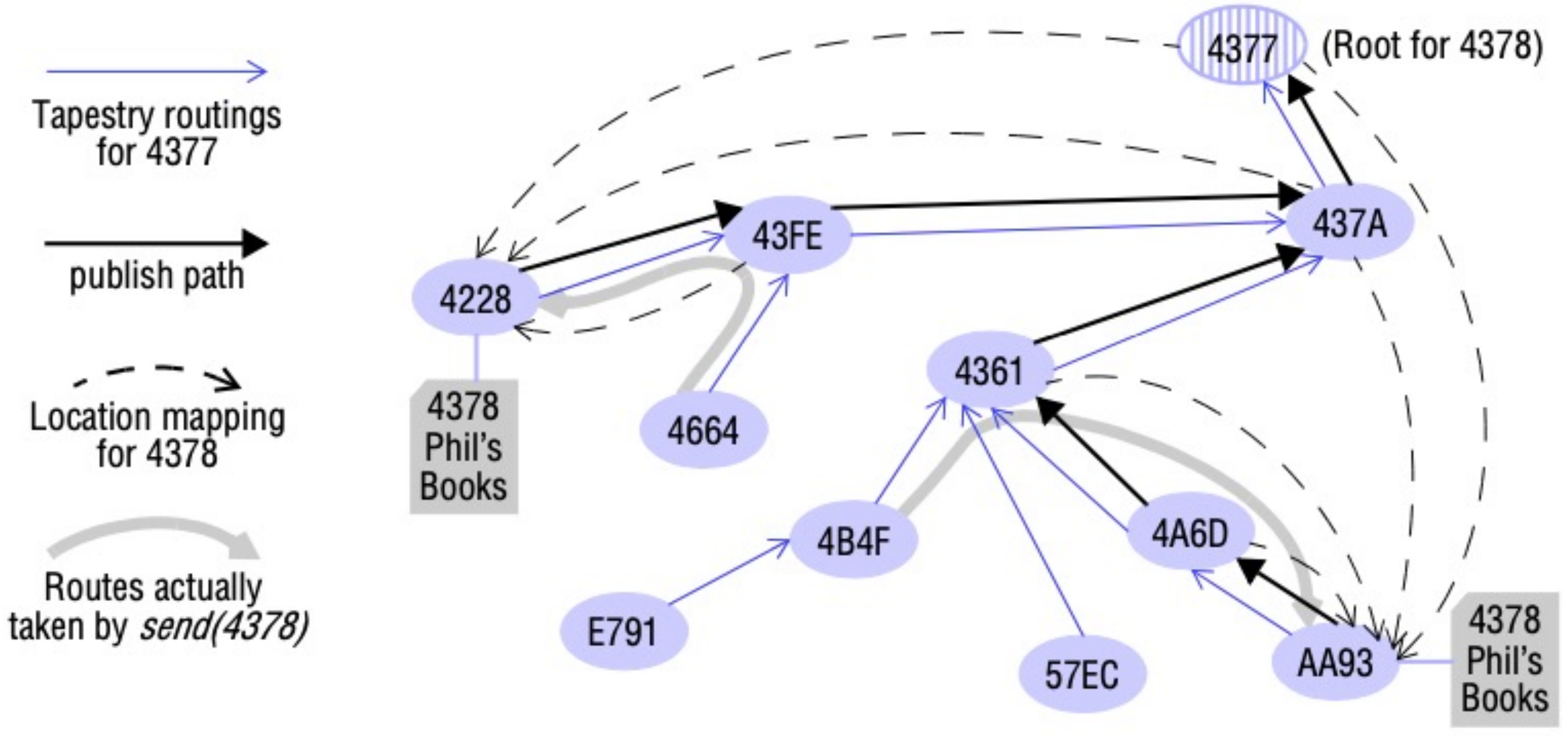


Tapestry

- Zhao, Kubiatowicz, “Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing”, 2001

- Very similar to Pastry
- Uses the same routing table and prefix routing semantics

Tapestry


Tapestry routings for 4377
publish path
Location mapping for 4378
Routes actually taken by *send(4378)*
4377
(Root for 4378)
437A
43FE
4228
4378 Phil's Books
4664
4361
4B4F
E791
57EC
4A6D
AA93
4378 Phil's Books

Tapestry

- Tapestry vs. Pastry
- Objects in Pastry are replicated without control by the owner
- Pastry assumes the actual object is replicated in replica and several neighbor nodes
- Pastry has no caching

Content Centric Networks (CCN)

CCN

- Also known as
  - Named Data Networking (NDN)
  - Information-Centric Networks

- We usually need some information about something from Internet
  - Why we just not ask about what we want?

