HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Overlay and P2P Networks

Applications

Prof. Sasu Tarkoma

11.2.2013

Contents

Today
- Bittorrent Mainline DHT
- Scribe and PAST
- P2PSIP
- Amazon’s Dynamo
- CDNs

Thursday
- Samu Varjonen: lookups and DNS

Monday
- Remaining applications
- Some advanced topics
- Summary

Bittorrent Mainline DHT

Decentralized tracker (trackerless torrent)

Based on Kademlia

Uses a custom RPC based on UDP

The **key** is the **info-hash**, the hash of the metadata. It uniquely identifies a torrent.

The **data** is a peer list of the peers in the swarm

Torrents have bootstrap nodes in the overlay

BitTorrent Mainline DHT

Each peer announces itself with the distributed tracker
Looking up the 8 nodes closest to the info-hash of the torrent and sending an announce message to them

Those 8 nodes will then add the announcing peer to the peer list stored at that info-hash

A peer joins a torrent by looking up the peer list at a specific info-hash

Nodes return the peer list if they have it

Kademlia in Bittorrent Mainline DHT

The implementation extends the single bit model discussed before

The single bit model can be seen to have a prefix first n-1 bits need to match for the nth list

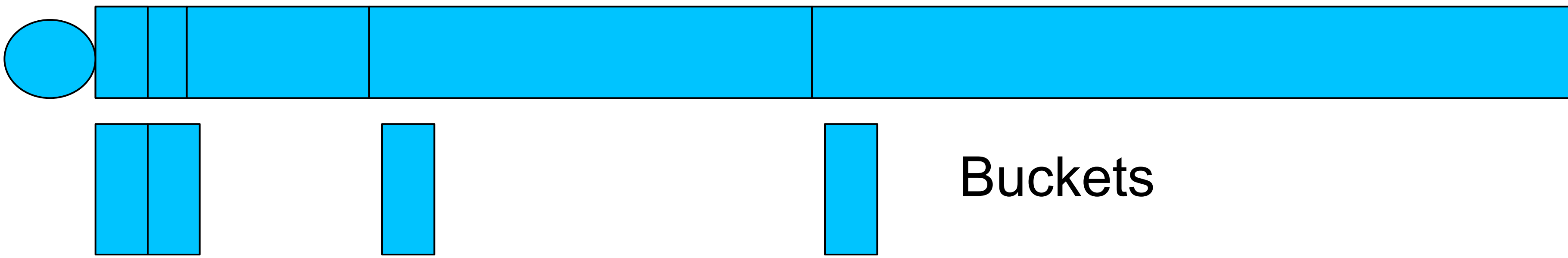
The extension introduces prefix (group of bits)-based operation with width w for digits, giving $2^w - 1$ k-buckets with the missing one containing the node ID

An m-bit prefix reduces the maximum number of lookups from $\log_2 n$ to $\log_2^w n$

This results in a prefix-based routing table!

Kademlia Routing Table Revisited

Node distance and subtrees



Each node knows more about close nodes than distant nodes

Key space of each bucket grows with the power of 2 with the distance

Querying for an ID will on average halve the distance to the target in the each step

Query Routing

Goal: Find k nodes closest to ID T

Initial Phase:

- Select α nodes closest to T from the routing table
- Send FIND_NODE(T) to each of the α nodes in parallel

Iteration:

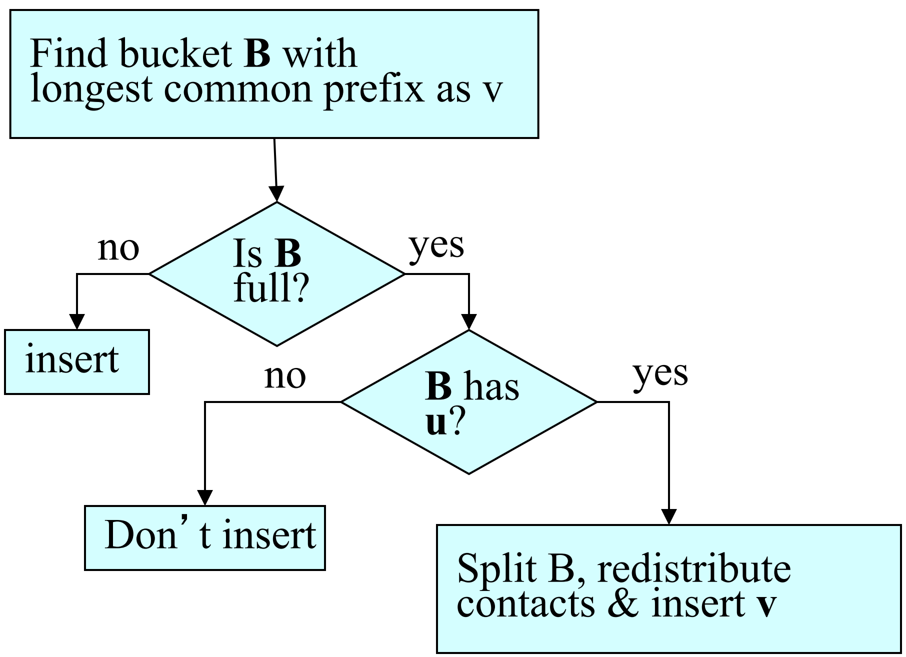
- Select α nodes closest to T from the results of previous RPC
- Send FIND_NODE(T) to each of the α nodes in parallel
- Terminate when a round of FIND_NODE(T) fails to return any closer nodes

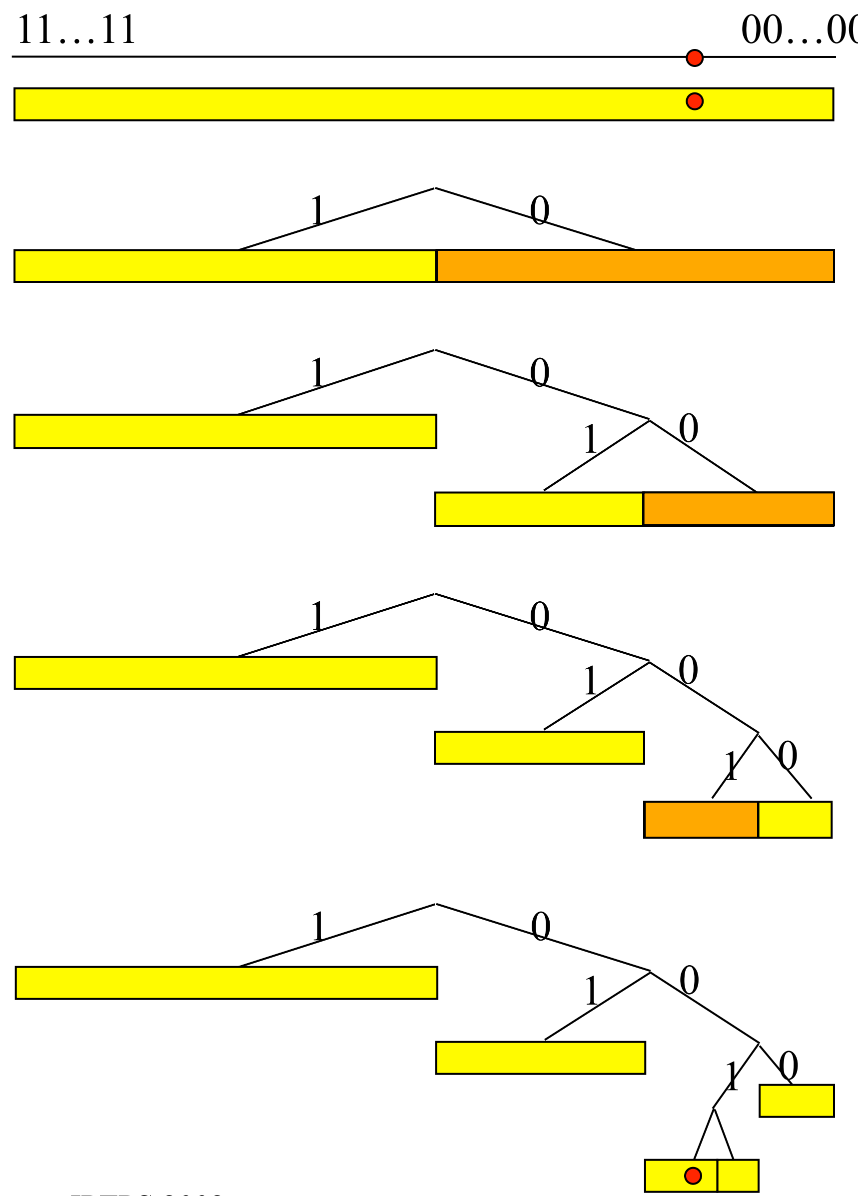
Final Phase:

- Send FIND_NODE(T) to all of k closest nodes not already queried
- Return when have results from all the k-closest nodes.

Node Joining & Routing Table Evolution

- Joining Node (**u**):
  - ✓ Borrow an alive node's ID (**w**) off-line
  - ✓ Initial routing table has a single k-bucket containing u and w.
  - ✓ u performs FIND_NODE(**u**) to learn about other nodes
- Inserting new entry (**v**)





Comparisons

Kademlia and Chord

- Chord has only one direction on the ring
- Incoming traffic cannot be used to improve routing table
- But Chord has pred/succ (sequential neighbours)

Kademlia and Pastry

- Pastry has more complex table
- Pastry has sequential neighbours

What about Mainline DHT in practice?

Implementation details

Mainline DHT implements Kademlia with a width of 2, and k = 8 nodes in each bucket

Keys are replicated on the three nodes with nodeID nearest the key with a 30-minute timeout

If a node fails, the keys will be lost

Nodes learn implicitly
- Iterative queries, incoming messages
- Lazy removal
- Ping LRU node when bucket full

Reported Problems with Mainline DHT

An Analysis of BitTorrent's Two Kademlia-Based DHTs

Scott A. Crosby and Dan S. Wallach, 2007

Do the DHTs work correctly? No. Mainline BitTorrent dead-ends its lookups 20% of the time and Azureus nodes reject half of the key store attempts.

What is the DHT lookup performance? Both implementations are extremely slow, with median lookup times around a minute.

Why do lookups take over a minute? Lookups are slow because the client must wait for RPCs to timeout while contacting dead nodes. Dead nodes are commonly encountered in the area closest to the destination key.

