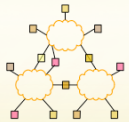
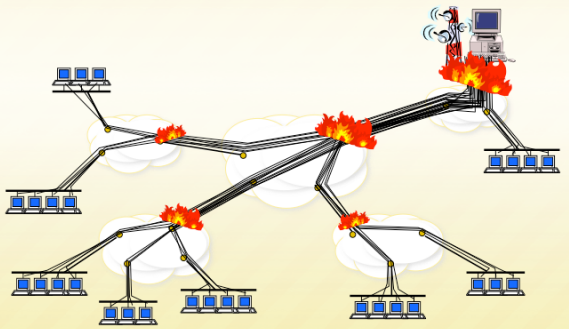
15-446 Distributed Systems Spring 2009



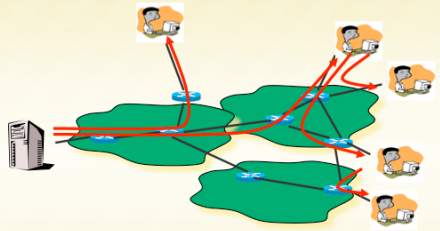
L-19 P2P

Scaling Problem

- Millions of clients ⇒ server and network meltdown



P2P System



- Leverage the resources of client machines (peers)
  - Computation, storage, bandwidth

Why p2p?

- Harness lots of spare capacity
  - 1 Big Fast Server: 1Gbit/s, $10k/month++
  - 2,000 cable modems: 1Gbit/s, $??
  - 1M end-hosts: Uh, wow.
- Build self-managing systems / Deal with huge scale
  - Same techniques attractive for both companies / servers / p2p
    - E.g., Akamai's 14,000 nodes
    - Google's 100,000+ nodes

Outline

- p2p file sharing techniques
  - Downloading: Whole-file vs. chunks
  - Searching
    - Centralized index (Napster, etc.)
    - Flooding (Gnutella, etc.)
    - Smarter flooding (KaZaA, …)
    - Routing (Freenet, etc.)
- Uses of p2p - what works well, what doesn't?
  - servers vs. arbitrary nodes
  - Hard state (backups!) vs soft-state (caches)
- Challenges
  - Fairness, freeloading, security, …

5

P2p file-sharing

- Quickly grown in popularity
  - Dozens or hundreds of file sharing applications
  - 35 million American adults use P2P networks -- 29% of all Internet users in US!
  - Audio/Video transfer now dominates traffic on the Internet

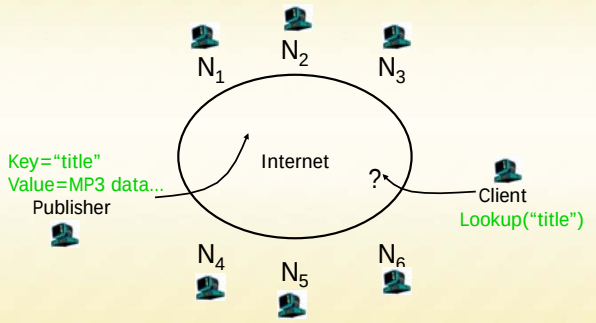
6

What's out there?

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs

7

Searching



8

Searching 2

- Needles vs. Haystacks
  - Searching for top 40, or an obscure punk track from 1981 that nobody's heard of?
- Search expressiveness
  - Whole word? Regular expressions? File names? Attributes? Whole-text search?
    - (e.g., p2p gnutella or p2p google?)

9

Framework

- Common Primitives:
  - **Join**: how to I begin participating?
  - **Publish**: how do I advertise my file?
  - **Search**: how to I find a file?
  - **Fetch**: how to I retrieve a file?

10

Outline

- Centralized Database
  - Napster
- Query Flooding
  - Gnutella
  - KaZaA
- Swarming
  - BitTorrent
- Unstructured Overlay Routing
  - Freenet
- Structured Overlay Routing
  - Distributed Hash Tables

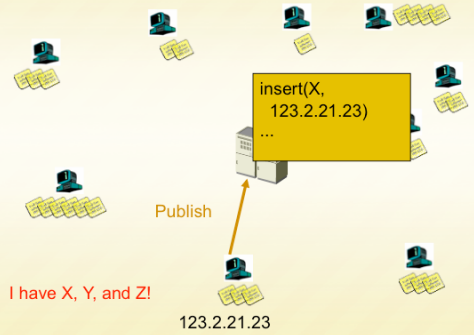
11

Napster

- History
  - 1999: Sean Fanning launches Napster
  - Peaked at 1.5 million simultaneous users
  - Jul 2001: Napster shuts down
- 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