CCN

- Current IP solution
  - Some entity publishes its content
    - There is no way to say others about its published content
  - Some companies (i.e. google) crawl and find published contents
  - We send request to google
  - Google tells us where (address) to get the information
  - Send request to that particular address

CCN

- What if we can just ask what we want?

- Why should we bother about where is the data then find and then connect to it

- Why should we translate what we want into numeric addresses?

CCN

- Few Samples
  - /ucla.edu/lixia/talks/icn16-arch.pptx
  - /ICN2016/tutorial/video/ndn-programming
  - /yourhome/bedroom/thermostat/temp

- CCN idea
  - Send request explicitly asking for the data without specifying destination

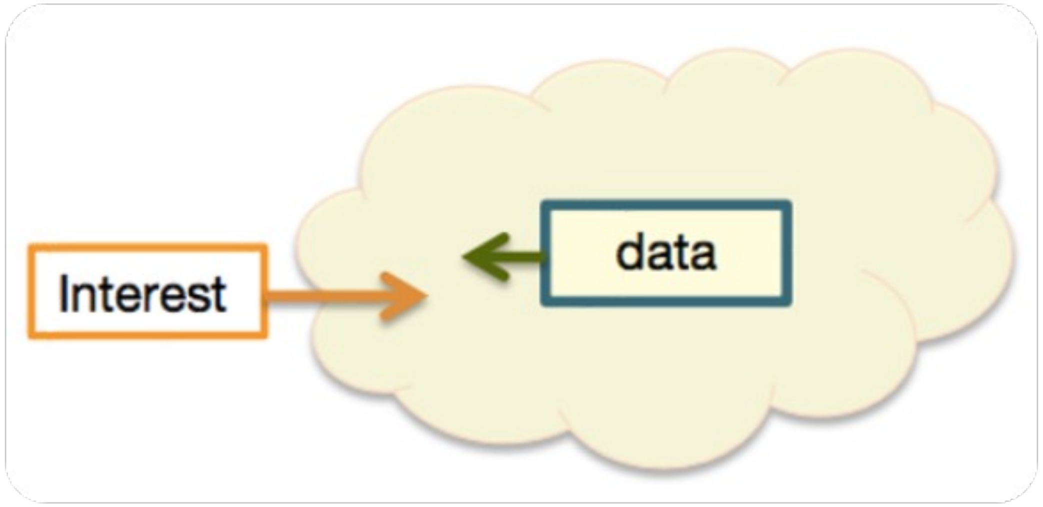
CCN

- History
  - It is developed in 2007 in PARC research center in California
  - In Jan 2016 source code is release publicly on Github
  - In Jan 2017 Cisco acquired CCN platform
  - Alcatel-Lucent (now part of Nokia), Huawei, Intel, Panasonic, and Samsung have also had substantial R&D efforts focused on CCN
    - It is currently named "Named Data Networking" (NDN)
      - https://named-data.net

- Remember we don't expect CCN to completely replace the Internet's protocols
  - Alternative architecture can offer better performance and security in many cases