Why are the routing tables full of dead nodes? Kademlia's use of iterative routing limits the ability for a node to opportunistically discover dead nodes in its routing table (refresh. explicit ping)

Design Problems

Iterative search can return dead nodes (no checking)
- Recursive routing would implicitly define liveness

Dead nodes are pruned only with refresh or explicit ping

XOR metric
- cannot enumerate nodes (as in Pastry or Chord)

Nodes can be ordered based on distance to given key

PAST

PAST: Cooperative, Archival File Storage and Distribution

Runs on top of Pastry, pastry routes to closest live nodeId

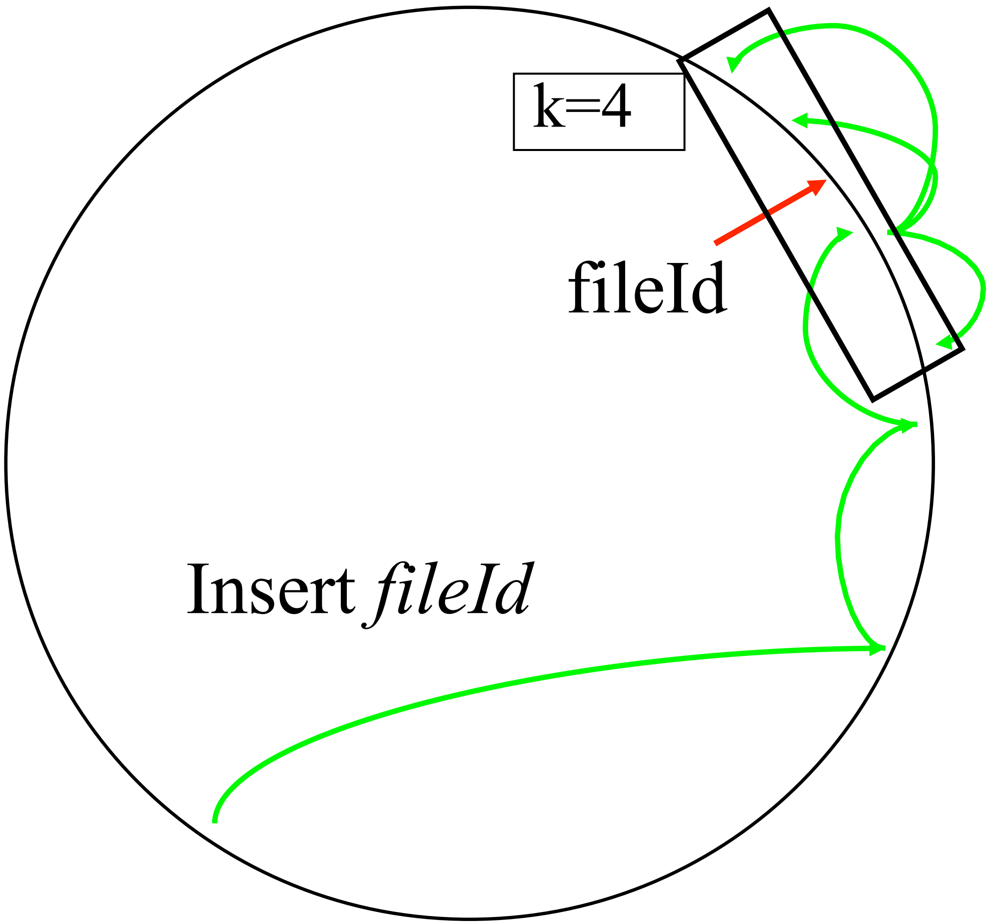
Strong persistence, high availability, scalability

API:

- Insert: store replica of a file at k diverse storage nodes
- Lookup: retrieve file from a nearby live storage node
- Reclaim: free storage associated with a file

Files are immutable!

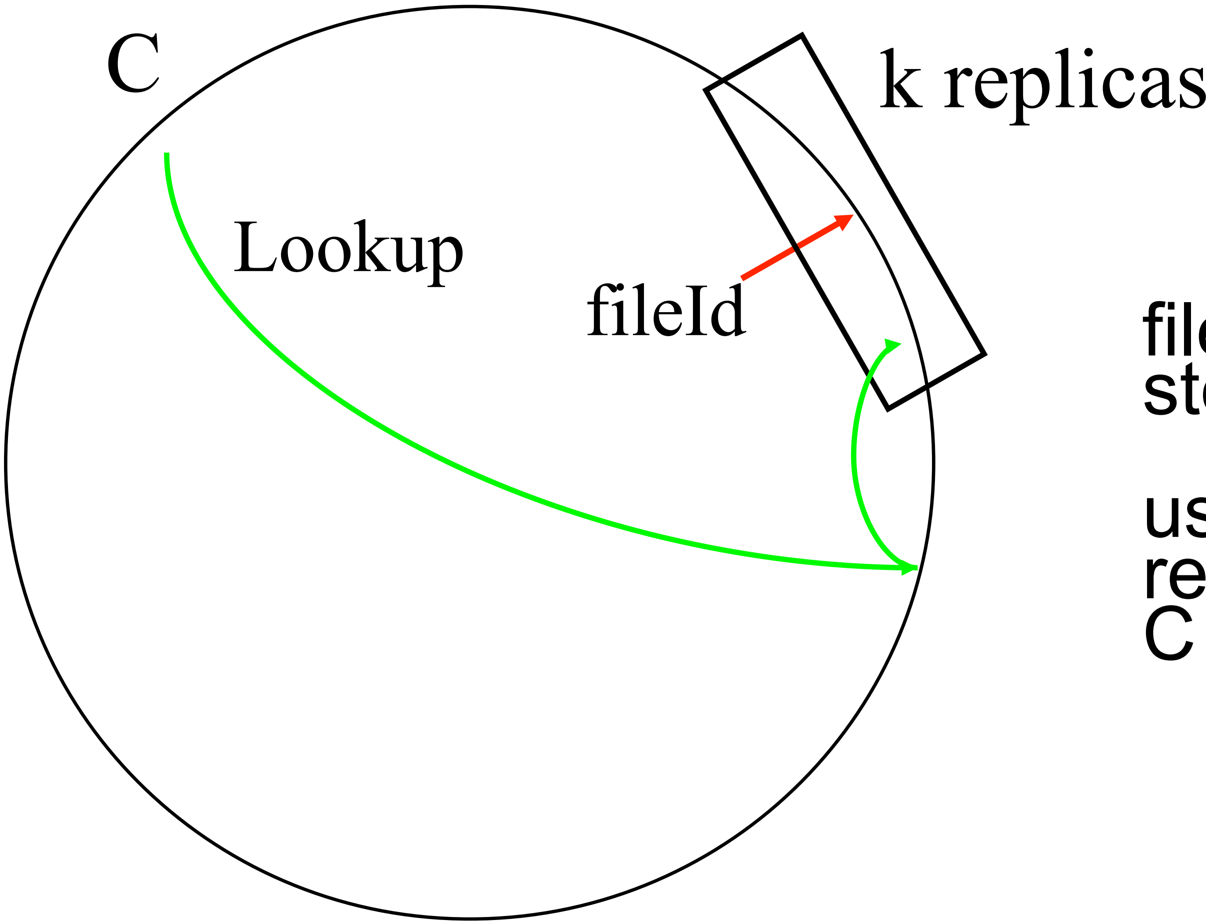
PAST File Storage



Storage Invariant: File "replicas" are stored on k nodes with nodeIds closest to fileId

(k is bounded by the leaf set size)

PAST File Retrieval



file located in $\log_{16} N$ steps (expected)

usually locates replica nearest client C

PAST Features

Caching

- Nodes cache on nodes along the route of lookup and insert messages (as in Freenet)
- Aim to balance load

Security

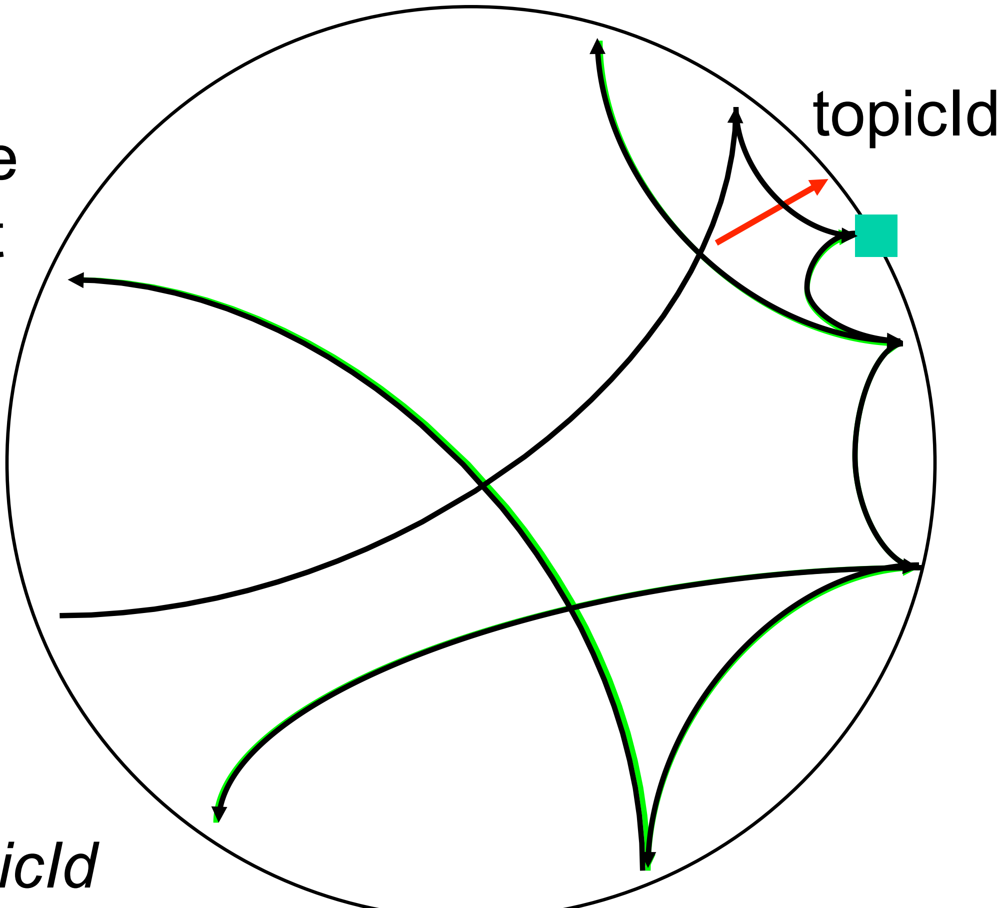
- No read access control, encryption can be used
- File authenticity with certificates
- System integrity: ids non-forgeable, sign sensitive messages
- Randomized routing

SCRIBE

SCRIBE: Large-scale, decentralized multicast

Intrastructure to support topic-based publish/
subscribe applications

Reasonable performance
compared to IP multicast



Publish *topicId*

Subscribe *topicId*

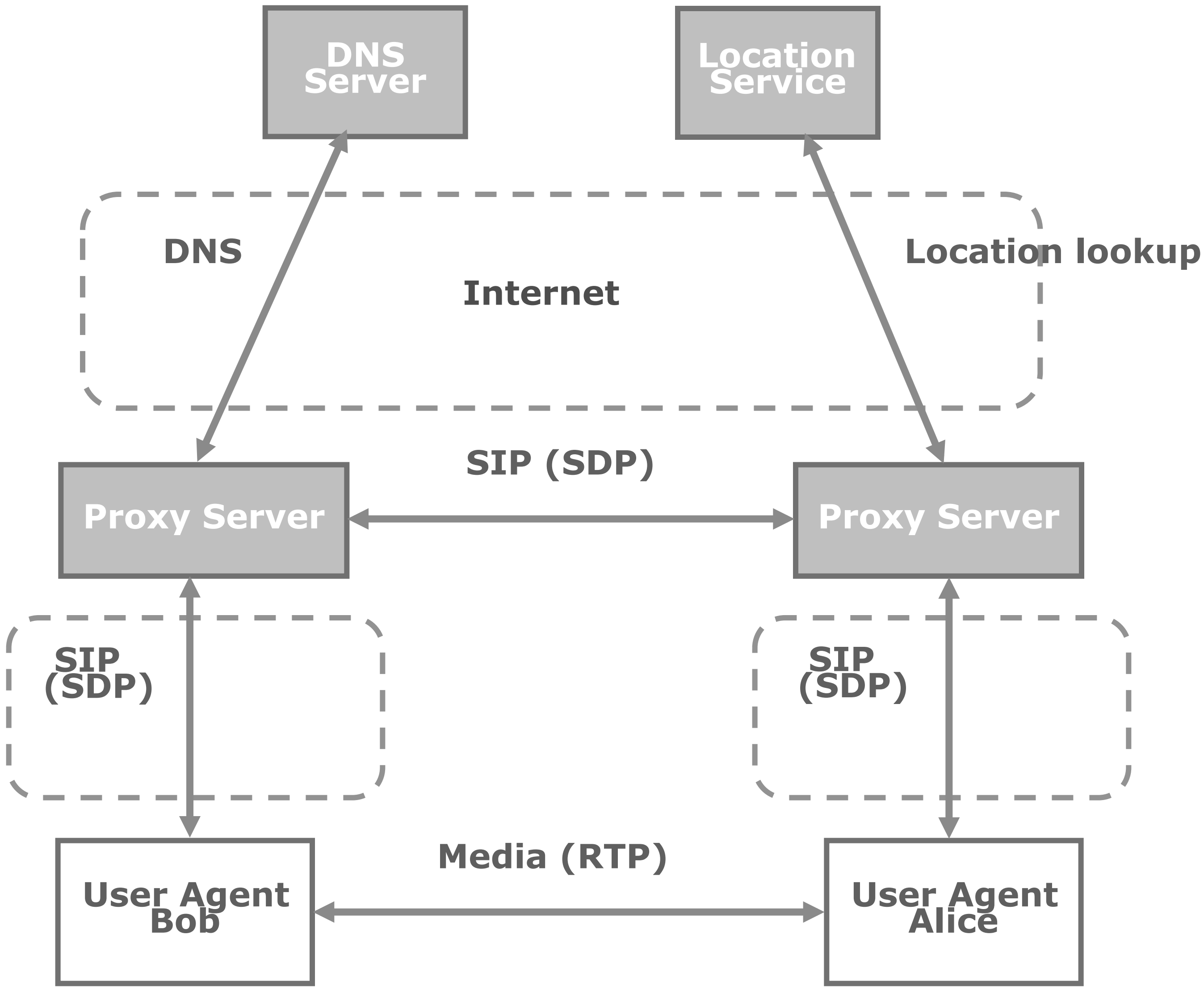
Session Initiation Protocol (SIP)

