

600.413 Topics in P2P Networked Systems

Week 3

Applications

Andreas Terzis

Slides from Ion Stoica, Robert Morris

Outline

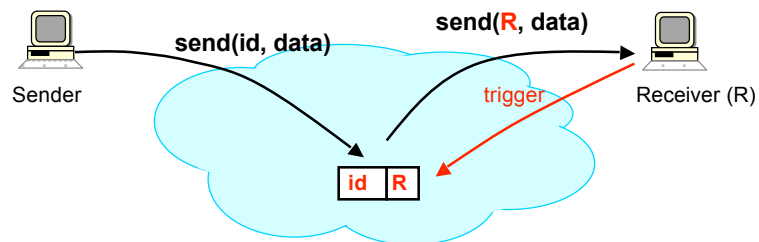
- **What we have covered so far**
 - First generation file-sharing systems (Napster, Gnutella)
 - Second generation DHT systems (Chord, CAN, Pastry)
- **This week: Applications built on top of DHTs**
 - Communication infrastructure (i3)
 - File Systems (CFS)
 - Security infrastructure (SOS)

The Problem

- Indirection: a key technique in implementing many network services, e.g.,
 - Mobility
 - Multicast, anycast
 - Web caching, replication, load-balancing
 - Anonymity
- IP doesn't provide efficient support for indirection → difficult and complex to deploy these services
- i3 approach: make indirection a first level design principle in network architecture

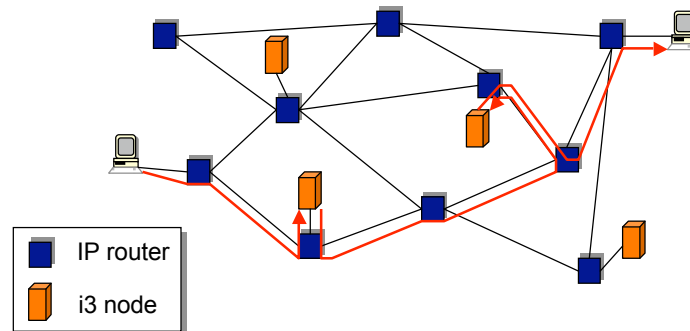
i3 Communication Abstraction

- Provide a rendezvous based communication abstraction (instead of point-to-point)
 - Each packet is associated an identifier *id*
 - To receive a packet with identifier *id*, receiver R maintains a **trigger** (*id*, R) into the overlay network



I3 Solution Outline

- Add an efficient indirection layer on top of IP
- Use an overlay network to implement it
 - Incrementally deployable; no need to change IP



Service Model

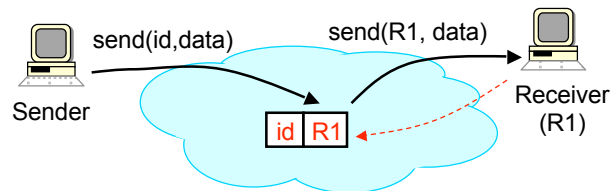
- API
 - `sendPacket(p);`
 - `insertTrigger(t);`
 - `removeTrigger(t) // optional`
- Best-effort service model (like IP)
- Triggers are periodically refreshed by end-hosts
- Reliability, congestion control, and flow-control implemented at end-hosts

The Promise

- Provide support for
 - Mobility
 - Multicast
 - Anycast
 - Service composition

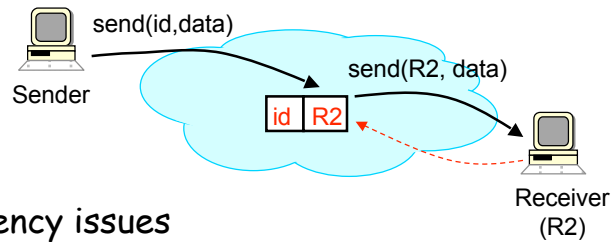
Mobility

- Host just needs to update its trigger as it moves from one subnet to another



Mobility

- Host just needs to update its trigger as moves from one subnet to another

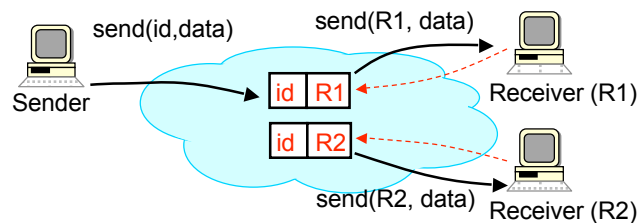


- Efficiency issues

- Cache IP address of i3 server responsible for the trigger
- Find an i3 server that is "close" to the end-points

Multicast

- Unifies multicast and unicast abstractions
 - Multicast: receivers insert triggers with the same identifier
- An application can dynamically switch between multicast and unicast



Matching generalizations

■ Inexact matching

- Trigger id_t matches identifier id if
 - id and id_t have a prefix match of at least k bits
 - There is no trigger that has a longer prefix match with id

