CS 194: Distributed Systems
DHT Applications: What and Why

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

Project Phase III

- What: Murali will discuss Phase III of the project
- When: Tonight, 6:30pm
- Where: 306 Soda

Remaining Lecture Schedule

4/11	DHT applications (start)	(Scott)
4/13	Web Services	(Ion)
4/18	DHTapps+OpenDHT	(Scott)
4/20	Jini	(Ion)
4/25	Sensornets	(Scott)
4/27	Robust Protocols	(Scott)
5/2	Resource Allocation	(Ion)
5/4	Game theory	(Scott)
5/9	Review	(both)

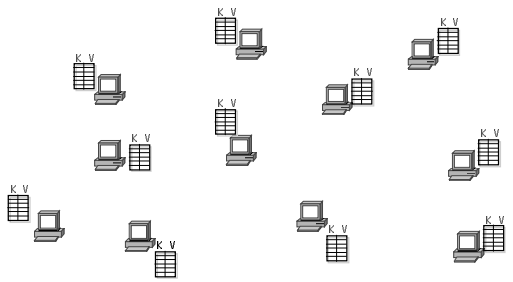
Note about Special Topics

- We won't require additional reading
- We will make clear what you need to know for the final

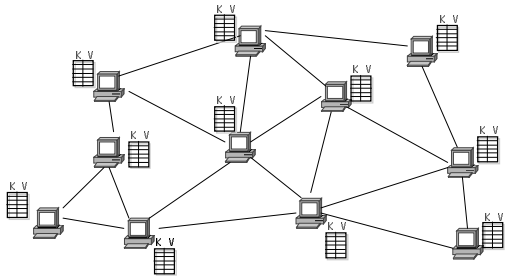
Outline for Today's Lecture

- What is a DHT? (review)
- Three classes of DHT applications (with examples):
  - rendezvous
  - storage
  - routing
- Why DHTs?
- DHTs and Internet Architecture?

A DHT in Operation: Peers

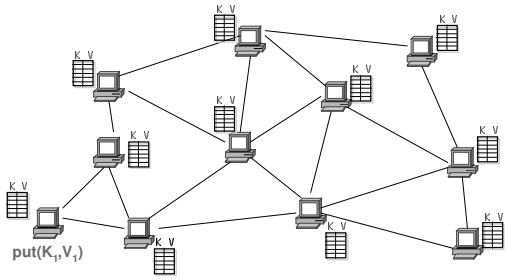


A DHT in Operation: Overlay



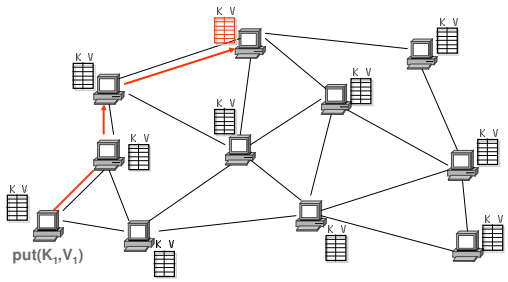
7

A DHT in Operation: put()



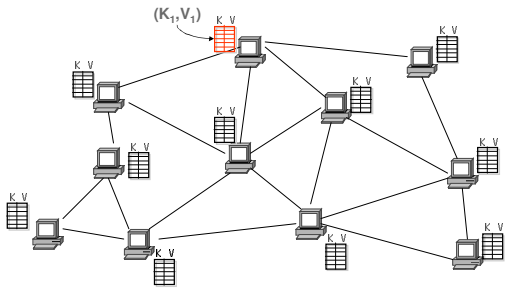
8

A DHT in Operation: put()



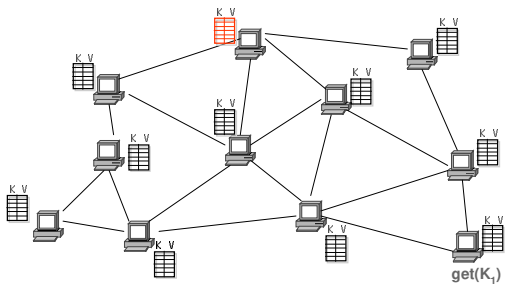
9

A DHT in Operation: put()