An Application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants

Sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution

Members in a session can communicate via multicast or via a mesh of unicast relations, or a combination of these

Text based, model similar to HTTP


DNS Server
Location Service
DNS
Internet
Location lookup
SIP (SDP)
Proxy Server
Proxy Server
SIP (SDP)
SIP (SDP)
Media (RTP)
User Agent Bob
User Agent Alice

P2P SIP

SIP is already ready for P2P

- Active standardization in IETF

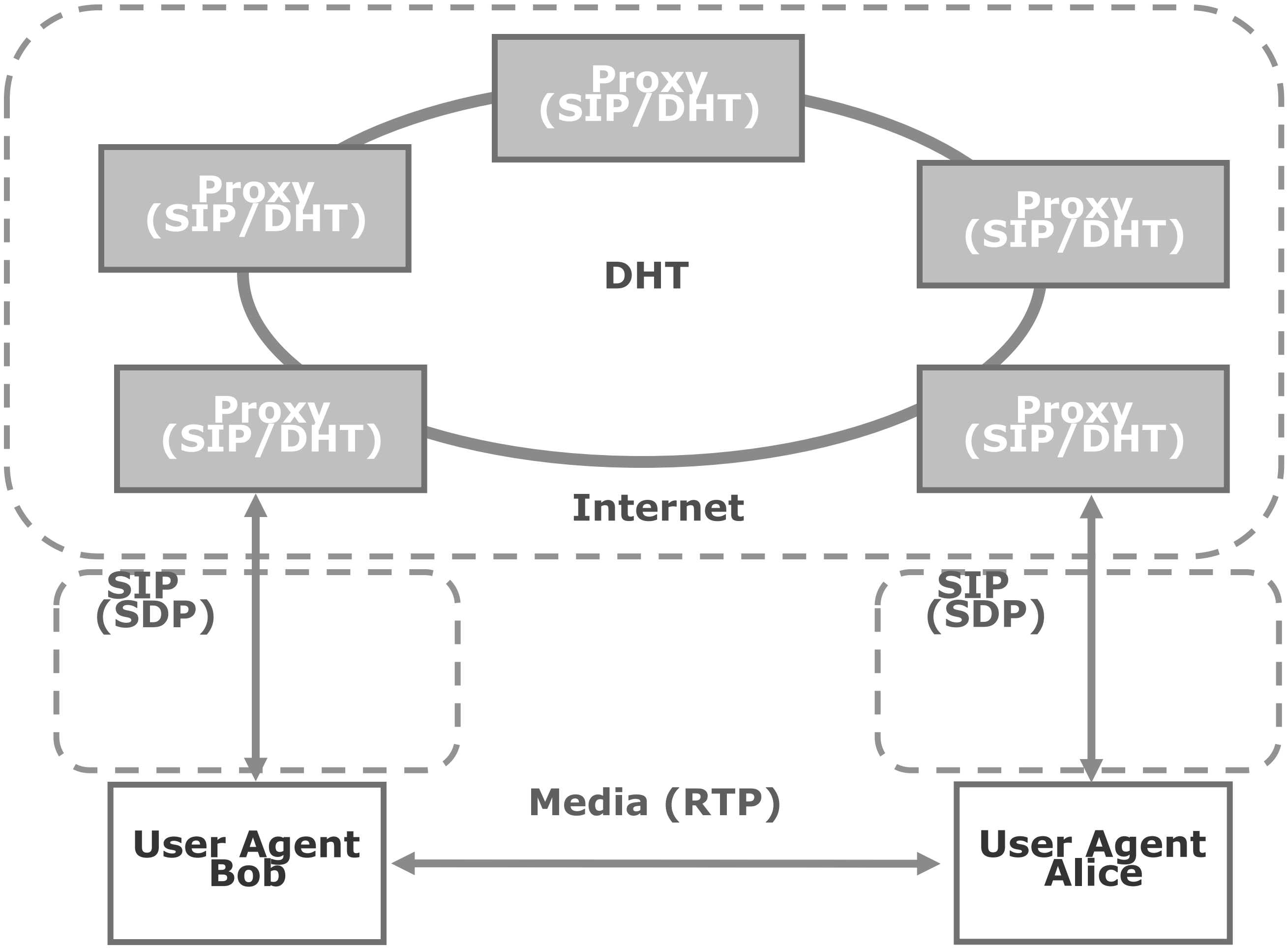
Uses symmetric, direct client-to-client communication

Intelligence resides mostly on the network border in the user agents

The proxies and the registrar only perform lookup and routing

The lookup/routing functions of the proxies/registrar can be replaced by a DHT overlay built in the user agents.

By adding join, leave and lookup capabilities, a SIP user agent can be transformed into a peer capable of operating in a P2P network


Proxy
(SIP/DHT)
Proxy
(SIP/DHT)
Proxy
(SIP/DHT)
DHT
Proxy
(SIP/DHT)
Proxy
(SIP/DHT)
Internet
SIP
(SDP)
SIP
(SDP)
Media (RTP)
User Agent
Bob
User Agent
Alice

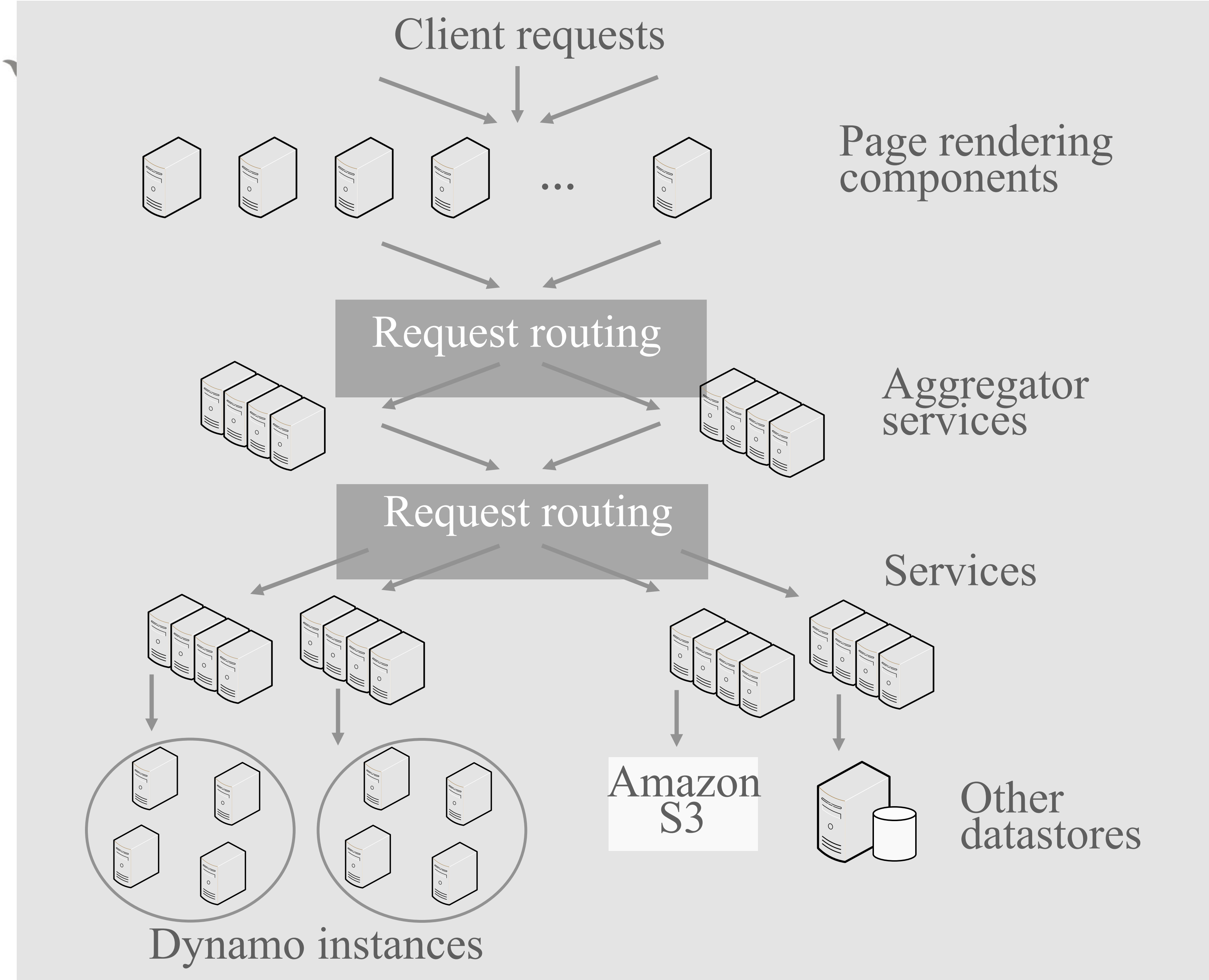
Amazon Dynamo Motivation

Aim is to store various kinds of data and have high availability

Build a distributed storage system:

- Scale
- Simple: **key-value**
- Highly available
- Guarantee **Service Level Agreements (SLA)**

Based on the SOSP 2007 presentation and paper:
Dynamo: Amazon's Highly Available Key-value Store


Client requests
Page rendering components
...
Request routing
Aggregator services
Request routing
Services
Amazon S3
Other datastores
Dynamo instances

System Assumptions and Requirements

Query Model: simple read and write operations to a data item that is uniquely identified by a key

ACID Properties: Atomicity, Consistency, Isolation, Durability

Efficiency: latency requirements which are in general measured at the 99.9th percentile of the distribution

Other Assumptions: operation environment is assumed to be non-hostile and there are no security related requirements such as authentication and authorization

Service Level Agreements (SLA)

Application can deliver its functionality in **bounded time**: Every dependency in the platform needs to deliver its functionality with even tighter bounds

Example: *service guaranteeing that it will provide a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second*

Dynamo Design Consideration

Sacrifice strong **consistency** for **availability**

Conflict resolution is executed during ***read*** instead of ***write***

- Use quorums and other techniques

Other principles:

- Incremental scalability
- Symmetry
- Decentralization
- Heterogeneity

CAP Theorem

CAP, first conceived in 2000 by Eric Brewer and formalized into a theorem in 2002 by Nancy Lynch

A useful model for describing the fundamental behavior of NoSQL systems

CAP is generally described as following:
Of three desirable properties you want in your system: **consistency**, **availability** and **tolerance** of network partitions,

you can only choose two.