■ Identifier stacks

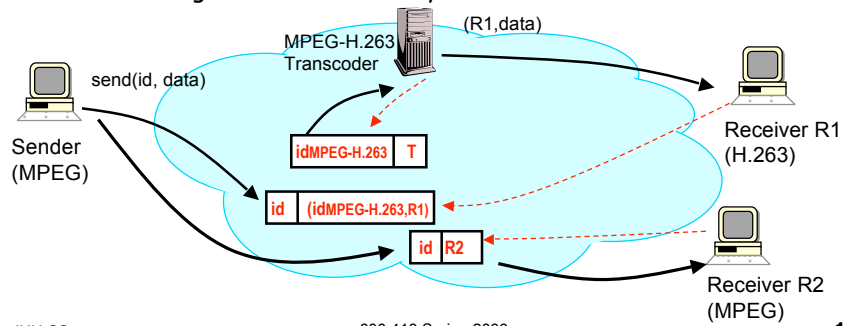
- Id stack = $(id_1, id_2, id_3, \dots, id_k)$ id_i is an identifier or an IP address
- Packet $p = (id_{stack}, data)$
- Trigger $t = (id, id_{stack})$

Matching generalizations (2)

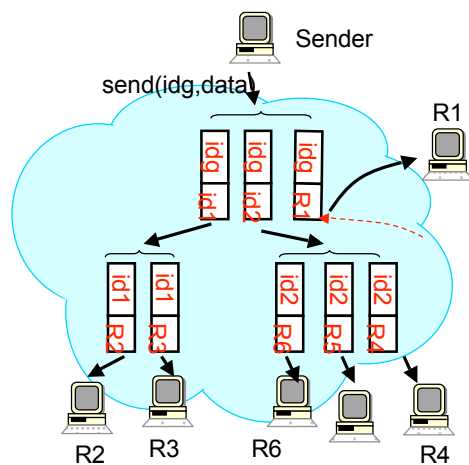
- Packet is forwarded based on id on top of the stack
- If id matches trigger (id, id_{stack}) then id is replaced by id_{stack} and forwarding continues

Service Composition

- Use a **stack of IDs** to encode the successions of operations to be performed on data
- Advantages
 - Don't need to configure path
 - Load balancing and robustness easy to achieve



Large Scale Multicast



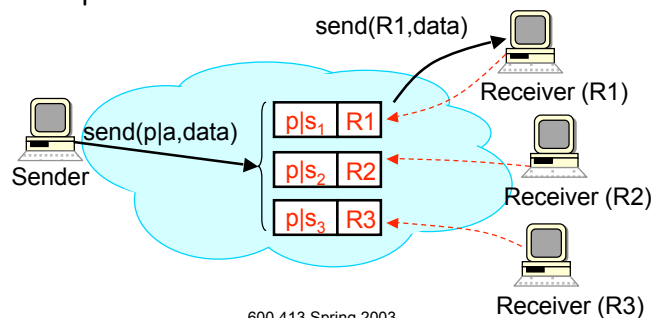
- Stack identifier is invisible to the sender
- How to pick the ids?

Anycast

- Generalize the matching scheme used to forward a packet
 - Until now we assumed exact matching
- Next, we assume:
 - Longest prefix matching (LPM) using a prefix larger than a predefined constant l to avoid collisions
 - In the current implementation: ID length, $m = 256$, $l = 128$

Anycast (cont'd)

- Anycast is simply a byproduct of the new matching scheme, e.g.,
 - Each receiver R_i in the anycast group inserts IDs with the same prefix p and a different suffix s_i
 - What to put in the suffix?

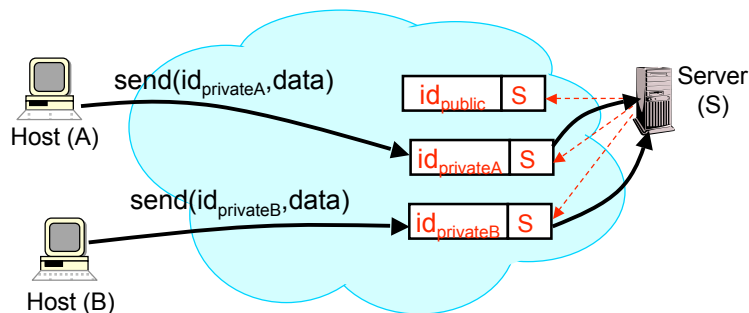


Desired Overlay Properties

- So far we have talked about the service model and services built on top of it
- How do we actually build it? How is it scalable and secure?
 - Robustness
 - Scalability
 - Efficiency
 - Stability

Public and Private Triggers

- Server maintains a public trigger id_{public}
 - Clients contact the server via its public trigger
- For each client i , the server allocates a private trigger
 - The private trigger is a **shared secret** between the server and client



Robustness

- Triggers are periodically refreshed
 - If not refreshed, triggers time out
 - If a trigger is lost it will be eventually updated
 - What happens in the meantime?
- Use a backup trigger id_{backup}
 - Packets are sent to id stack (id, id_{backup})
- Replicate triggers on the overlay network
 - For Chord replicate trigger on the immediate successor

Routing Efficiency

- i3 uses Chord for routing
- Sender caches the i3's server IP address
 - Can help efficiency (assumes alternate triggers)
- Caching does not affect correctness
 - If server moves from s to s' then s forwards to s' and update is sent to sender
- Still suffers from *triangle routing*
 - Receivers should select *private* triggers that are close to servers
 - Receivers inserts random triggers (id, A) and measures RTT to each of them

Scalability

- **Avoiding Hot Spots**
 - If rate > threshold server pushes copy of trigger to other server
 - In Chord case, push trigger to predecessor
- **#triggers (n) = #end-hosts (N)+ #flows**
- **Each server stores n/N triggers**

Security

- **Show that i3's flexibility can improve (not hurt) end-host security**
- **Redesign i3 to make it secure without compromising its flexibility and performance**

Key Observation

- To improve end-host security it is **necessary** to give end-hosts more control on routing and data forwarding in the infrastructure

Why?