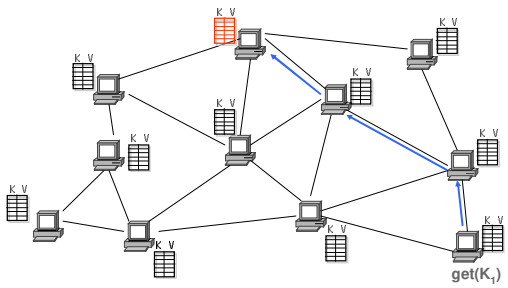
10

A DHT in Operation: get()



11

A DHT in Operation: get()



12

Key Requirement

- All puts and gets for a particular key must end up at the same machine
  - Even in the presence of failures and new nodes (churn)
- This depends on the DHT routing algorithm (last time)
  - Must be robust and scalable

13

Two Important Distinctions

- When talking about DHTs, must be clear whether you mean
  - Peers vs Infrastructure
  - Library vs Service

14

Peers or Infrastructure

- Peer:
  - Application users provide nodes for DHT
  - Example: music sharing, cooperative web cache
  - Easier to get, less well behaved
- Infrastructure:
  - Set of managed nodes provide DHT service
  - Perhaps serve many applications
  - Example: Planetlab
  - Harder to get, but more reliable

15

Library or Service

- Library: DHT code bundled into application
  - Runs on each node running application
  - Each application requires own routing infrastructure
  - Allows customization of interface
  - Very flexible, but much duplication
- Service: single DHT shared by applications
  - Requires common infrastructure
  - But eliminates duplicate routing systems
  - Harder to get, and much less flexible, but easier on each individual app

16

Not Covered Today

- Making lookup scale under churn
  - Better routing algorithms
- Manage data under churn
  - Efficient algorithms for creating and finding replicas
- Network awareness
  - Taking advantage of proximity without relying on it
- Developing proper analytic tools
  - Formalizing systems that are constantly in flux

17

Not Covered Today (cont'd)

- Dealing with adversaries
  - Robustness with untrusted participants
- Maintaining data integrity
  - Cryptographic hashes and Merkle trees
  - Consistency
- Privacy and anonymity
- More general functionality
  - Indexing, queries, etc.
- Load balancing and heterogeneity

18

DHTs vs Unstructured P2P

- DHTs good at:
  - exact match for "rare" items
- DHTs bad at:
  - keyword search, etc. [can't construct DHT-based Google]
  - tolerating extreme churn
- Gnutella etc. good at:
  - general search
  - finding common objects
  - very dynamic environments
- Gnutella etc. bad at:
  - finding "rare" items

19

Three Classes of DHT Applications

Rendezvous, Storage, and Routing

20

Rendezvous Applications

- Consider a pairwise application like telephony
- If A wants to call B (using the Internet), A can do the following:
  - A looks up B's "phone number" (IP address of current machine)
  - A's phone client contacts B's phone client
- What is needed is a way to "look up" where to contact someone, based on a username or some other global identifier

21

Using DHT for Rendezvous

- Each person has a globally unique key (say 128 bits)
  - Can be hash of a unique name, or something else
- Each client (telephony, chat, etc.) periodically stores the IP address (and other metadata) describing where they can be contacted
  - This is stored using their unique key
- When A wants to "call" B, it first does a get on B's key

22

Key Point

- The key (or identifier) is globally unique and static
- The DHT infrastructure is used to store the mapping between that static (persistent) identifier and the current location
  - DHT functions as a dynamic and flat DNS
- This can handle:
  - IP mobility
  - Chat
  - Internet telephony
  - DNS
  - The Web!

23

Using DHTs for the Web

Oversimplified:

- Name data with key
- Store IP address of file server(s) holding data
  - replication trivial!
- To get data, lookup key
- If want CDN-like behavior, make sure IP address handed back is close to requester (several ways to do this)

24

Three Classes of DHT Applications

Rendezvous, Storage, and Routing

25

Storage Applications

- Rendezvous applications use the DHT only to store small pointers (IP addresses, etc.)
- What about using DHTs for more serious storage, such as file systems

26

Examples of Storage Applications

- File Systems
- Backup
- Archiving
- Electronic Mail
- Content Distribution Networks
- .....