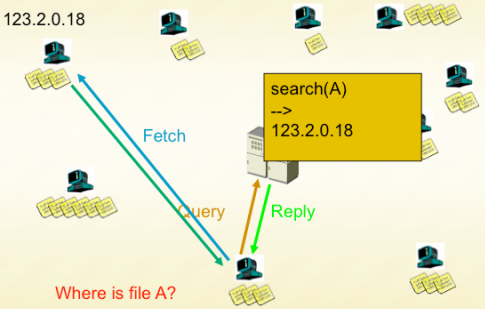
12

Napster: Publish



13

Napster: Search



14

Napster: Discussion

- Pros:
  - Simple
  - Search scope is O(1)
  - Controllable (pro or con?)
- Cons:
  - Server maintains O(N) State
  - Server does all processing
  - Single point of failure

15

Outline

- Centralized Database
  - Napster
- Query Flooding
  - Gnutella
  - KaZaA
- Swarming
  - BitTorrent
- Unstructured Overlay Routing
  - Freenet
- Structured Overlay Routing
  - Distributed Hash Tables

16

Gnutella

- History
  - In 2000, J. Frankel and T. Pepper from Nullsoft released Gnutella
  - Soon many other clients: Bearshare, Morpheus, LimeWire, etc.
  - In 2001, many protocol enhancements including "ultrapeers"
- 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

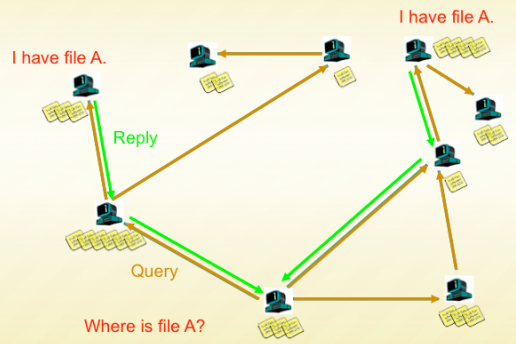
17

Gnutella: Overview

- 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

18

Gnutella: Search



19

Gnutella: Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
  - Processing @ each node permits powerful search semantics
- Cons:
  - Search scope is O(*N*)
  - Search time is O(???)
  - Nodes leave often, network unstable
- TTL-limited search works well for haystacks.
  - For scalability, does NOT search every node. May have to re-issue query later

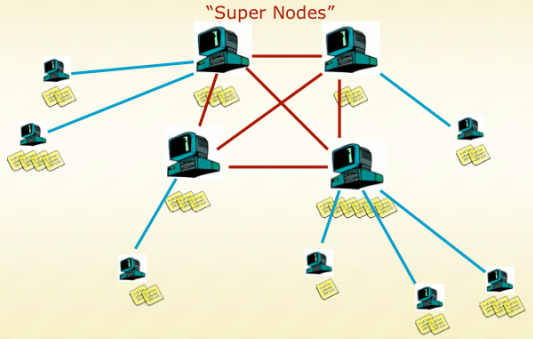
20

KaZaA

- History
  - In 2001, KaZaA created by Dutch company Kazaa BV
  - Single network called FastTrack used by other clients as well: Morpheus, giFT, etc.
  - Eventually protocol changed so other clients could no longer talk to it
- "Supernode" Query Flooding:
  - Join: on startup, client contacts a "supernode" ... may at some point become one itself
  - Publish: send list of files to supernode
  - Search: send query to supernode, supernodes flood query amongst themselves.
  - Fetch: get the file directly from peer(s); can fetch simultaneously from multiple peers