- End-hosts are in the best position to **detect** when they are under attack
 - E.g., flash-crowd vs. DoS, SYN attack
- Once an end-host detects an attack, it should be able to **stop/redirect** the offending traffic before it arrives at its inbound connection
 - i3 gives end-host this flexibility

Attacks on Triggers

■ Public triggers

- Use public key crypto to exchange private triggers

■ Private triggers

- Very large trigger space, difficult to do a brute force attack
- Periodically change private triggers

■ Trigger hijacking

- Malicious user removes a host's public trigger
- Solution: Another level of indirection (!)
 - Rather than inserting (id,S) insert (id,x) and (x,S)

DoS Attacks

■ Attacks on end-hosts

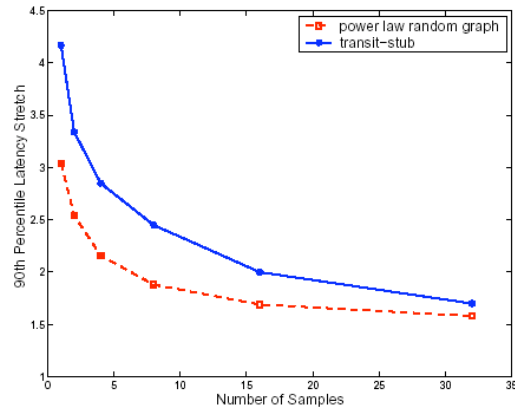
- Create a tree of triggers similar to the one in scalable multicast to target many victims
- Solution: use challenges

■ Attacks on infrastructure

- Create trigger loops (exponentially increase the number of packets)
- (Partial) Solution: Use Fair Queuing on the number of triggers
- Loop detection

Simulation Results

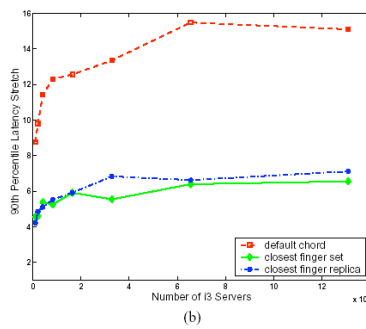
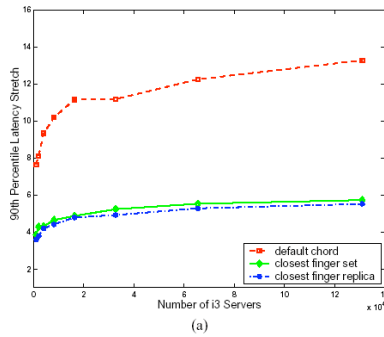
■ Latency Stretch



Simulation Results (2)

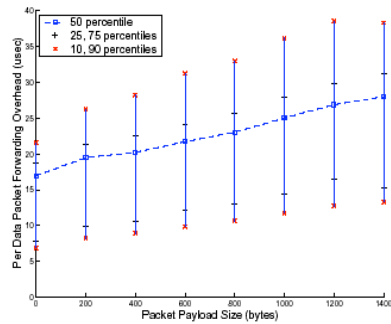
■ First packet stretch

- Standard Chord routing
- Proximity Routing

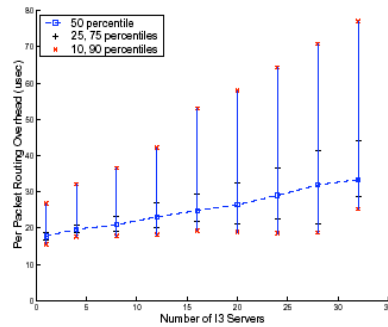


Performance Results

Packet Forwarding



Packet Routing



Conclusions

■ Indirection, key primitive to support

- Basic communication abstractions, e.g., multicast, anycast, mobility
- Improve end-host security

■ This research advocates for:

- Leaving IP do what is doing best: point-to-point unicast communication
- Building an efficient Indirection Layer on top of IP

Discussion

■ Positives

- Broad scope
- Clean primitives
- Solid design

■ Negatives

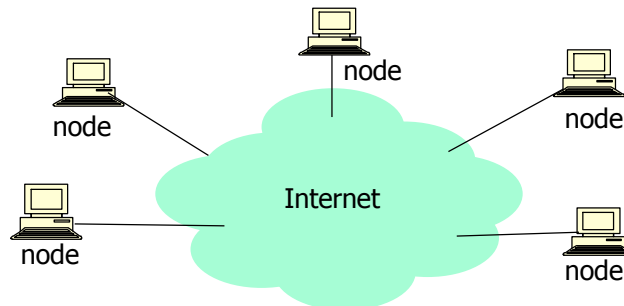
- Security is still an open issue
- Performance
- Programming model