27

Why store data in a DHT?

- High storage capacity: many disks
- High serving capacity: many access links
- High availability by replication
- Simple application model

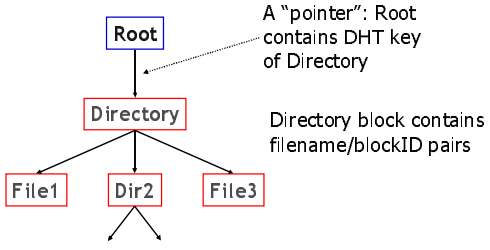
28

Example: CFS (DHash over Chord)

- Goal: serve a read-only file system
- Publisher inserts file system into DHT
- CFS client looks like an NFS file system:
  - /cfs/7ff23bda0092
- CFS client fetches data from the DHT

29

CFS Uses Tree of Blocks



30

CFS Uses Self-authentication

Immutable block: (Content-Hash Block)

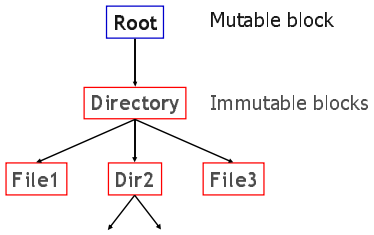
```
key = CryptographicHash(value)
  encourages data sharing!
```

Mutable block: (Public-key Block)

```
key = K_pub
value = data + Sign[data]_Kpriv
```

31

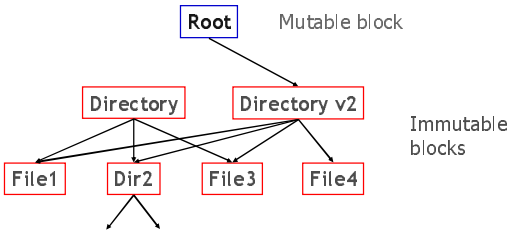
Most Blocks are Immutable



- This is a single-writer mutable data structure

32

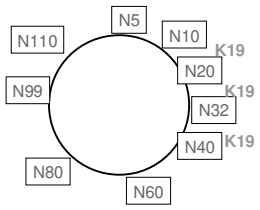
Adding a File to a Directory



33

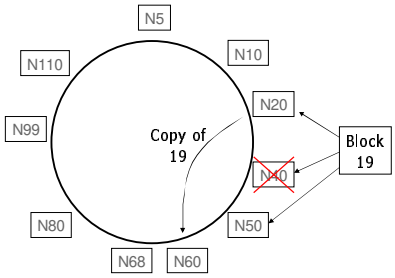
Data Availability via Replication

- DHash replicates each key/value pair at the nodes after it on the circle
- It's easy to find replicas
- Put(k,v) to all
- Get(k) from closest



34

First Live Successor Manages Replicas



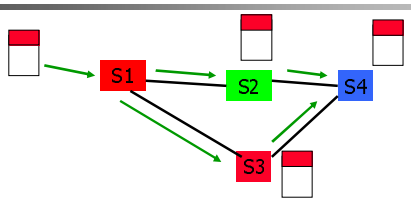
35

Usenet over a DHT

- Bulletin board (started in 1981)
  - Has grown exponentially in volume
  - 2004 volume is 1.4 Terabyte/day
- Hosting full Usenet has high costs
  - Large storage requirement
  - Bandwidth required: OC3+ ($30,000/month)
- Only 50 sites with full feed
- Goal: save Usenet news by reducing needed storage and bandwidth

36

Posting a Usenet Article



- User posts article to local server
- Server exchanges headers & article w. peers
- Headers allow sorting into newsgroups