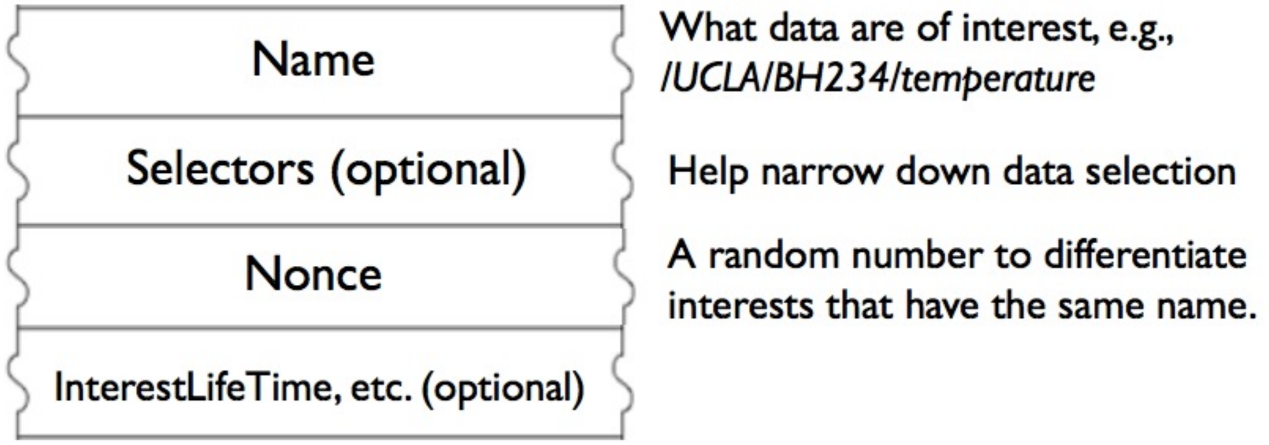
CCN

- CCN has two packet types
  - In IP Networks there is just one packet type to send request and deliver content



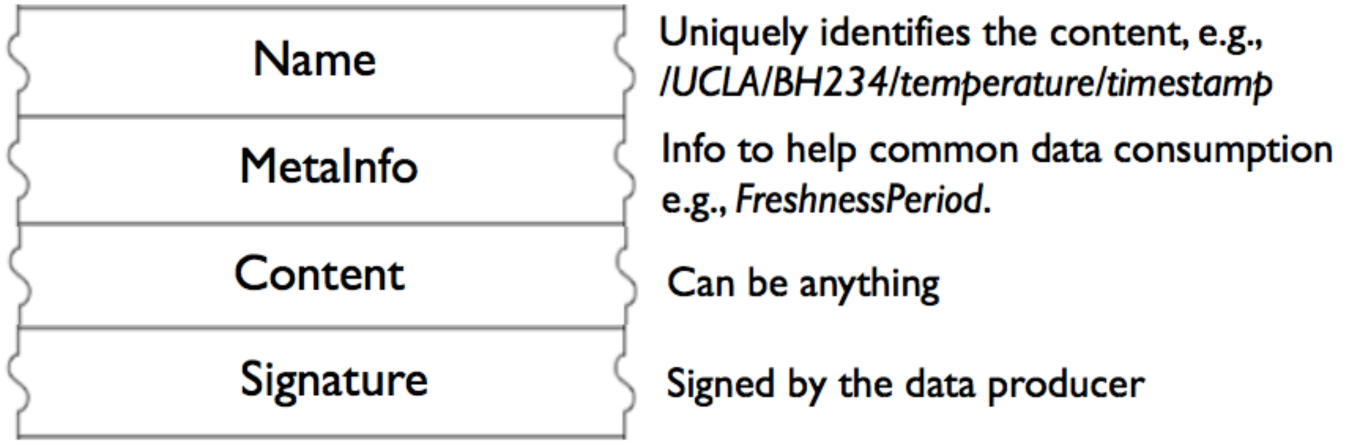
CCN Pakcets

- Interest Packet
- Interests are sent to retrieve Data


Name
Selectors (optional)
Nonce
InterestLifeTime, etc. (optional)
What data are of interest, e.g., /UCLA/BH234/temperature
Help narrow down data selection
A random number to differentiate interests that have the same name.

CCN Packets

- Data Packet
- Data is immutable, when content changes, name as well as signature should change too.


Name
MetaInfo
Content
Signature
Uniquely identifies the content, e.g., */UCLA/BH234/temperature/timestamp*
Info to help common data consumption e.g., *FreshnessPeriod*.
Can be anything
Signed by the data producer

CCN Names

- The name on a packet's label is a uniform resource identifier (URI)

- Name is hierarchical and has three main parts
  1. A prefix
     1. routers use to lookup the general destination for a piece of content
  2. Description of the specific content the packet holds or wishes to find
  3. List of any additional information,
     1. like when the content was created or in what order it should appear in a series.