Wide-Area Cooperative Storage with CFS

Robert Morris
Frank Dabek, M. Frans Kaashoek,
David Karger, Ion Stoica

MIT and Berkeley

Target CFS Uses



- **Serving data with inexpensive hosts:**
 - open-source distributions
 - off-site backups
 - tech report archive
 - efficient sharing of music

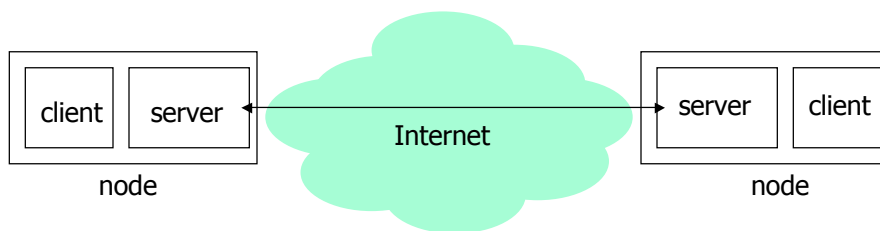
How to mirror open-source distributions?

- **Multiple independent distributions**
 - Each has high peak load, low average
- **Individual servers are wasteful**
- **Solution: aggregate**
 - Option 1: single powerful server
 - Option 2: distributed service
 - But how do you find the data?

Design Challenges

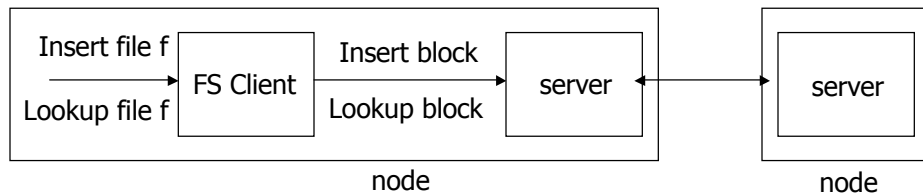
- Avoid hot spots
- Spread storage burden evenly
- Tolerate unreliable participants
- Fetch speed comparable to whole-file TCP
- Avoid $O(\#\text{participants})$ algorithms
 - Centralized mechanisms [Napster], broadcasts [Gnutella]
- CFS solves these challenges

CFS Architecture



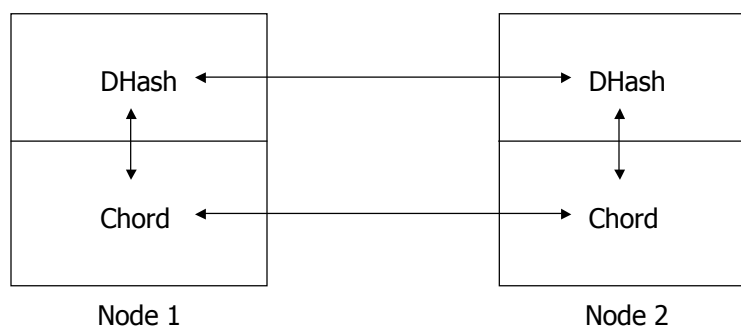
- Each node is a client and a server (like xFS)
- Clients can support different interfaces
 - File system interface
 - Music key-word search (like Napster and Gnutella)

Client-server interface



- Files have unique names
- Files are read-only (single writer, many readers)
- Publishers split files into blocks
- Clients check files for authenticity [SFSRO]

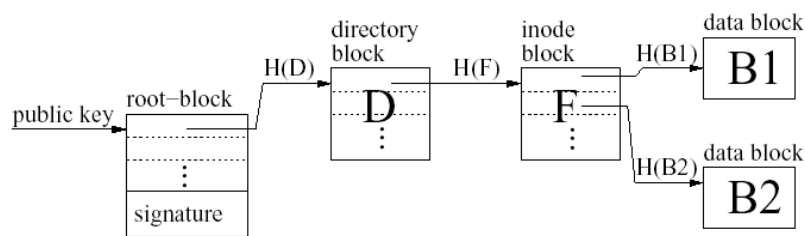
Server Structure



- DHash stores, balances, replicates, caches blocks
- DHash uses Chord [SIGCOMM 2001] to locate blocks

File System Structure

- Each block is data or FS meta-data
 - Parent block contains Ids of children
- Publisher uses content hash as the block's ID
- Publisher signs the root block with priv key