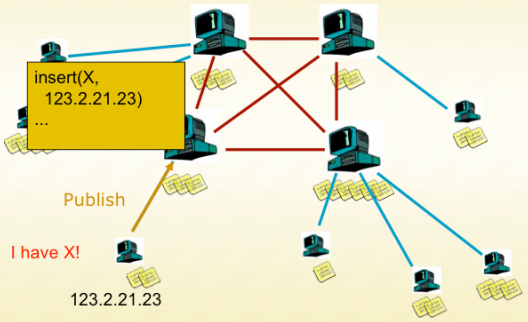
21

KaZaA: Network Design



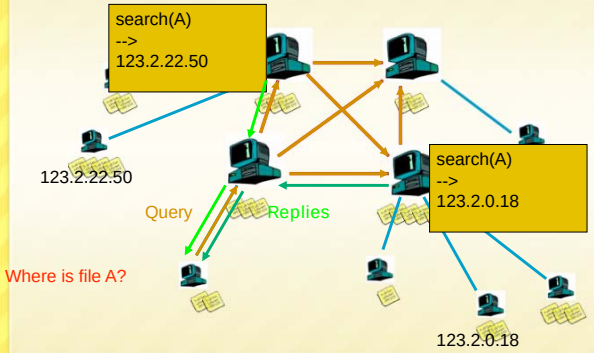
22

KaZaA: File Insert



23

KaZaA: File Search



24

KaZaA: Fetching

- More than one node may have requested file...
- How to tell?
  - Must be able to distinguish identical files
  - Not necessarily same filename
  - Same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - Alternatives: MD5, SHA-1
- How to fetch?
  - Get bytes [0..1000] from A, [1001...2000] from B
  - Alternative: Erasure Codes

25

KaZaA: Discussion

- Pros:
  - Tries to take into account node heterogeneity:
    - Bandwidth
    - Host Computational Resources
    - Host Availability (?)
  - Rumored to take into account network locality
- Cons:
  - Mechanisms easy to circumvent
  - Still no real guarantees on search scope or search time
- Similar behavior to gnutella, but better.

26

Stability and Superpeers

- Why superpeers?
  - Query consolidation
    - Many connected nodes may have only a few files
    - Propagating a query to a sub-node would take more b/w than answering it yourself
  - Caching effect
    - Requires network stability
- Superpeer selection is time-based
  - How long you've been on is a good predictor of how long you'll be around.

27

Outline

- Centralized Database
  - Napster
- Query Flooding
  - Gnutella
  - KaZaA
- Swarming
  - BitTorrent
- Unstructured Overlay Routing
  - Freenet
- Structured Overlay Routing
  - Distributed Hash Tables

28

BitTorrent: History

- In 2002, B. Cohen debuted BitTorrent
- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - E.g., Slashdot effect, CNN on 9/11, new movie/game release
- Focused on Efficient *Fetching*, not *Searching*:
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
- Has some "real" publishers:
  - Blizzard Entertainment using it to distribute the beta of their new game

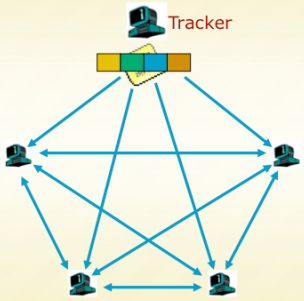
29

BitTorrent: Overview

- Swarming:
  - **Join**: contact centralized "tracker" server, get a list of peers.
  - **Publish**: Run a tracker server.
  - **Search**: Out-of-band. E.g., use Google to find a tracker for the file you want.
  - **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.
- Big differences from Napster:
  - Chunk based downloading
  - "few large files" focus
  - Anti-freeloading mechanisms

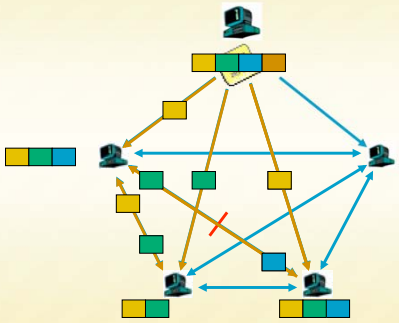
30

BitTorrent: Publish/Join



31

BitTorrent: Fetch



32

BitTorrent: Sharing Strategy

- Employ "Tit-for-tat" sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from him
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate
  - Let N peop
- Goal: Pareto Efficiency
  - Game Theory: "No change can make anyone better off without making others worse off"
  - Does it work? → lots of work on breaking/ improving this

33

BitTorrent: Summary

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Pareto Efficiency relatively weak
  - Central tracker server needed to bootstrap swarm

34

Outline

- Centralized Database
  - Napster
- Query Flooding
  - Gnutella
  - KaZaA
- Swarming
  - BitTorrent
- Unstructured Overlay Routing
  - Freenet
- Structured Overlay Routing
  - Distributed Hash Tables

35

Distributed Hash Tables: History

- Academic answer to p2p
- Goals
  - Guatanteed lookup success
  - Provable bounds on search time
  - Provable scalability
- Makes some things harder
  - Fuzzy queries / full-text search / etc.
- Read-write, not read-only
- Hot Topic in networking since introduction in ~2000/2001

36

DHT: Overview

- Abstraction: a distributed "hash-table" (DHT) data structure:
  - put(id, item);
  - item = get(id);
- Implementation: nodes in system form a distributed data structure
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...