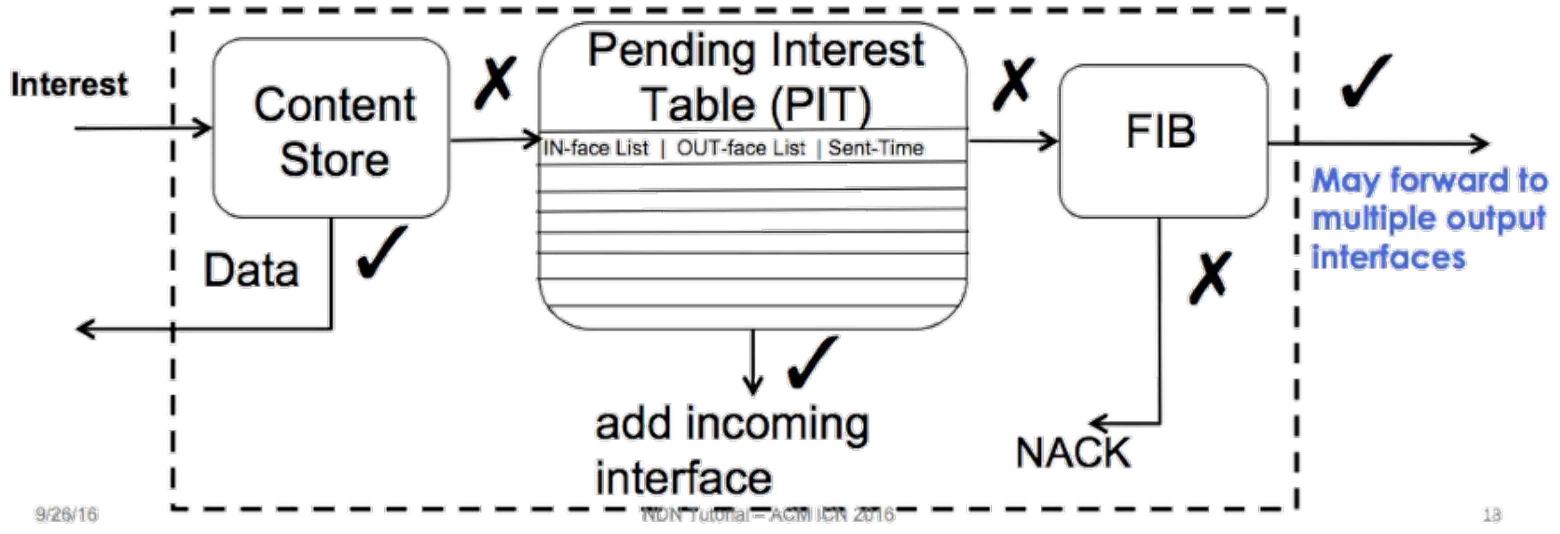
- Examples
  - /spectrum.ieee/2017/April/ver=2/chunk=9:540
    - Spectrum.ieee is the routable prefix for the second version of the article
    - The specific packet in question is the 9th packet of 540 that make up the complete article
  - /_ThisRoom/Projector/command/TurnOn/

Packet Forwarding

- No destination address in interest packets
- CCN Forwarders = routers
  - The forwarding engine decides
    - Where to store content
    - How to balance loads when traffic is heavy
    - Which route between two hosts is best
  - Has three major components
    - Content store
    - Pending Interest Table
    - Forwarding Information Base

Packet Forwarding

1. If data is available in content store (= cache) return data
2. If a similar request is pending wait to receive data
3. Search FIB database


Interest
Content Store
✗
Pending Interest Table (PIT)
IN-face List | OUT-face List | Sent-Time
✗
FIB
✓
May forward to multiple output interfaces
Data
✓
✓
add incoming interface
✗
NACK
9/26/16
NDN Tutorial – ACM ICN 2016
13