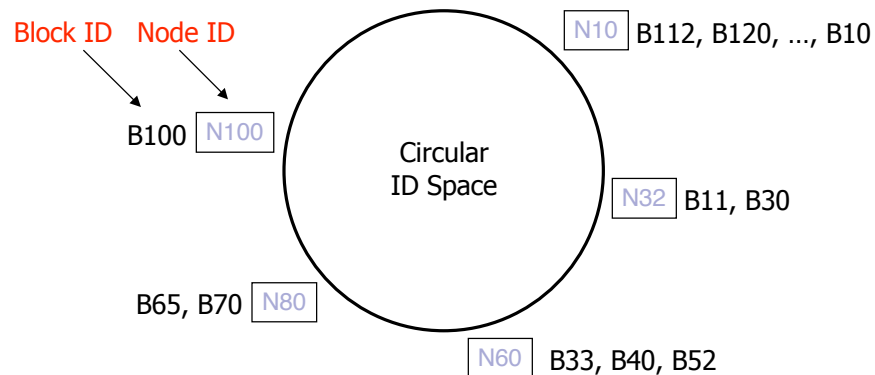


JHU CS

600.413 Spring 2003

39

Chord Hashes a Block ID to its *Successor*



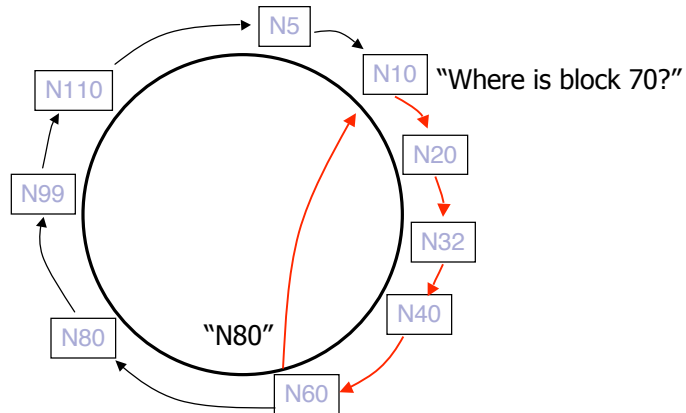
- Nodes and blocks have randomly distributed IDs
- **Successor: node with next highest ID**

JHU CS

600.413 Spring 2003

40

Basic Lookup



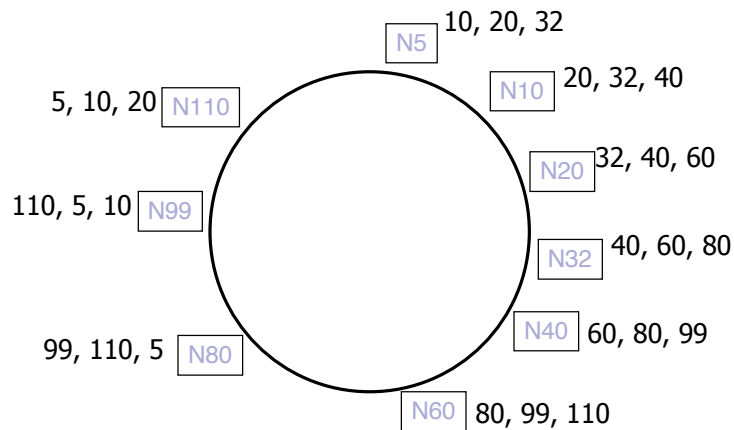
- Lookups find the ID's predecessor
- Correct if successors are correct

JHU CS

600.413 Spring 2003

41

Successor Lists Ensure Robust Lookup



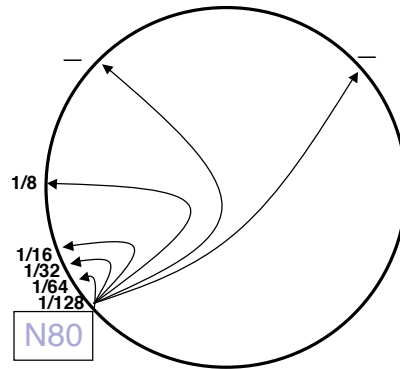
- Each node stores r successors, $r = 2 \log N$
- Lookup can skip over dead nodes to find blocks

JHU CS

600.413 Spring 2003

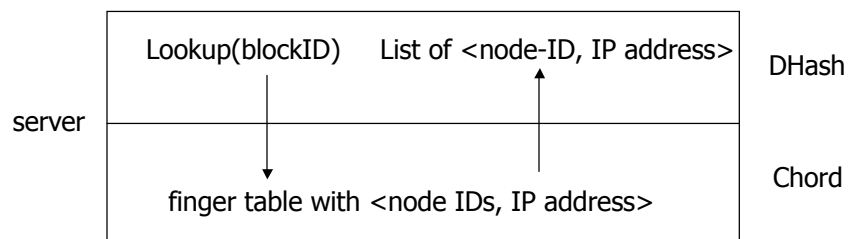
42

Chord Finger Table Allows $O(\log N)$ Lookups



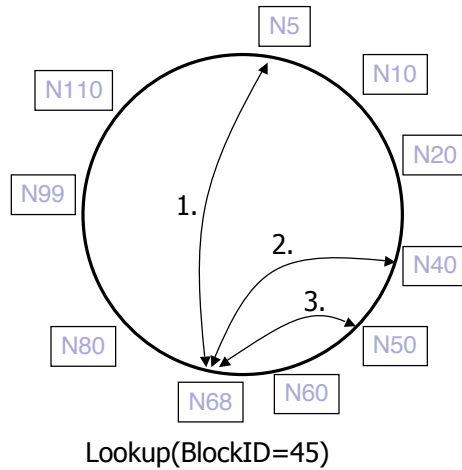
- See [SIGCOMM 2000] for table maintenance