Summary of techniques used in *Dynamo* and their advantages

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff (use another server for replica if proper one is not available)	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees (summarization of key ranges of virtual nodes)	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Dynamo Implementation

Data Stores

Nodes in the system are spread around a logical circle

Nodes are responsible for the region between it and its predecessor

Virtual nodes are evenly dispersed and appear to be regular nodes in the system, but in reality are just handled by the nodes of the system

Can be geographically distributed

Object Data

Uses hashing of an object's key to determine where to store the object

Each object is replicated across N nodes (N-1 successor nodes to the coordinator node)

Consistent Hashing Revisited

Properties

- Smoothness→addition of bucket does not cause movement between existing buckets
- Spread & Load→small set of buckets that lie near object
- Balance→no bucket is responsible for large number of objects

Moderate load imbalance is possible

Virtual nodes address this

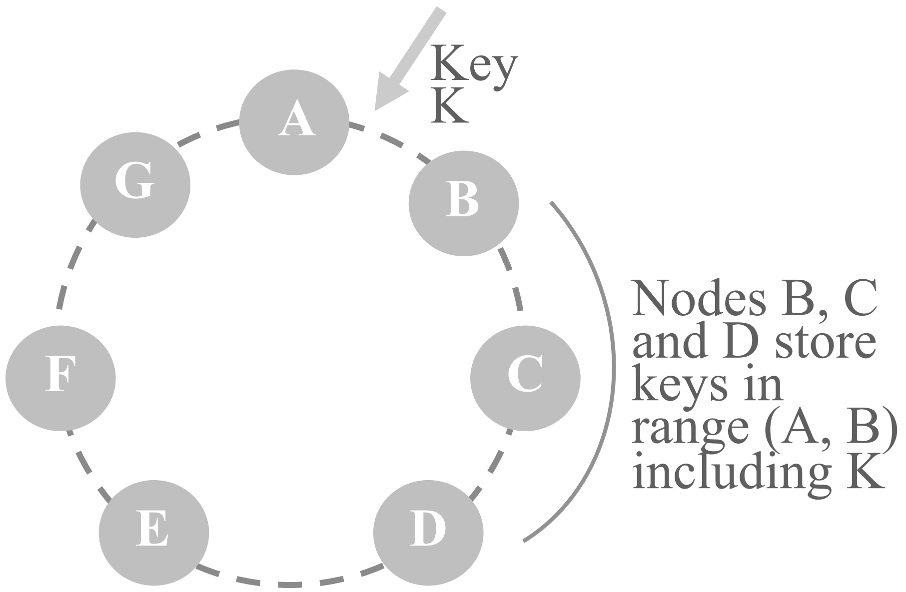
Log n replication factor gives O(items/n) balance with high probability for a high number of uniformly distributed items

Partition Algorithm

Consistent hashing: the output range of a hash function is treated as a fixed circular space or “ring”.

”Virtual Nodes”: Each node can be responsible for more than one virtual node.

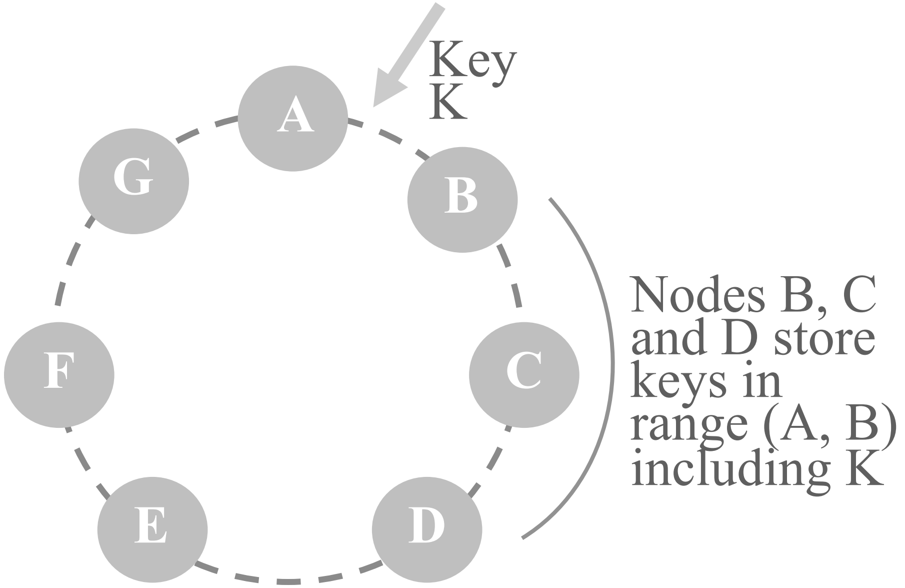
Virtual nodes are needed to address data/node imbalance problem



Replication

Each data item is replicated at N hosts

"*preference list*": The list of nodes that is responsible for storing a particular key



Data Versioning

A put() call may return to its caller before the update has been applied at all the replicas

A get() call may return many versions of the same object

Challenge: an object having distinct version sub-histories, which the system will need to reconcile in the future

Solution: uses **vector clocks** in order to capture causality between different versions of the same object

Vector Clock

A vector clock is a list of (node, counter) pairs

Every version of every object is associated with one vector clock

If the counters on the first object's clock are less-than-or-equal to all of the nodes in the second clock, then the first is an ancestor of the second and can be forgotten

Sloppy Quorum

The sloppy quorum technique is used to handle temporal faults

Read/Write involve **N** nodes (preference list)
R/W is the minimum number of nodes that must participate in a successful read/write operation

Setting **R + W > N** yields a quorum-like system.

In this model, the latency of a get (or put) operation is dictated by the slowest of the R (or W) replicas

R and W are usually configured to be less than N, to provide better latency

Typical values (3,2,2)

Gossip

A gossip-based protocol propagates membership changes and maintains an eventually consistent view of membership

Each node contacts a peer chosen at random every second

The two nodes efficiently reconcile their persisted membership change histories.

Also reconcile position information on the ring (virtual buckets)

Hinted handoff

The hinted handoff is also used to handle temporal faults

Assume N = 3. When A is temporarily down or unreachable during a write, send replica to D

D is hinted that the replica belongs to A and it will deliver to A when A is recovered

As a result A is always writable

Dynamo Execution

Writes

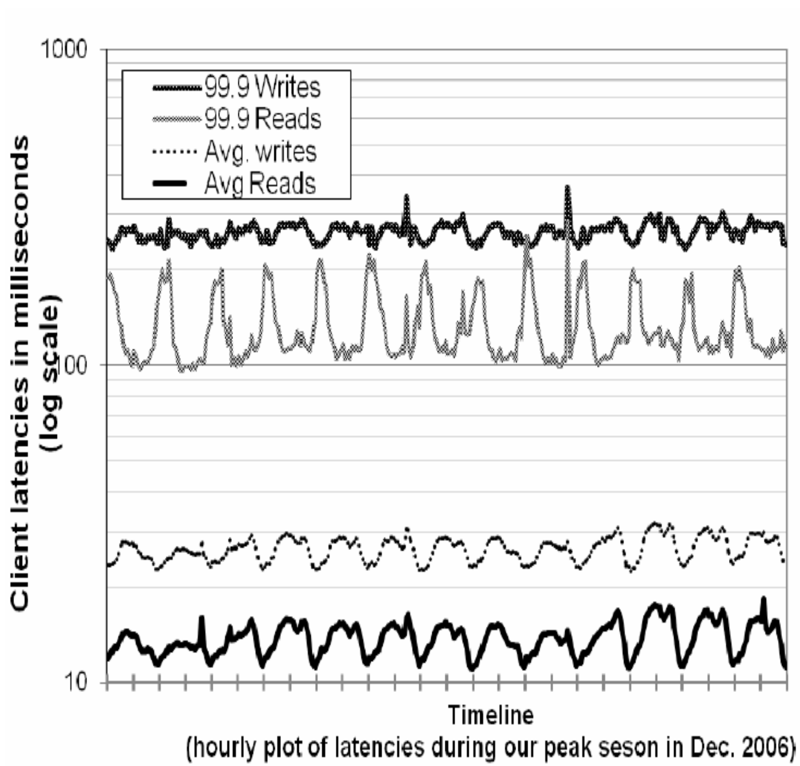
- Requires generation of a new vector clock by coordinator
- Coordinator writes locally
- Forwards to N nodes, if W-1 respond then the write was successful

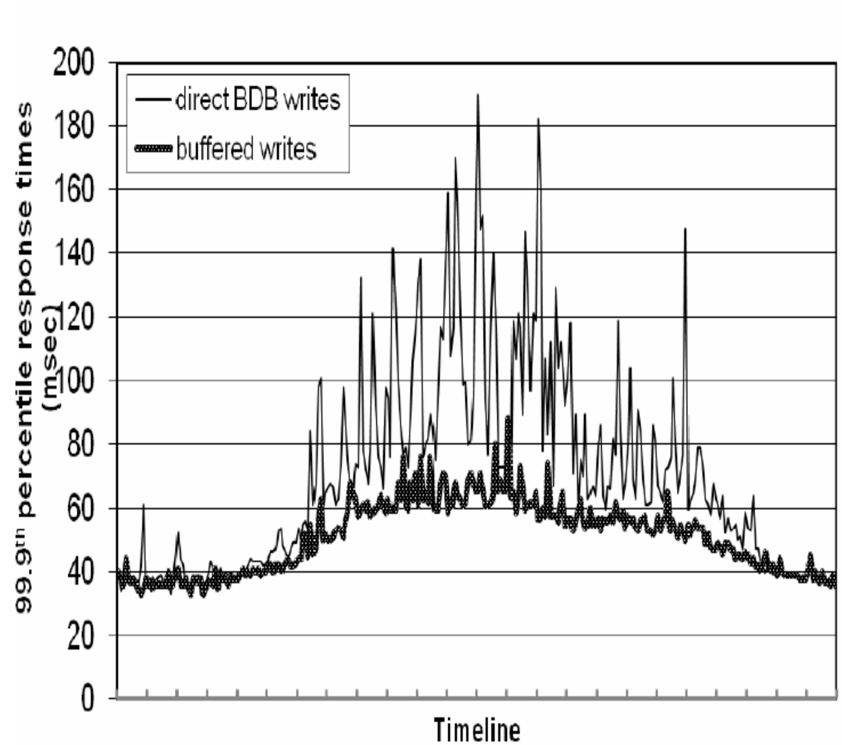
Reads

- Forwards to N nodes, if R-1 respond then forwards to user
- Only unique responses forwarded
- User handles merging if multiple versions exist

Results

Their response requirement is 300ms for any request (read or write)





Dynamo Summary

"Eventually" consistent data store

Always writable

Decentralized

All nodes have the same responsibilities

Amazon.com's Resolution

- Weakening consistency property in the system
- Increase the availability

Content Delivery Networks (CDN)

Geographically distributed network of Web servers around the globe (by an individual provider, E.g. Akamai).

Improve the performance and scalability of content retrieval.

Allow several content providers to replicate their content in a network of servers.