37

DHT: Overview (2)

- Structured Overlay Routing:
  - **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

38

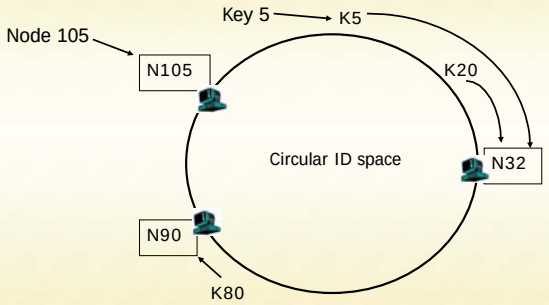
DHT: Example - Chord

- Associate to each node and file a unique *id* in an *uni*-dimensional space (a Ring)
  - E.g., pick from the range $[0...2^m]$
  - Usually the hash of the file or IP address
- Properties:
  - Routing table size is O(log *N*) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log *N*) hops

from MIT in 2001

39

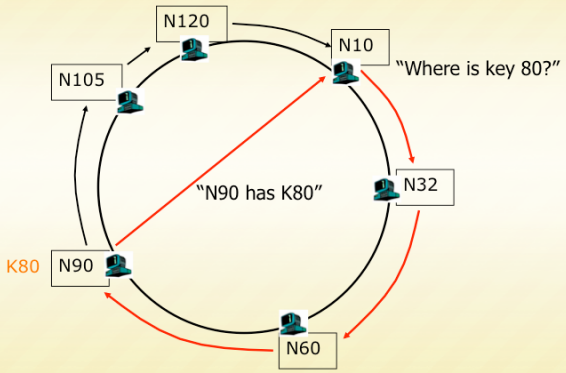
DHT: Consistent Hashing



A key is stored at its successor: node with next higher ID

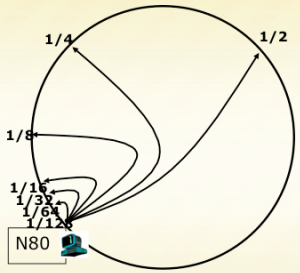
40

DHT: Chord Basic Lookup



41

DHT: Chord "Finger Table"

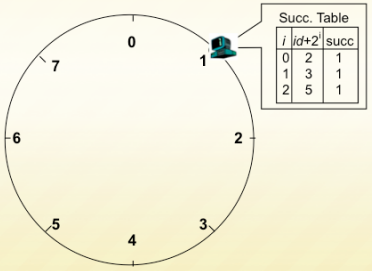


- Entry *i* in the finger table of node *n* is the first node that succeeds or equals $n + 2^i$
- In other words, the ith finger points $1/2^{n-i}$ way around the ring

42

DHT: Chord Join

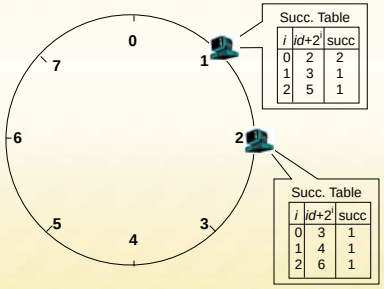
- Assume an identifier space [0..8]
- Node n1 joins