DHash/Chord Interface



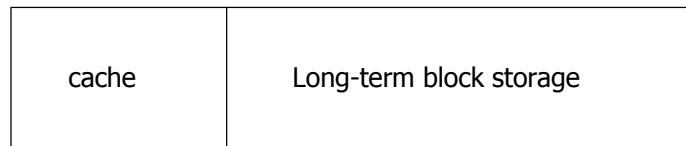
- *lookup()* returns list with node IDs closer in ID space to block ID
 - Sorted, closest first

DHash Uses Other Nodes to Locate Blocks



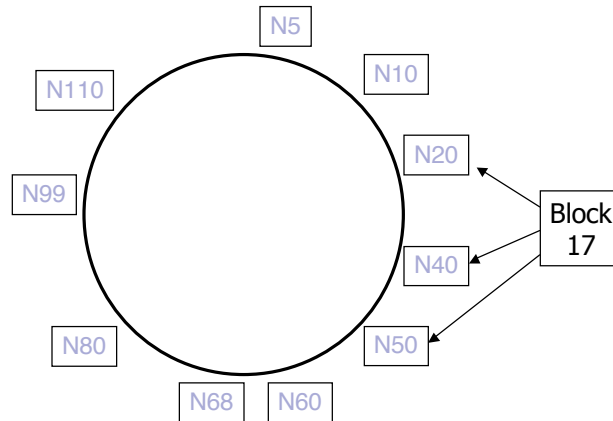
Storing Blocks

disk:



- Long-term blocks are stored for a fixed time
 - Publishers need to refresh periodically
- Cache uses LRU

Replicate blocks at r successors



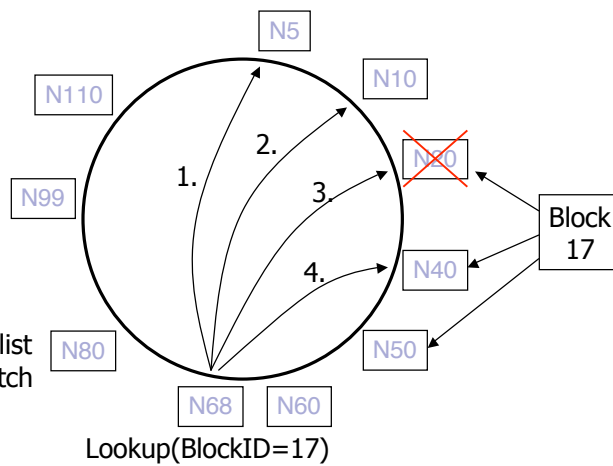
- Node IDs are SHA-1 of IP Address
- Ensures independent replica failure

JHU CS

600.413 Spring 2003

47

Lookups find replicas



RPCs:

1. Lookup step
2. Get successor list
3. Failed block fetch
4. Block fetch

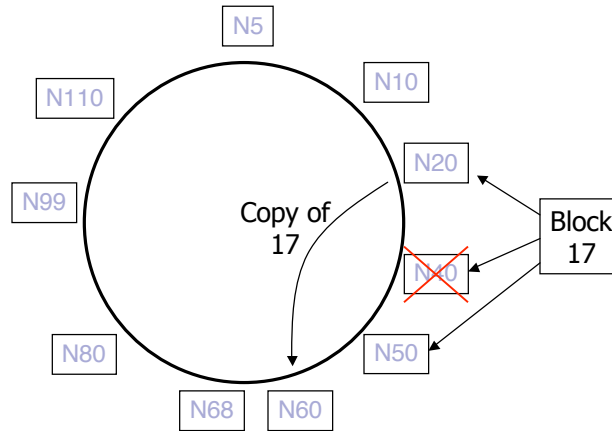
Lookup(BlockID=17)

JHU CS

600.413 Spring 2003

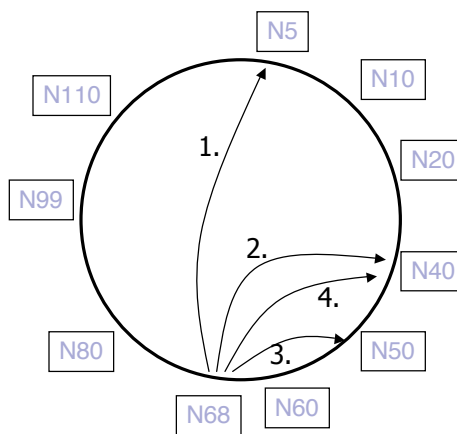
48

First Live Successor Manages Replicas



- Node can locally determine that it is the first live successor

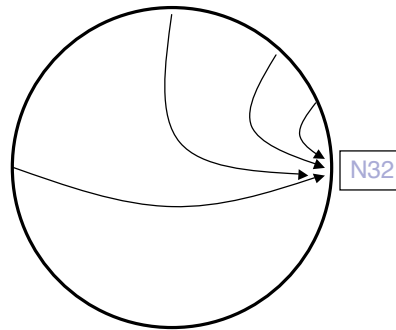
DHash Copies to Caches Along Lookup Path