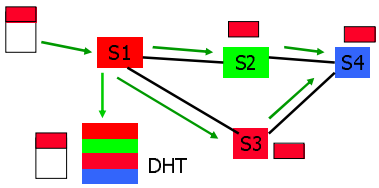
37

UsenetDHT

- Store article in shared DHT
- Only “single” copy of Usenet needed
- Can scale DHT to handle increased volume
- Incentive for ISPs: cut external bandwidth by providing high-quality hosting for local DHT server

38

Usenet Architecture



- User posts article to local server
- Server writes article to DHT
- Server exchanges headers only
- All servers know about each article

39

UsenetDHT Tradeoff

- Distribute headers as before:
  - clients have local access to headers
- Bodies held in global DHT
  - only accessed when read
  - greater latency, lower overhead

40

UsenetDHT: *potential* savings

	Net bandwidth	Storage
Usenet	12 Megabyte/s	10 Terabyte/week
UsenetDHT	120 Kbyte/s	60 Gbyte/week

- Suppose 300 site network
- Each site reads 1% of all articles

41

Three Classes of DHT Applications

Rendezvous, Storage, and Routing

42

"Routing" Applications

- Application-layer multicast
- Video streaming
- Event notification systems
- ...

43

DHT-Based Multicast

- Application-layer, not IP layer
- Single-source, not any-source multicast
- Easy to extend to anycast

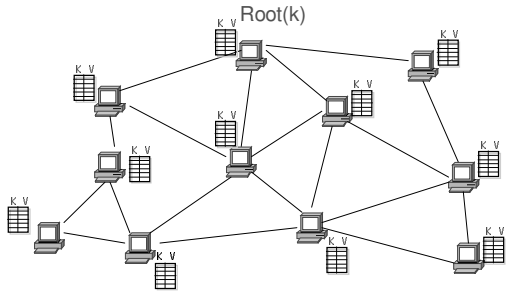
44

Tree Formation

- Group is associated with key
- "root" of group is node that owns key
- Any node that wants to join sends message to root, leaving forwarding state along path
- Message stops when it hits existing state for group
- Data sent from root reaches all nodes

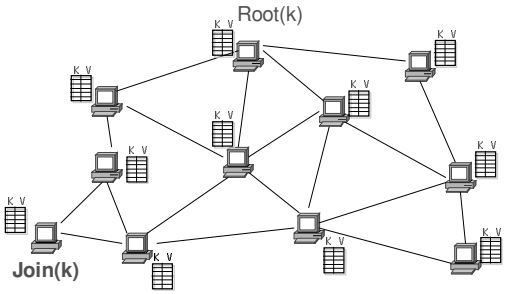
45

Multicast



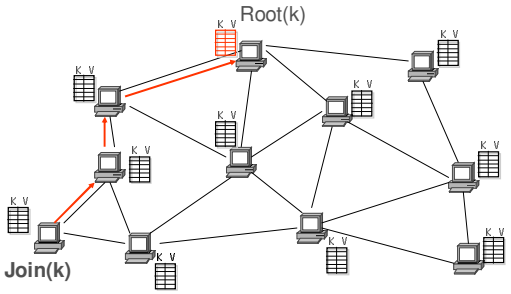
46

Multicast Join



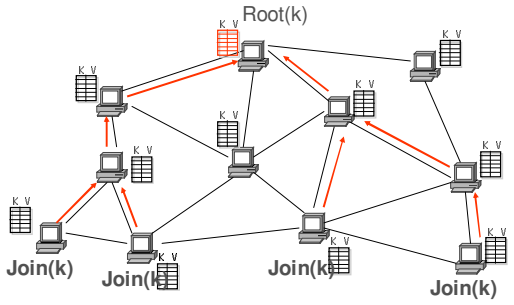
47

Multicast Join



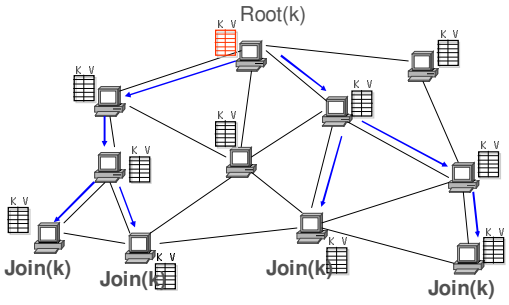
48

Multicast Join



49

Multicast Send



50

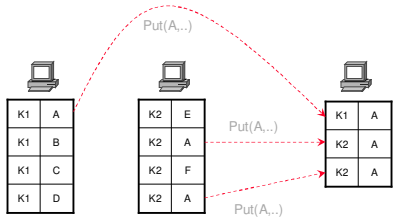
Challenges

- Repairing tree
- Balancing duties among peers
- Low-latency routing (proximity-based DHT routing)