Motivation

Network cost

Huge cost involved in setting up clusters of servers around the globe and corresponding increase in network traffic

Economic cost

Higher cost per service rate making them inaccessible to lower and medium level customers

Social cost

Monopolization of revenue

CDN Technology

Intelligent wide area traffic management

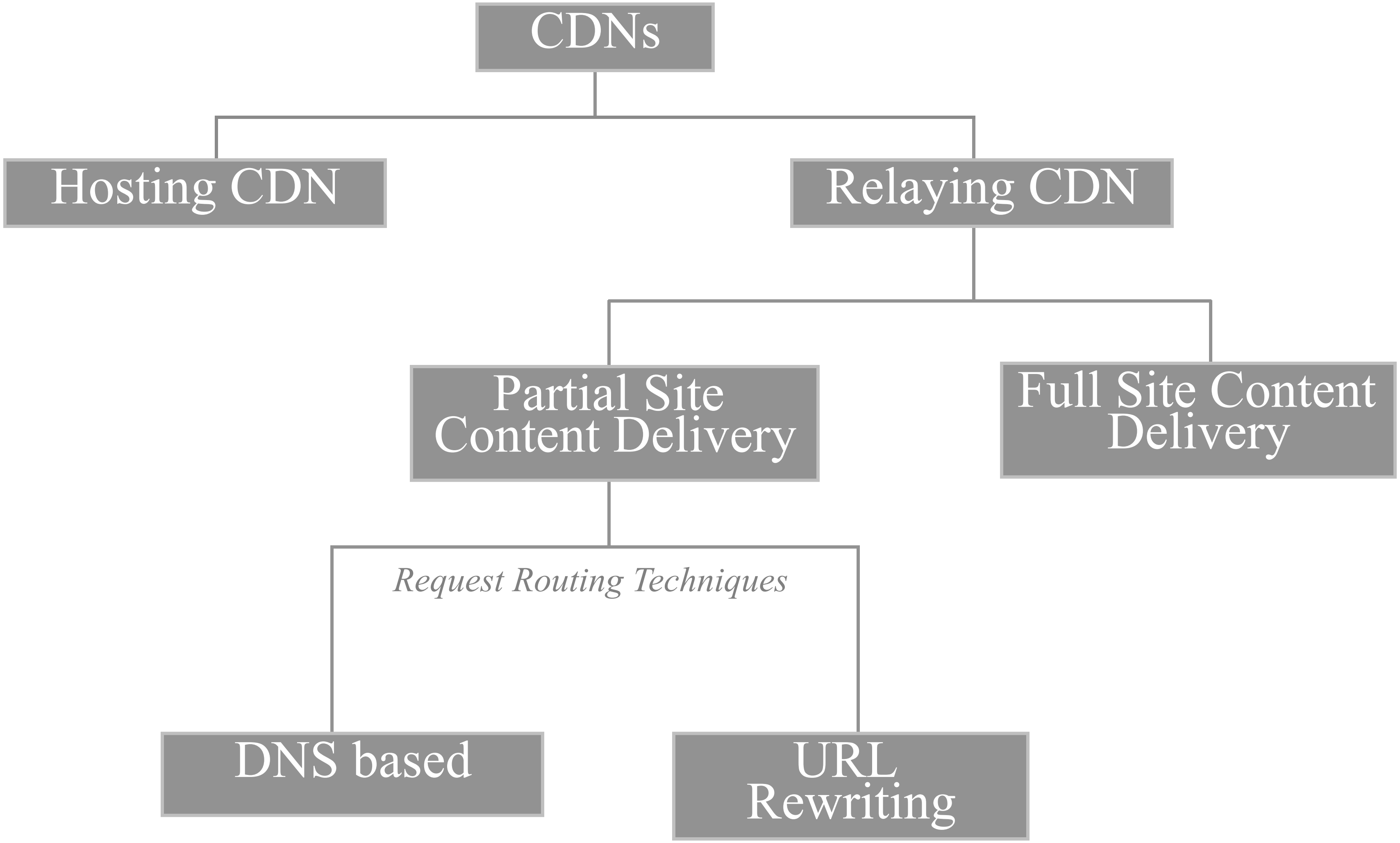
- Direct clients' requests to optimal site based on topological proximity

Two types of redirection: **DNS redirection** or **URL rewriting**

Cache

- Saves useful contents in cache nodes.
- Two cache policies: least frequently used standard and least recently used standard.

CDN Types *(Skeletal)*



CDN

Replicate content on many servers

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
  - DNS, HTTP redirect, anycast, etc.

Akamai

Server Selection

Service and content is replicated in many places in network

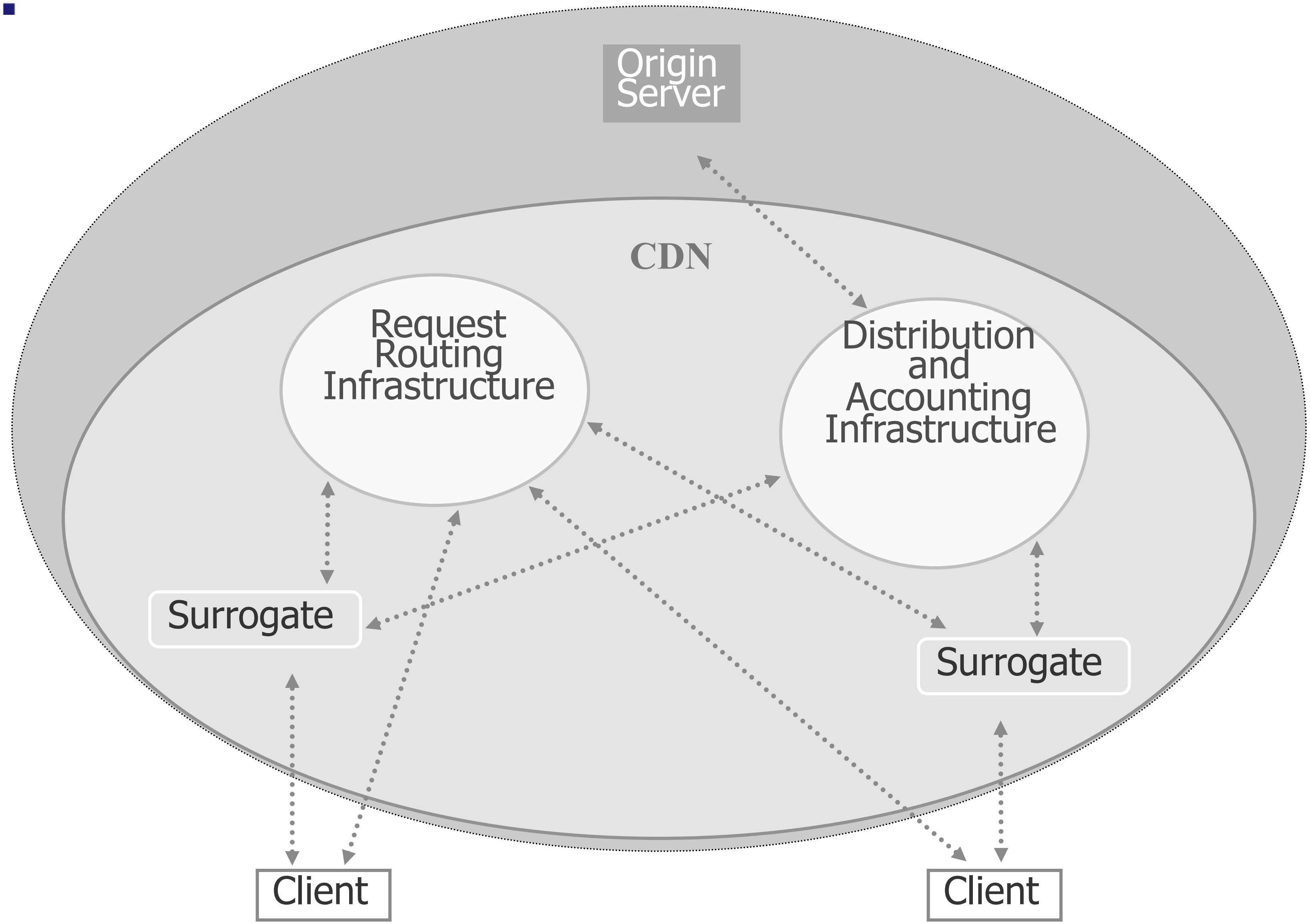
How to direct clients to a particular server?

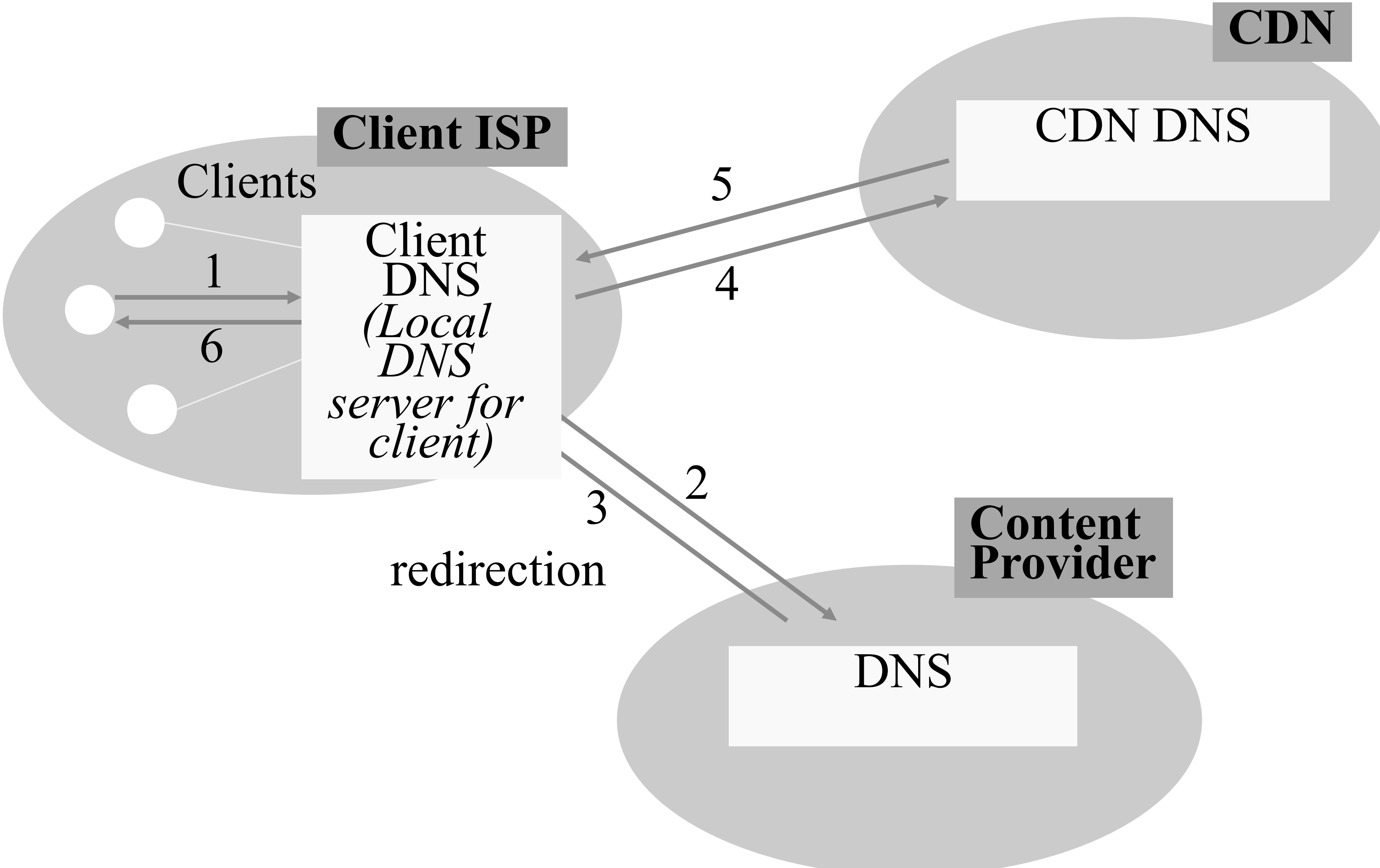
- As part of routing → anycast, cluster load balancing
- As part of application → HTTP redirect
- As part of naming → DNS

Which server to use?