43

DHT: Chord Join

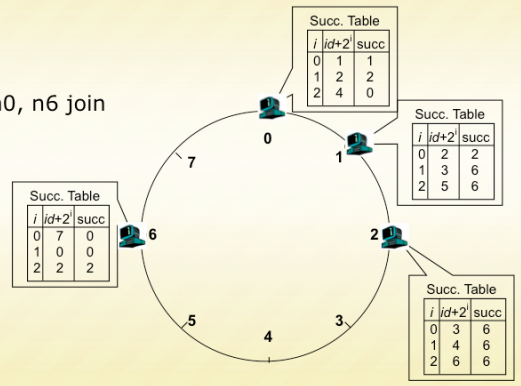
- Node n2 joins



44

DHT: Chord Join

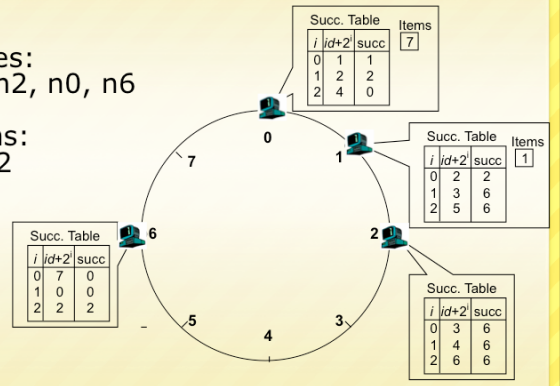
- Nodes n0, n6 join



45

DHT: Chord Join

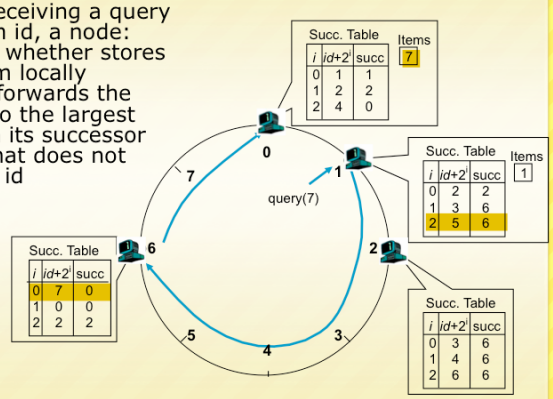
- Nodes:
  n1, n2, n0, n6
- Items:
  f7, f2



46

DHT: Chord Routing

- Upon receiving a query for item id, a node:
- Checks whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed id



47

DHT: Chord Summary

- Routing table size?
  - Log N fingers
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => expect O(log N) hops.
- Pros:
  - Guaranteed Lookup
  - O(log *N*) per node state and search scope
- Cons:
  - No one uses them? (only one file sharing app)
  - Supporting non-exact match search is hard

48

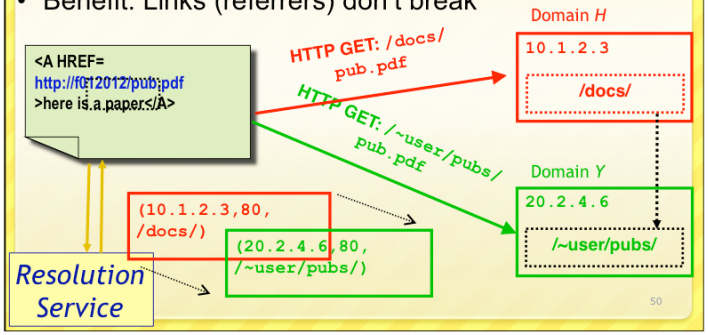
P2P-enabled Applications: Flat-Naming

- Most naming schemes use hierarchical names to enable scaling
- DHT provide a simple way to scale flat names
  - E.g. just insert name resolution into Hash(name)

49

Flat Names Example

- SID abstracts all object reachability information
- Objects: any granularity (files, directories)
- Benefit: Links (referrers) don't break



50

i3: Motivation

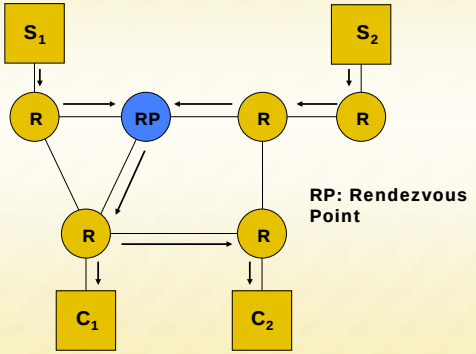
- Today's Internet based on point-to-point abstraction
- Applications need more:
  - Multicast
  - Mobility
  - Anycast