- RPCs:
1. Chord lookup
 2. Chord lookup
 3. Block fetch
 4. Send to cache

Lookup(BlockID=45)

Caching at Fingers Limits Load



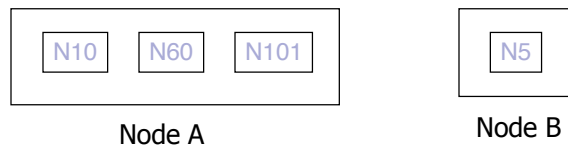
- Only $O(\log N)$ nodes have fingers pointing to N32
- This limits the single-block load on N32

JHU CS

600.413 Spring 2003

51

Virtual Nodes Allow Heterogeneity



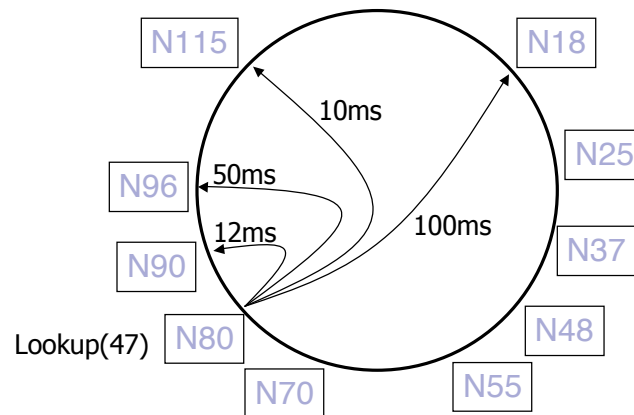
- Hosts may differ in disk/net capacity
- Hosts may advertise multiple IDs
 - Chosen as $\text{SHA-1}(\text{IP Address, index})$
 - Each ID represents a "virtual node"
- Host load proportional to # v.n.'s
- Manually controlled

JHU CS

600.413 Spring 2003

52

Fingers Allow Choice of Paths



- Each node monitors RTTs to its own fingers
- Tradeoff: ID-space progress vs delay

JHU CS

600.413 Spring 2003

53

Why Blocks Instead of Files?

- Cost: one lookup per block
 - Can tailor cost by choosing good block size
- Benefit: load balance is simple
 - For large files
 - Storage cost of large files is spread out
 - Popular files are served in parallel

JHU CS

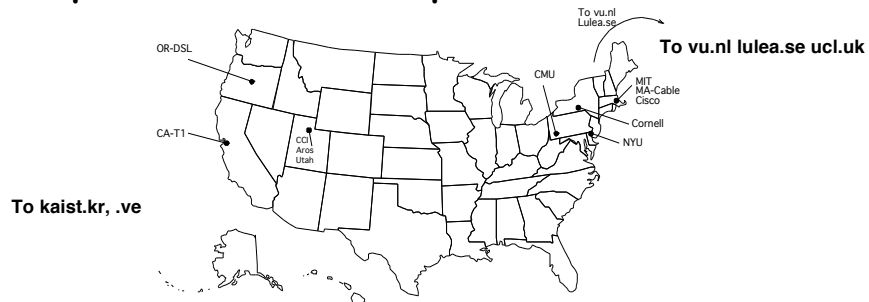
600.413 Spring 2003

54

CFS Project Status

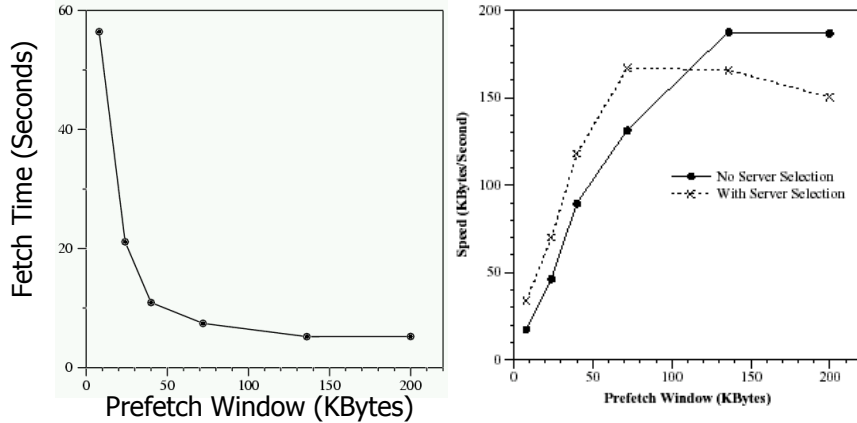
- Working prototype software
- Some abuse prevention mechanisms
- SFSRO file system client
 - Guarantees authenticity of files, updates, etc.
- Napster-like interface in the works
 - Decentralized indexing system
- Some measurements on RON testbed
- Simulation results to test scalability

Experimental Setup (12 nodes)