- Best performance → to improve client performance
  - Based on Geography? RTT? Throughput? Load?
- Lowest load → to balance load on servers
- Any active node → to provide availability

CDN Architecture




CDN
CDN DNS
Client ISP
Clients
5
Client DNS (Local DNS server for client)
1
4
6
2
3
Content Provider
redirection
DNS

CDN	Type	Coverage	Solutions
Akamai	Commercial CDN service including streaming data	Market leader	Edge platform for handling static and dynamic content, DNS-based request-routing
Limelight Networks	Commercial On-demand distribution, live video, music, games, …	Surrogate servers in over 70 locations in the world	Edge-based solutions for content delivery, streaming support, custom CDN for custom delivery solutions, DNS-based request-routing
Coral	Academic Content replication based on popularity (on demand), addresses flash crowds	Experimental, hosted on PlanetLab	Uses a DHT algorithm (Kademlia), support for static content, DNS-based request-routing
CoDeeN	Academic testbed Caching of content and redirection of HTTP requests	Experimental, hosted on PlanetLab, collaborative CDN	Support for static content, HTTP direction Consistent hashing for mapping data to servers
Globule	Academic Replication of content, server monitoring, redirection to available replicas	Apache extension, Open Source collaborative CDN	Support for static content, monitoring services, DNS-based request-routing

Akamai

Clients fetch html document from primary server
URLs for replicated content are replaced in html

Client resolves aXYZ.g.akamaitech.net hostname

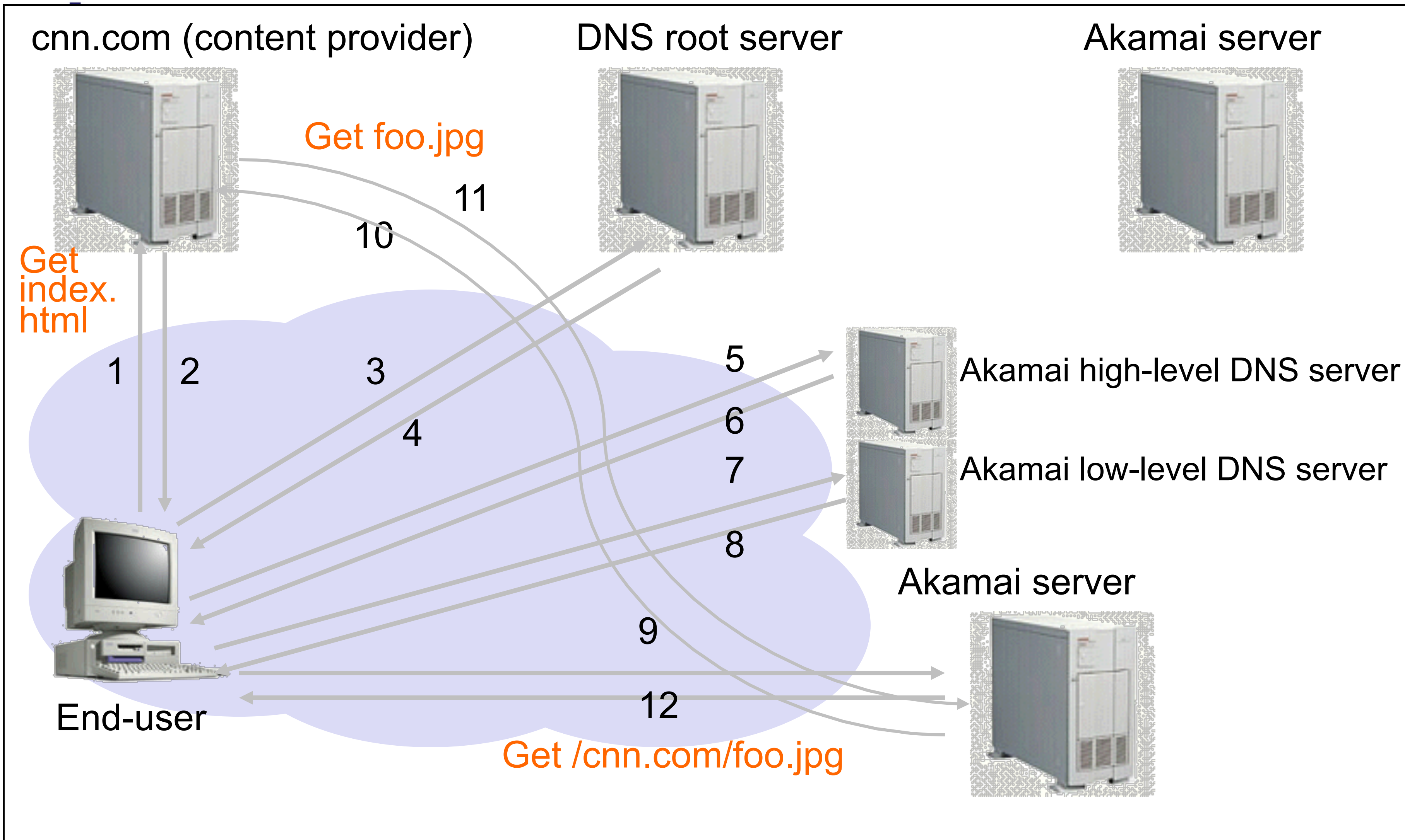
Akamai.net name server returns NS record for g.akamaitech.net

- G.akamaitech.net nameserver choses server in region

- Should try to choose server that has file in cache - How to choose?

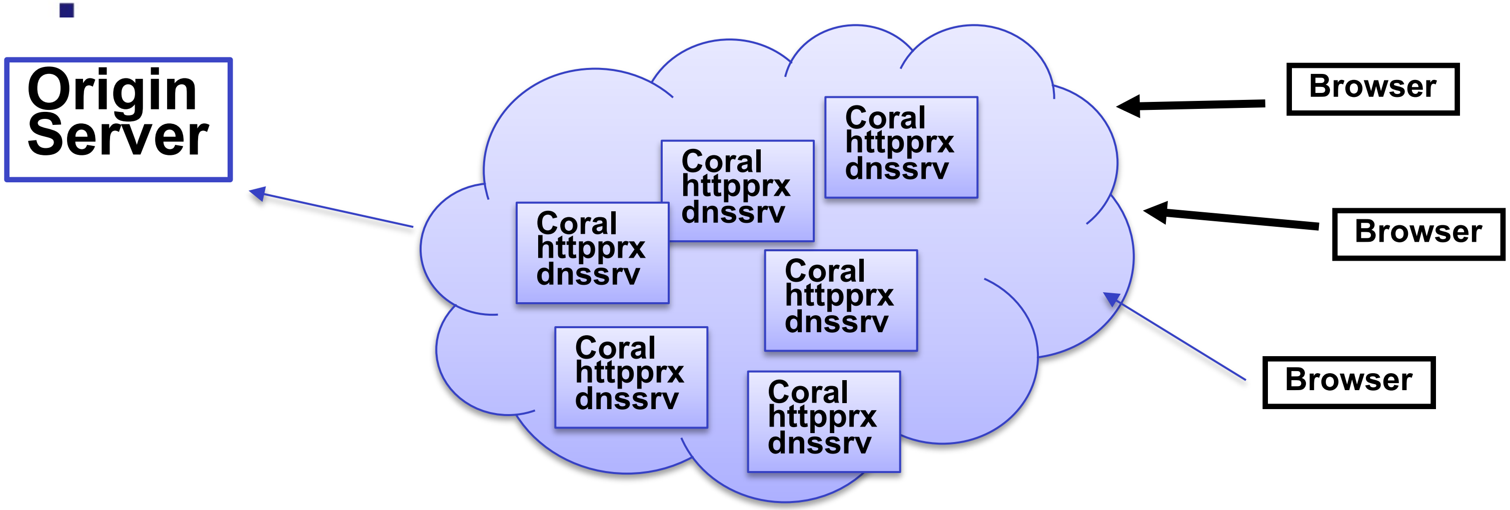
- Uses aXYZ name and consistent hash

How Akamai Works



Source: www.cs.cmu.edu/~srini/15-744/S08/lectures/17-DNS.ppt

Coral: An Open CDN



Pool resources to dissipate flash crowds

Implement an open CDN

Allow anybody to contribute

Works with unmodified clients

CDN only fetches once from origin server

Runs in PlanetLab

Based on NSDI 2004 presentation and paper

Using CoralCDN

Rewrite URLs into "Coralized" URLs

www.x.com → www.x.com.nyud.net:8090

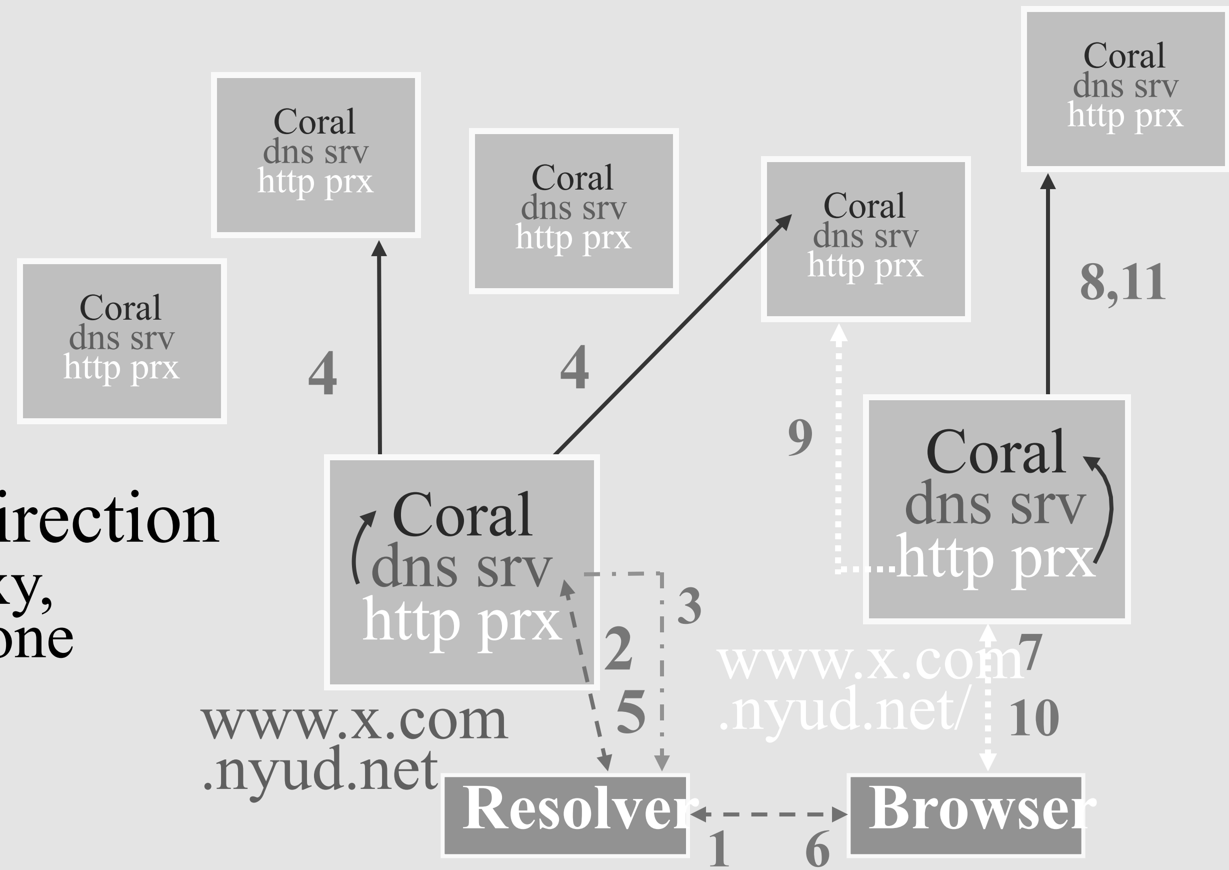
- Coral distributes the load

Who might "Coralize" URLs?