51

Internet-Scale Query Processing

- Superficial motivation:
  - Database joins implemented with hash tables so...
  - Distributed joins can be implemented with DHTs
  - Scaling: latency O(log n) while computation O(n)

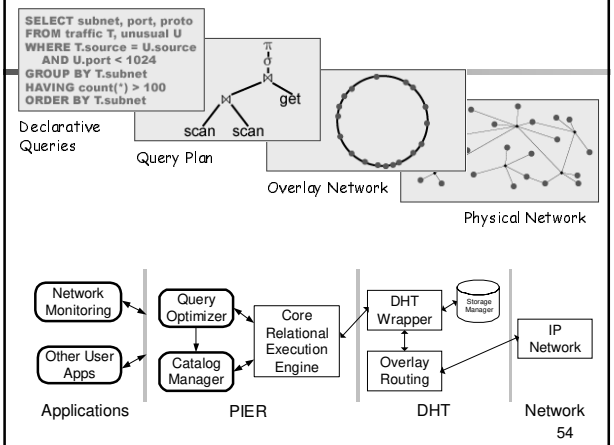


52

PIER

- Range of operators
  - Joins, aggregation (routing!), recursive, continuous queries
- Intended targets:
  - Data "in the wild" (filesharing, net monitoring, etc.)
  - No need for ACID semantics, just *best-effort*
- Future: more sophisticated queries
  - Range searches, etc.
  - Prefix Hash Tree

53



54

What's the Fuss about DHTs?

Goals, Strategy, Tactics

55

Distributed Systems Pre-Internet

- Connected by LANs (low loss and delay)
- Small scale (10s, maybe 100s per server)
- PODC literature focused on algorithms to achieve strict semantics in the face of failures
  - Two-phase commits
  - Synchronization
  - Byzantine agreement
  - Etc.

56

Distributed Systems Post-Internet

- Very different context:
  - Huge scales (thousands if not millions)
  - Highly variable connectivity
  - Failures common
  - Organic growth
- Abandoned distributed strict semantics
  - Adaptive apps rather than "guaranteed" infrastructure
- Adopted pairwise client-server approach
  - Server is centralized (even if server farm)
  - Relatively primitive approach (no sophisticated dist. algms.)
  - Little support from infrastructure or middleware

57

Problems with Centralized Server Farms

- Weak availability:
  - Susceptible to point failures and DoS attacks
- Management overhead
  - Data often manually partitioned to obtain scale
  - Management and maintenance large fraction of cost
- Per-application design (e.g., GoogleOS)
  - High hurdle for new applications
- Don't leverage the advent of powerful clients
  - Limits scalability and availability

58

The DHT Community's Goal

Produce a common infrastructure that will help solve these problems by being:

- Robust in the face of failures and attacks
  - Availability solved
- Self-configuring and self-managing
  - Management overhead reduced
- Usable for a wide variety of applications
  - No per-application design
- Able to support very large scales, with no assumptions about locality, etc.
  - No scaling limits, few restrictive assumptions

59

The Strategy

Define an interface for this infrastructure that is:

- Generally useful for a wide variety of applications
  - So many applications can leverage this work
- Can be supported by a robust, self-configuring, widely-distributed infrastructure
  - Addressing the many problems raised before

60

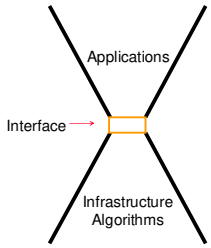
Research Plan (Tactics)

Two main research themes:

- **Above Interface:** Investigate the variety of applications that can use this interface
  - Many prototypes, trying to stretch limits
  - Some exploratory, others more definitive
- **Below Interface:** Investigate techniques for supporting this interface
  - Many designs and performance experiments
  - Looking at extreme limits (size, churn, etc.)