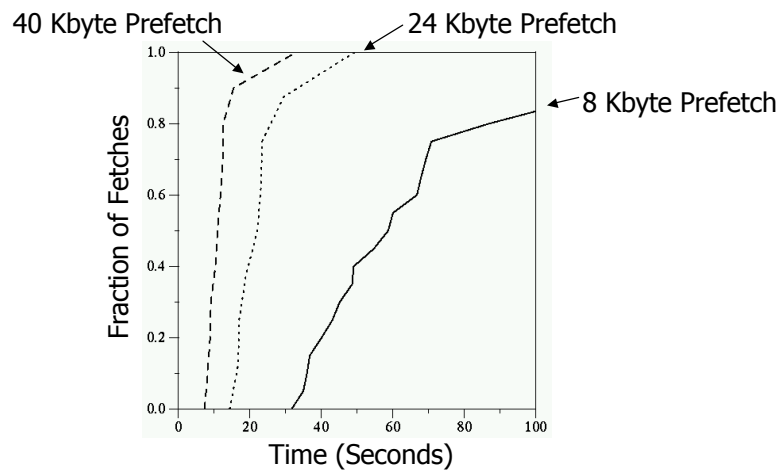
- One virtual node per host
- 8Kbyte blocks
- RPCs use UDP
- Caching turned off
- Proximity routing turned off

CFS Fetch Time for 1MB File

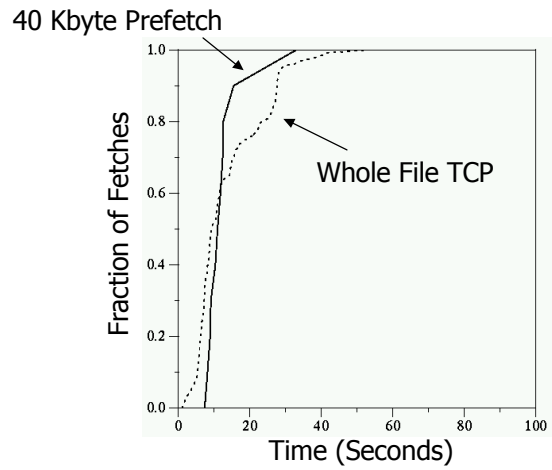


- Average over the 12 hosts
- No replication, no caching; 8 KByte blocks

Distribution of Fetch Times for 1MB



CFS Fetch Time vs. Whole File TCP

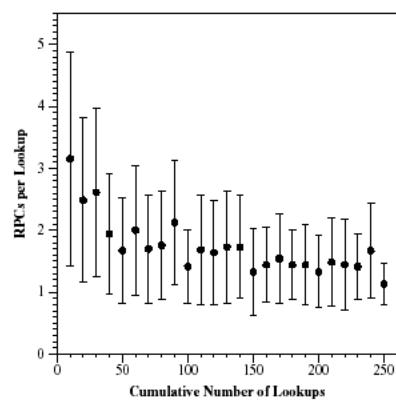
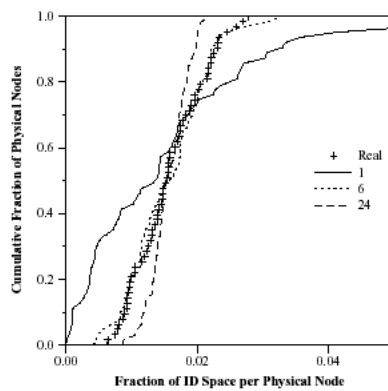


JHU CS

600.413 Spring 2003

59

Load balancing and replication

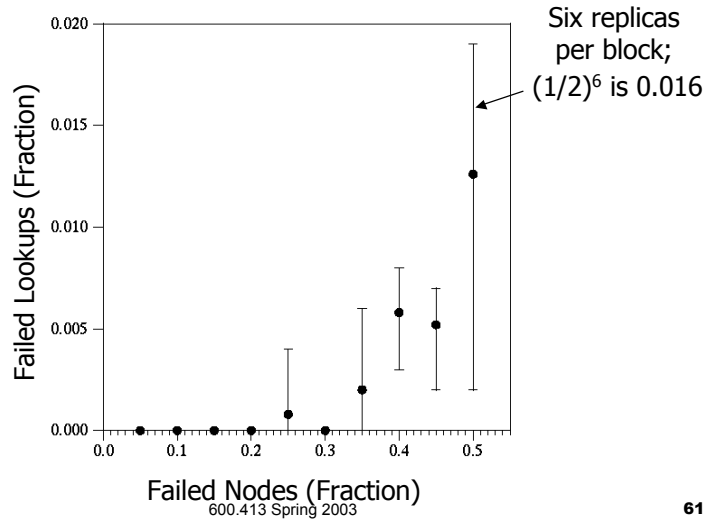


JHU CS

600.413 Spring 2003

60

Robustness vs. Failures



61

CFS Summary

- CFS provides peer-to-peer r/o storage
- Structure: DHash and Chord
- It is efficient, robust, and load-balanced
- It uses block-level distribution
- The prototype is as fast as whole-file TCP

<http://www.pdos.lcs.mit.edu/chord>

JHU CS

600.413 Spring 2003

62



CFS Evaluation

■ Positive

- (among the) first distributed file systems on top of DHTs
- Trusted file system from untrusted storage
- Use of blocks to reduce load and increase load balancing

■ Negative

- Read-Only
- Poor temporal locality
- Susceptible to adversarial attacks