- Web server operators Coralize URLs
- Coralized URLs posted to portals, mailing lists
- Users explicitly Coralize URLs

DNS Redirection

Return proxy, preferably one near client



Cooperative Web Caching

Coral Server Discovery

- Each Coral server inserts its IP network prefix as key, its IP address as value
  - DNS server does DHT lookup on client IP prefix to find nearby Coral server
- Each Coral server uses traceroute to find nearby routers
  - Registers itself under IP of each nearby router
  - Coral DNS server traceroutes to client
    - Looks up each router IP address in mapping

Hierarchical DHT

A hierarchy of DHTs, with clustering at lower levels
- DHT based on XOR metric

- Nearby (< 20 ms) Coral nodes form an L2 DHT
  - L1: 60 ms
  - L0: global
- Search in L2 DHT first
  - If nearby copy exists, will find it first
  - Only search L1, L0 if miss in lower level

Finding URLs

Look up the URL in a DHT

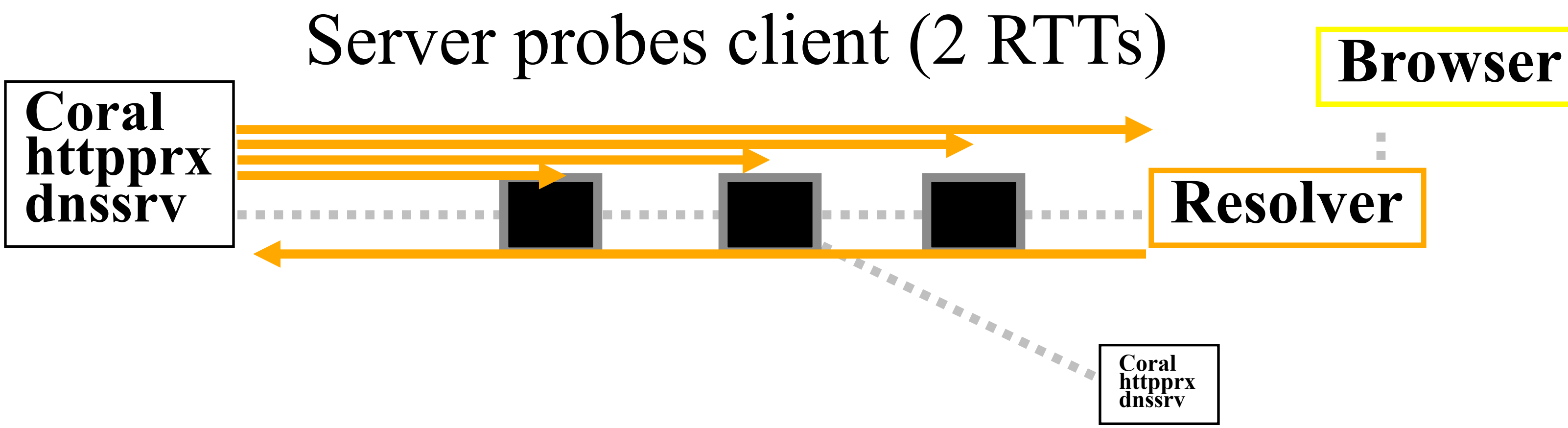
key=URL, value=IP addr of Coral cache that has the URL

Coral cache fetches the page from that other cache

If DHT had more than one value for key, fetch page from more than one

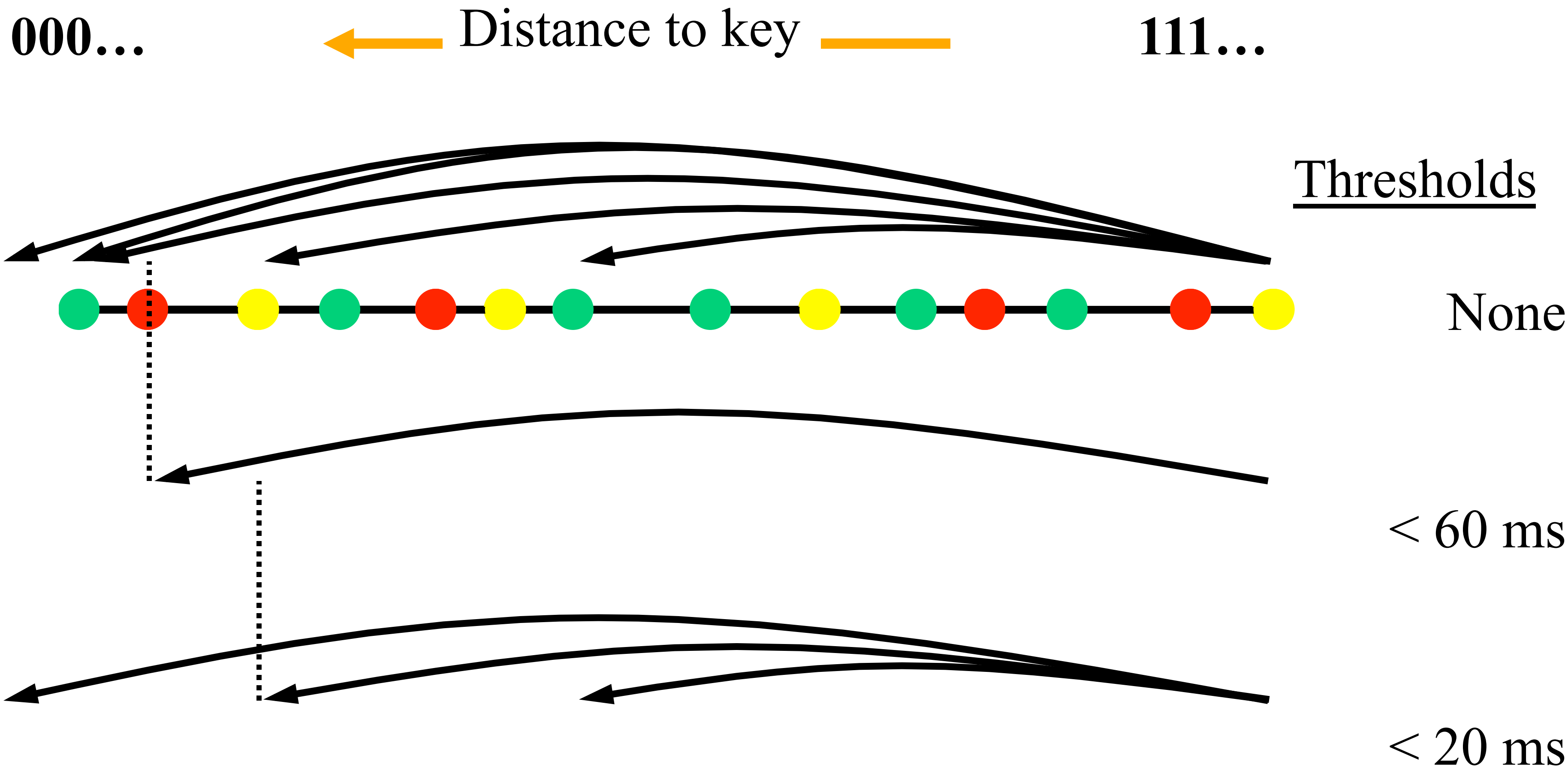
In case one is down or slow

DNS measurement mechanism



- Return servers within appropriate cluster
  - e.g., for resolver RTT = 19 ms, return from cluster < 20 ms
- Use network hints to find nearby servers
  - i.e., client and server on same subnet
- Otherwise, take random walk within cluster

Key-based XOR routing

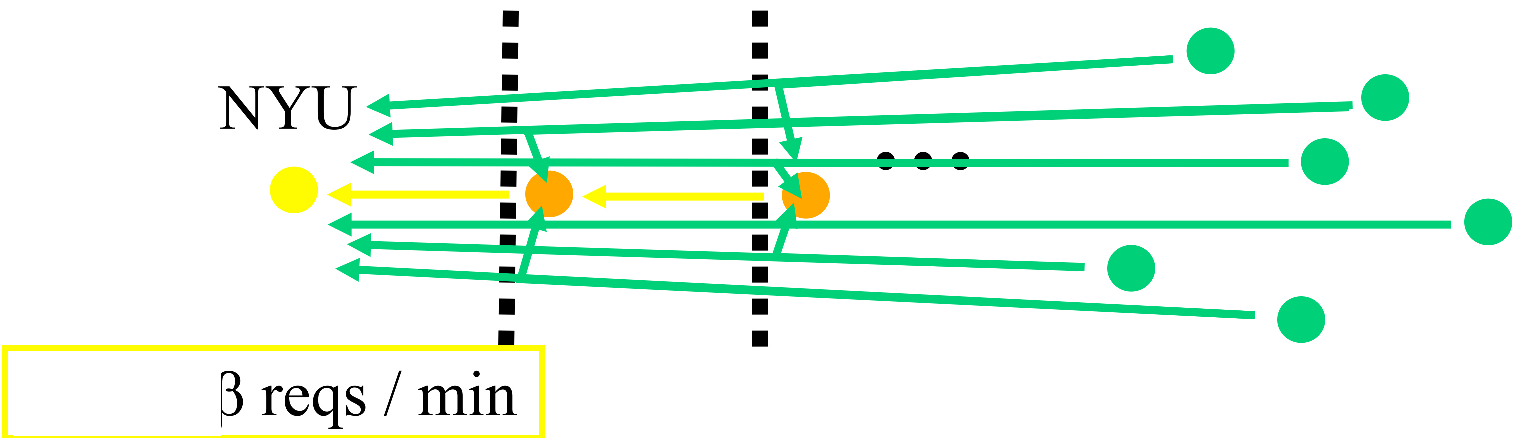


Minimizes lookup latency

Prefer values stored by nodes within faster clusters

Prevent insertion hotspots

- Store value once in each level cluster
  - Always storing at closest node causes hotspot



Halt put routing at full and loaded node

Full → M vals/key with TTL > ½ insertion TTL

Loaded → β puts traverse node in past minute

Store at furthest, non-full node seen

Challenges for DNS Redirection

Coral lacks…

- Central management
- *A priori* knowledge of network topology
  - Anybody can join system
- Any special tools (e.g., BGP feeds)

Coral has…

- Large number of vantage points to probe topology
- Distributed index in which to store network hints
- Each Coral node maps nearby networks to self

Coral's DNS Redirection

Coral DNS server probes resolver

Once local, stay local

- When serving requests from nearby DNS resolver
  - Respond with nearby Coral proxies
  - Respond with nearby Coral DNS servers
    - → Ensures future requests remain local

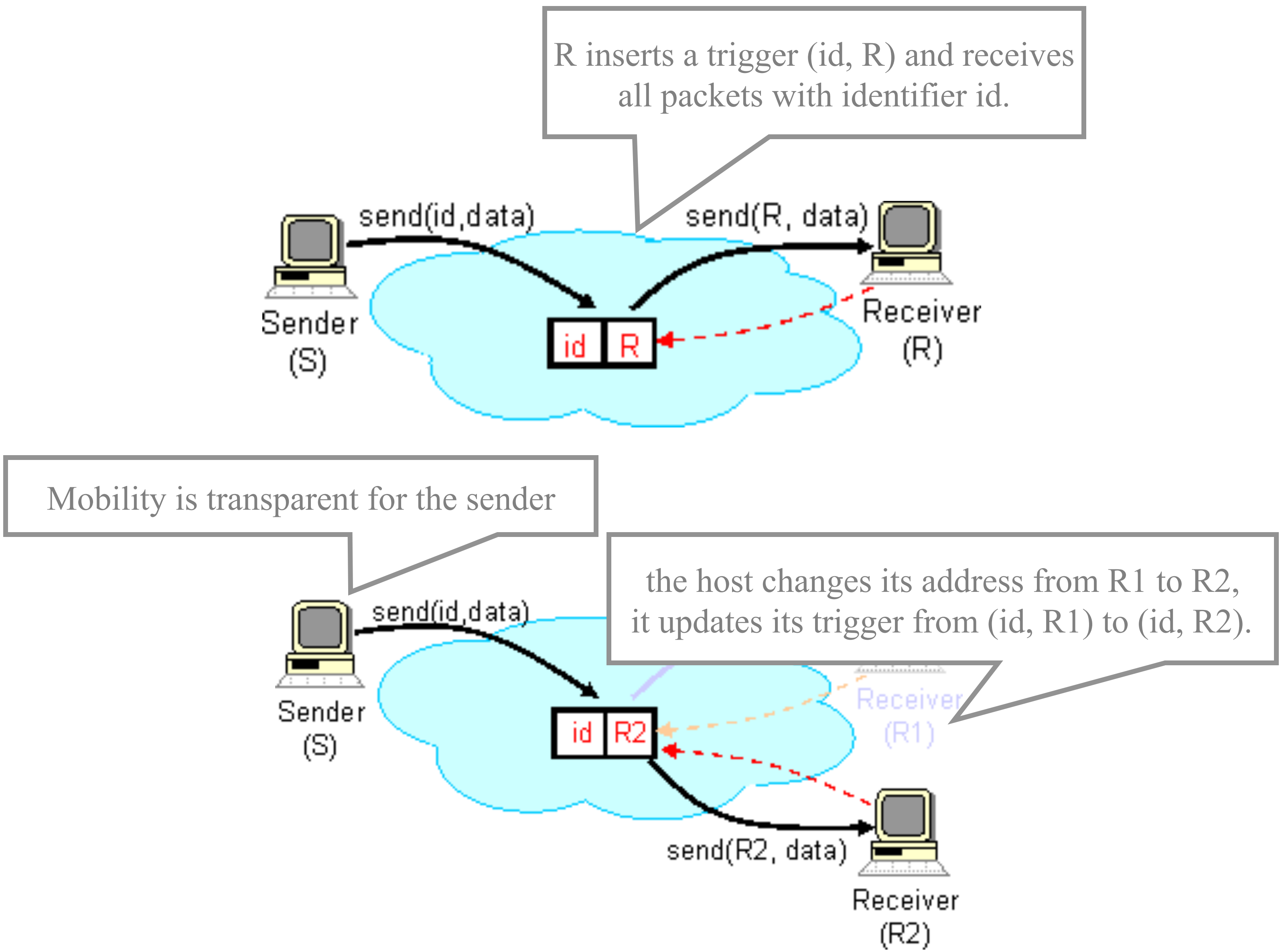
Else, help resolver find local Coral DNS server