So, what's the problem?
A different solution for each service

- Existing solutions:
  - Change IP layer
  - Overlays

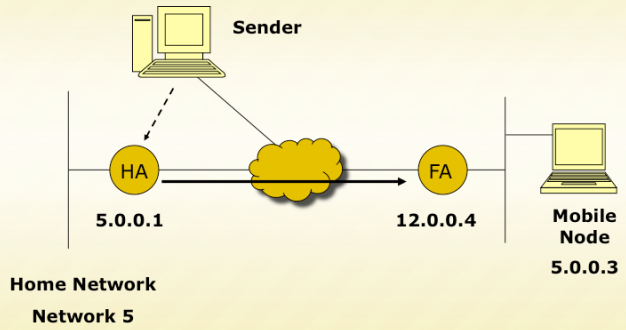
51

Multicast



52

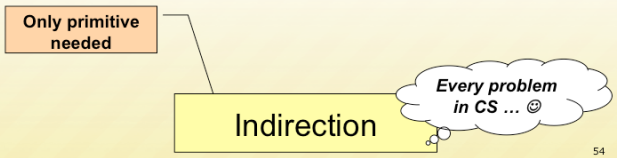
Mobility



53

The i3 solution

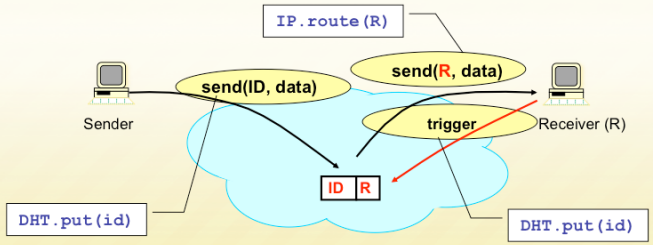
- Solution:
  - Add an indirection layer on top of IP
  - Implement using overlay networks
- Solution Components:
  - Naming using "identifiers"
  - Subscriptions using "triggers"
  - DHT as the gluing substrate



54

i3: Implementation

- Use a Distributed Hash Table
  - Scalable, self-organizing, robust
  - Suitable as a substrate for the Internet



55

P2P-enabled Applications: Distributed File Systems

56

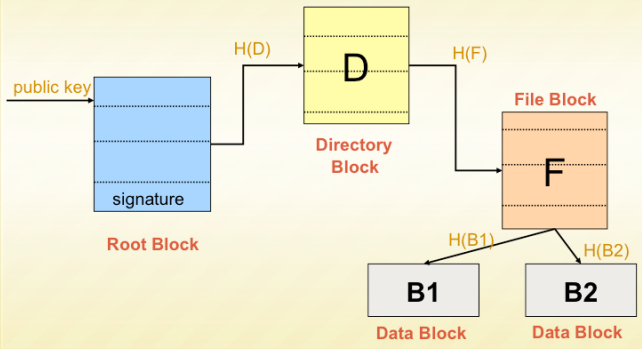
De-centralized file systems

- CFS [Chord]
  - *Block* based read-only storage
- PAST [Pastry]
  - *File* based read-only storage
- Ivy [Chord]
  - *Block* based read-write storage

CFS

- Blocks are inserted into Chord DHT
  - insert(blockID, block)
  - Replicated at successor list nodes
- Read root block through public key of file system
- Lookup other blocks from the DHT
  - Interpret them to be the file system
- Cache on lookup path

CFS



P2P-enabled Applications: Self-Certifying Names

- Name = Hash(pubkey, salt)
- Value = <pubkey, salt, data, signature>
  - can verify name related to pubkey and pubkey signed data
- Can receive data from caches or other 3rd parties without worry
  - much more opportunistic data transfer

When are p2p / DHTs useful?

- Caching and "soft-state" data
  - Works well! BitTorrent, KaZaA, etc., all use peers as caches for hot data
- Finding read-only data
  - Limited flooding finds hay
  - DHTs find needles
- BUT

61

A Peer-to-peer Google?

- Complex intersection queries ("the" + "who")
  - Billions of hits for each term alone
- Sophisticated ranking
  - Must compare many results before returning a subset to user
- Very, very hard for a DHT / p2p system
  - Need high inter-node bandwidth
  - (This is exactly what Google does - massive clusters)

62

Writable, persistent p2p

- Do you trust your data to 100,000 monkeys?
- Node availability (aka "churn") hurts
  - Ex: Store 5 copies of data on different nodes
  - When someone goes away, you must replicate the data they held
  - Hard drives are *huge*, but cable modem upload bandwidth is tiny - perhaps 10 Gbytes/day
  - Takes many days to upload contents of 200GB hard drive. Very expensive leave/replication situation!

63

P2P: Summary

- Many different styles; remember pros and cons of each
  - centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freeloading
  - Privacy and security are important
  - Structure can provide theoretical bounds and guarantees

64