61

Hourglass Analogy



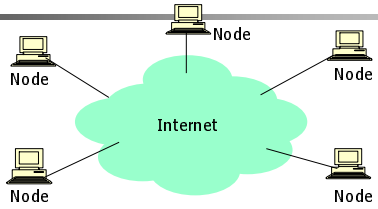
62

Two Crucial Design Decisions

- Technology for infrastructure: P2P
  - Take advantage of powerful clients
  - Decentralized
  - Nodes can be desktop machines or server quality
- Choice of interface: Lookup and Hash Table
  - Lookup(key) returns IP of host that "owns" key
  - Put()/Get() standard HT interface
  - Some flexibility in interface (no strict layers)

63

What is a P2P system?



- A distributed system architecture:
  - No centralized control
  - Nodes are symmetric in function
- Large number of (perhaps) server-quality nodes
- Enabled by technology improvements

64

P2P as Design Style

- Resistant to DoS and failures
  - Safety in numbers, no single point of attack or failure
- Self-organizing
  - Nodes insert themselves into structure
  - Need no manual configuration or oversight
- Flexible: nodes can be
  - Widely distributed or colocated
  - Powerful hosts or low-end PCs
  - Trusted or unknown peers

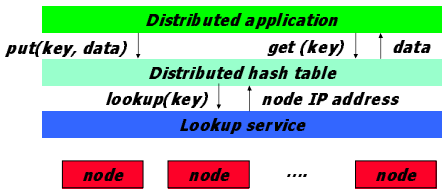
65

But What Interface?

- Challenge for P2P systems: finding content
  - Many machines, must find one that holds file
- Essential task: Lookup(key)
  - Given key, find host (IP) that has file with that key
- Higher-level interface: Put()/Get()
  - Easy to layer on top of lookup()
  - Allows application to ignore details of storage
    - System looks like one hard disk
  - Good for some apps, not for others

66

DHT Layering



- Application may be distributed over many nodes
- DHT distributes data storage over many nodes

67

Virtues of DHT Interface

- Simple and proven useful
  - Hash tables common implementation tool
- API supports a wide range of applications
  - No structure/meaning imposed on keys
  - Scalable, flat name space!
- Key/value pairs are persistent and global
  - Can store keys in other DHT values
  - And thus build complex data structures

68

Scenarios for DHT Usage

Where might there be a need for another approach?

69

Scenario #1: Public Infrastructure

- Consider CiteSeer or other nonprofit systems:
  - Service is very valuable to community
  - No source of revenue
- How can it expand?
  - Not enough support for expanding centralized facility
  - But many institutions would donate remote use of their local machines
- System problem:
  - Coordinating donated distributed infrastructure

70

The DHT Approach

- DHTs are well-suited to such settings
  - Inherently distributed with general interface
  - Naturally provides rendezvous and data sharing
- Developers can focus on how to layer app on top of DHT library
  - Resilience, scaling, all taken care of by DHT
- Typical assumption for important services:
  - Server-like nodes with good network access

71

Examples

- CiteSeer
  - Replicate current service (OverCite), but with 10x performance improvement
  - Use additional capacity to provide new features (e.g., SmartSeer's alerts)
- Cooperative CDNs
  - Coral allows universities to collaboratively handle "slashdot" workloads
  - Operational today with many users
- UsenetDHT
  - Allows cooperative institutions to share bandwidth load
  - Operational system with small feed running

72

Scenario #2: Scaling Enterprise Apps

- Enterprises rely on several crucial services
  - Email, backup, file storage
- These services must be
  - Scalable
  - Robust
  - Easy to deploy
  - Easy to manage
  - Inexpensive

73

The DHT approach

- Build all services on DHT interface
- DHT infrastructure:
  - Scalable (just add nodes, need not be local)
  - Robust
  - Easy to deploy
  - Easy to manage
  - Exploits inexpensive commodity components

74

Examples

- Email
  - ePOST (Rice)
- Backup
  - MIT
- File storage
  - OceanStore

75

Scenario #3: Supporting Tiny Apps

- Many apps could use DHT interface, but are too small to deploy one themselves
  - Small: user population, importance, etc.
- Such an application could use a DHT *service*
- OpenDHT is a public DHT service
  - Lecture on this next week...