Internet Indirection Infrastructure (i3)

- A DHT - based overlay network
  - Based on Chord
- Aims to provide more flexible communication model than current IP addressing
- Also a forwarding infrastructure
  - i3 packets are sent to identifiers
  - each identifier is routed to the i3 node responsible for that identifier
  - the node maintains triggers that are installed by receivers
  - when a matching trigger is found the packet is forwarded to the receiver

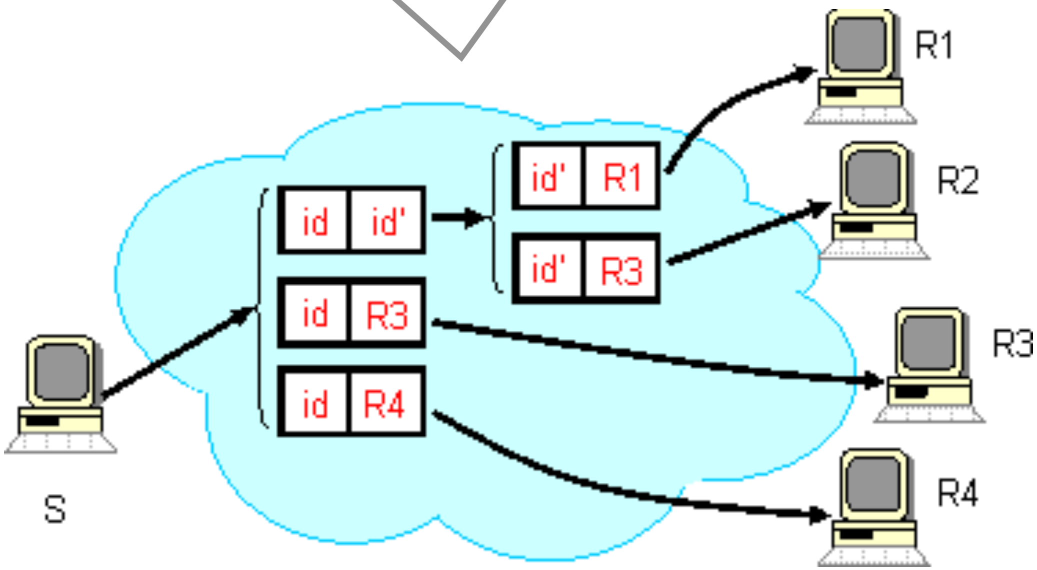
i3 II

- An i3 identifier may be bound to a host, object, or a session
- i3 has been extended with ROAM
  - Robust Overlay Architecture for Mobility
  - Allows end hosts to control the placement of rendezvous-points (indirection points) for efficient routing and handovers
  - Legacy application support
    - user level proxy for encapsulating IP packets to i3 packets



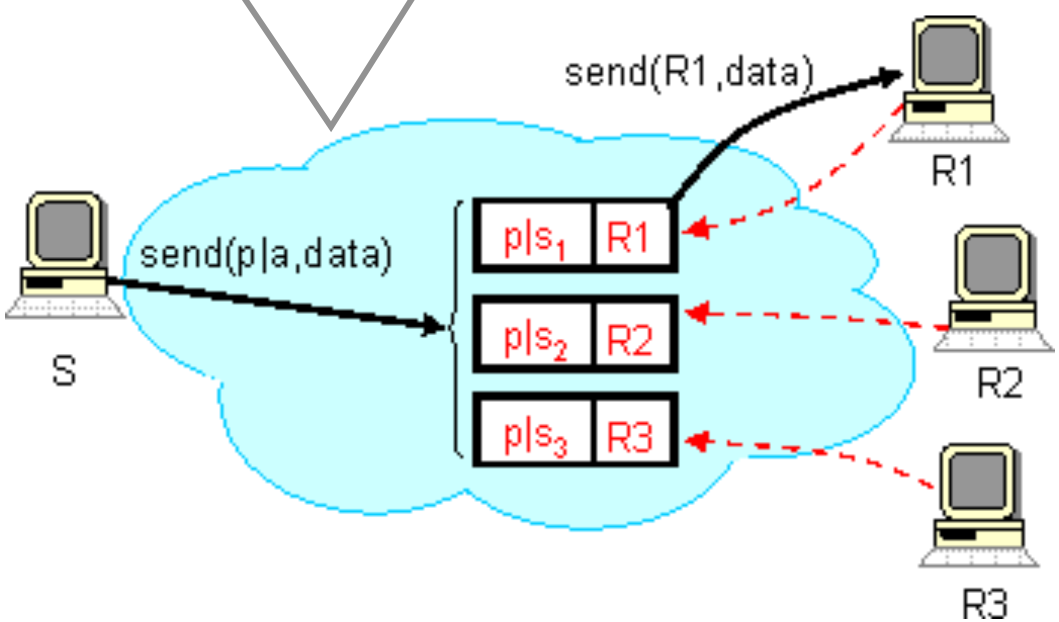
Source: http://i3.cs.berkeley.edu/

A multicast tree using a hierarchy of triggers



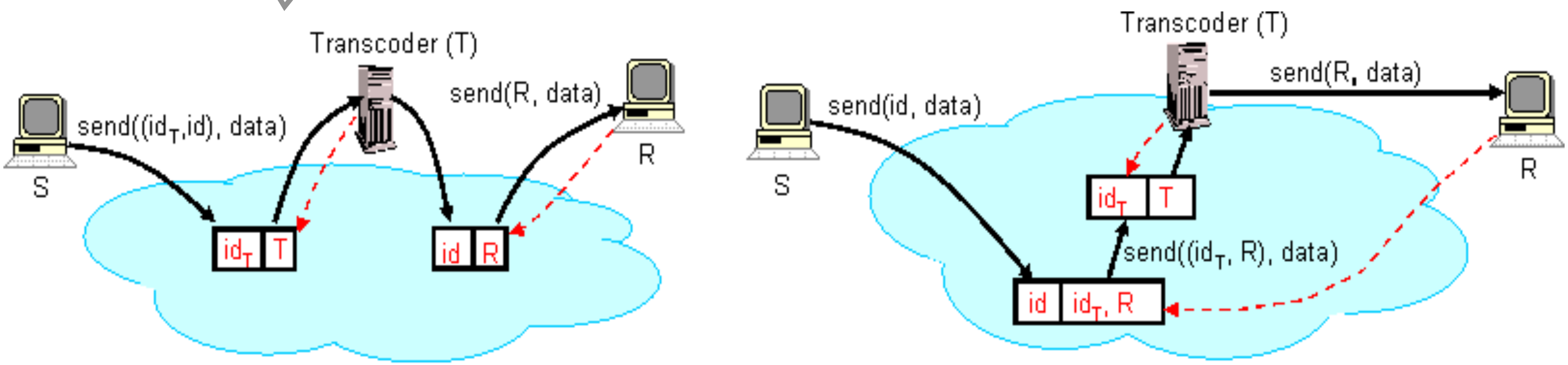
Source: http://i3.cs.berkeley.edu/

Anycast using the longest matching prefix rule.



Source: http://i3.cs.berkeley.edu/

Sender-driven service composition using a stack of identifiers



(a) Sender-driven service composition

(b) Receiver-driven service composition

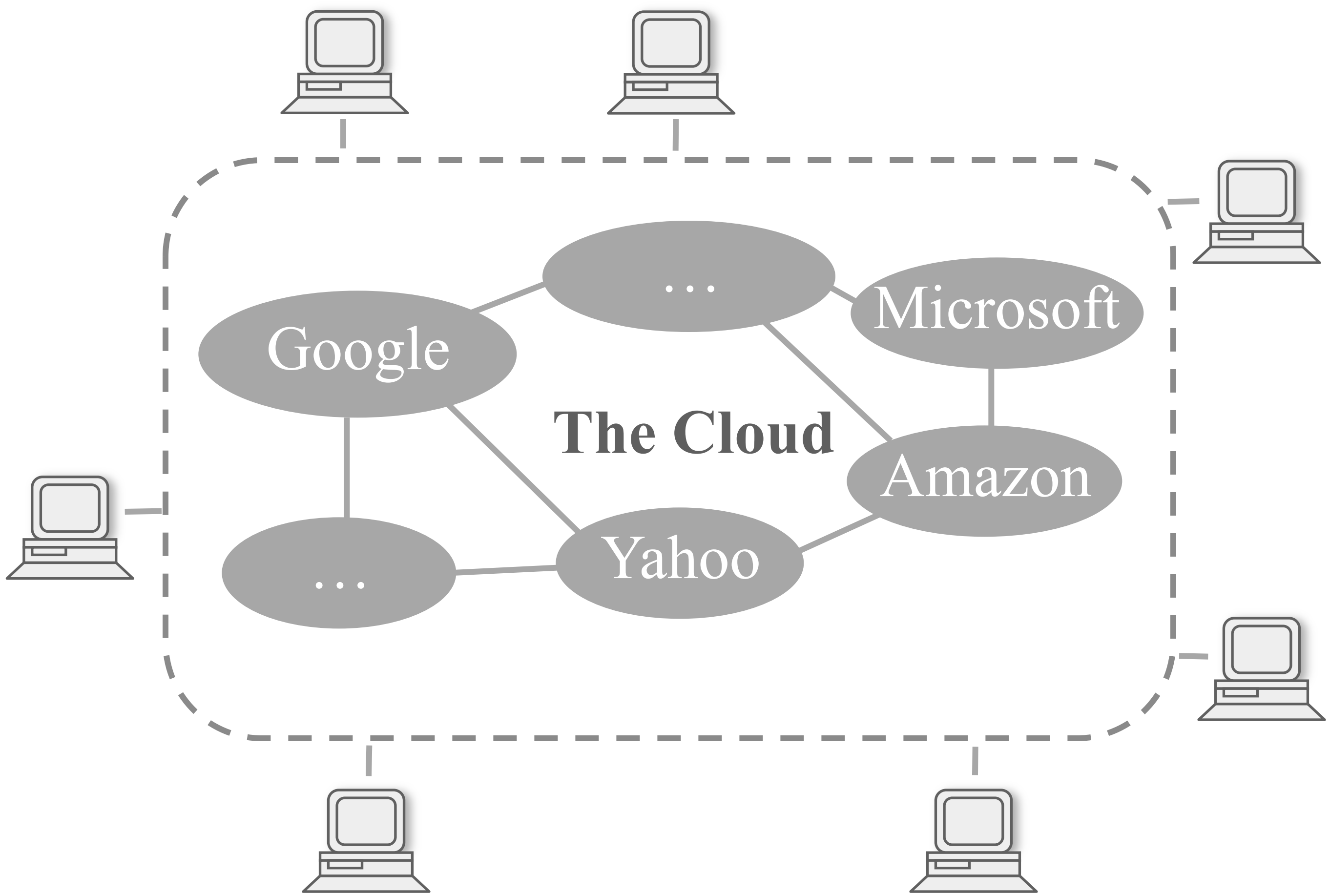
Receiver-driven service composition using a stack of identifiers

Source: http://i3.cs.berkeley.edu/

Summary

Key applications

- Kademlia and Mainline DHT (XOR geometry)
- PAST and SCRIBE (Pastry)
- Akamai (consistent hashing)
- Amazon (Dynamo, consistent hashing, ring geometry)
- Coral (XOR geometry)


Google
…
Microsoft
The Cloud
Amazon
…
Yahoo