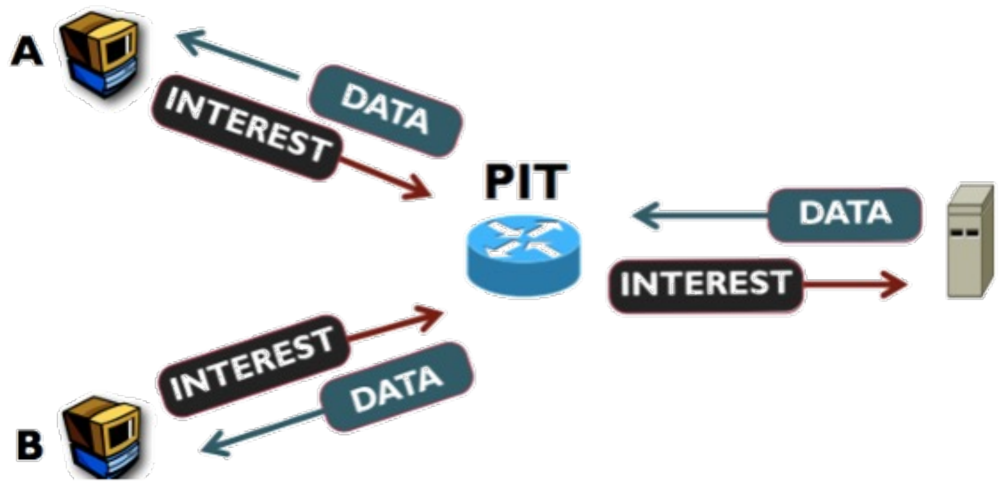
Content Store

- Any node can cache any content
  - No need for special nodes like CDN, cache servers in current IP networks

- Advantages
  - Reduce redundant traffic for ISPs
  - Reduce server load for producers, especially during attacks
  - Reduce response time for consumers
    - The entire Internet acts like a big CDN
  - It helps loss recovery and mobility
  - Decouples data production and data consumption

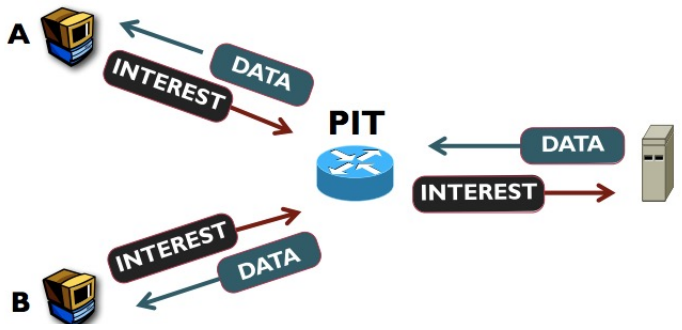
Pending Interests Table

- Each entry records: [name, nonce, incoming faces]
- Created when a new Interest is forwarded
- Updated when more interests carrying the same name arrive
- Deleted when a matching Data returns
- Rows are saved and searched based on the longest prefix matching from URI
    - Compare with LPM of IP addresses



Pending Interests Table

- Native Multicast
- Suppress duplicate packets
- Provide closed-loop feedback for the success/performance of data retrieval, at every hop
  - Minimize path changes, or look for the shortest delay, or look for higher throughput, or always multicast/broadcast
- Different from conventional networks, where routers immediately "forget" information they've forwarded.



Forwarding Information Base (FIB)

- Index of all URI prefixes, or routable destinations, in the entire network
- It stores name-prefixes and corresponding next-hops: /UCLA
- Perform longest prefix match with incoming Interest’s name.
  - E.g., /UCLA/RoyceHall/⋯
- Multiple next-hops, which may lead to different data sources
  - Since it is forwarding towards data, not a particular destination
  - CCN enjoys more forwarding choices since there is no concern of loops

- Building FIB
  - Applications register their data’s name prefix with local node
  - Forwarders can use any model to announce prefixes
    - Link states, flooding, ⋯

CCN

- It is shown that CCN is better than current IP networks in

- Reliability
- Scalability
- Security
  - Data is encrypted and has digital signature, thus can be saved anywhere in the network without any concern

I'M FINISHED