76

Scenario #4: Super-Resilence

- DHTs are a natural way to build super-resilient services
- DHTs would be a natural candidate for the next generation name service, or other such crucial pieces of the infrastructure

77

Not Just for Applications

- DHTs resolve flat names scalably
  - We haven't been able to do this before
- How would we redesign the Internet, now that we can resolve flat names?

78

DHTs and Internet Architecture?

79

Early Applications Were Host-Centric

- Destination part of user's goal:
  - e.g., Telnet
- Specified by hostname, not IP address
  - DNS translates between the two
- DNS built around hierarchy:
  - local decentralized control (writing)
  - efficient hostname resolution (reading)

80

Internet Naming is Host-Centric

- DNS names and IP addresses are the only global naming systems in Internet
- These structures are host-centric:
  - IP addresses: network location of host
  - DNS names: domain of host
- Both are closely tied to an underlying structure:
  - IP addresses: network topology
  - DNS names: domain structure

81

The Web is Data-Centric

- URLs function as the name of data
  - Users usually care about *content*, not *location*
  - www.cnn.com is a *brand*, not a *host*
  - Tying data to hosts is unnatural
- URLs are bad names for data:
  - Not persistent (name changes when data moves)
  - Can't handle piecewise replication
  - Legal contention over names

82

Larger Lesson

- For many objects, we will want persistent names
- If a name refers to properties of its referent that can change, the name is necessarily ephemeral.
  - IP addresses can't serve as persistent host names
  - URLs can't serve as persistent data names
- Why do names have structure, anyway?

83

Old Implicit Assumption

- Internet names must have hierarchical structure in order to be resolvable
- Setting up a new naming scheme requires defining a new (globally recognized) hierarchy
- Problem: For these names to be persistent, the hierarchy must match the natural structure of the objects they name.
  - What is the natural hierarchy of documents?

84

DHTs Enable Flat Names

- Flat names are names with no structure
- DHTs resolve flat names in logarithmic time
  - And often much faster
  - This is the same as in a tree
  - No longer need hierarchy for resolution speed
- But, flat names pose other problems (return to later)
  - Control (used to be locally managed)
  - Locality (part of DNS's success)
  - User-friendliness

85

Why Are Flat Names Good?

- Flat names impose no structure on the objects they name
  - Not true with structured names like DNS or IP add's
- Flat names can be used to name anything
- Once you have a large flat namespace, you never need another naming system
  - One namespace
  - One resolution infrastructure

86

Semantic-Free Referencing (SFR)

- Replace URLs by flat, semantic-free keys
  - Persistent
  - No contention
- Use a DHT to resolve keys to host/path
  - "A DNS for data"
  - Replication easy: multiple entries
- Other design issues:
  - Ensure data security and integrity
  - Provide fate-sharing and locality

87

Elegant but Unusable?

- How to get the keys you want?
  - Third-party services will provide mapping between user-level names and keys (think: Google)
  - Competitive market outside infrastructure
- Do you have the key you wanted?
  - Metadata includes signed "testimonials" (3rd party)
- Who is going to supply the resolution service?
  - Competitive market much like tier-1 ISPs?
  - Each access or store is by or for customers

88

Why Stop with the Web?

- DHTs enable use of flat names
- Names should not impose structure on referents
  - Flat names can name *anything*
- Why not a single name resolution infrastructure?
  - A generalized DNS
- New architecture proposed to support:
  - endpoint identifiers
  - service identifiers

89

Layered Naming for the Internet

Software should use names at the proper level of abstraction

Application (SIDs)

SID Resolution (to EID)

Transport Protocol (EIDs)

EID Resolution (to IP address)

IP (IP addresses)

90