

Architectures and Algorithms for Internet-Scale (P2P) Data Management

Joe Hellerstein
Intel Research & UC Berkeley

Overview

- Preliminaries
 - What, Why
 - The Platform
- "Upleveling"
 - Network Data Independence
- Early P2P architectures
 - Client-Server
 - Floodsast
 - Hierarchies
 - A Little Gossip
 - Commercial Offerings
 - Lessons and Limitations
- Ongoing Research
 - Structured Overlays: DHTs
 - Query Processing on Overlays
 - Storage Models & Systems
 - Security and Trust
- Joining the fun
 - Tools and Platforms
 - Closing thoughts

Acknowledgments

- For specific content in these slides
 - Frans Kaashoek
 - Petros Maniatis
 - Sylvia Ratnasamy
 - Timothy Roscoe
 - Scott Shenker
- Additional Collaborators
 - Brent Chun, Tyson Condie, Ryan Huebsch, David Karger, Ankur Jain, Jinyang Li, Boon Thau Loo, Robert Morris, Sriram Ramabhadran, Sean Rhea, Ion Stoica, David Wetherall

Preliminaries

Outline

- Scoping the tutorial
- Behind the "P2P" Moniker
 - Internet-Scale systems
- Why bother with them?
- Some guiding applications

Scoping the Tutorial

- Architectures and Algorithms for Data Management
- The perils of overviews
 - Can't cover everything. So much here!
- Some interesting things we'll skip
 - Semantic Mediation: data integration on steroids
 - E.g., Hyperion (Toronto), Piazza (UWash), etc.
 - High-Throughput Computing
 - I.e. The Grid
 - Complex data analysis/reduction/mining
 - E.g. p2p distributed inference, wavelets, regression, matrix computations, etc.

Moving Past the "P2P" Moniker: The Platform

- The "P2P" name has lots of connotations
 - Simple filestealing systems
 - Very end-user-centric
- Our focus here is on:
 - Many participating machines, symmetric in function
 - Very Large Scale (MegaNodes, not PetaBytes)
 - Minimal (or non-existent) management
 - Note: user model is flexible
 - Could be embedded (e.g. in OS, HW, firewall, etc.)
 - Large-scale hosted services a la Akamai or Google
 - A key to achieving "autonomic computing"?

Overlay Networks

- P2P applications need to:
 - Track identities & (IP) addresses of peers
 - May be many!
 - May have significant Churn
 - Best not to have n^2 ID references
 - Route messages among peers
 - If you don't keep track of all peers, this is "multi-hop"
- This is an *overlay network*
 - Peers are doing both naming and routing
 - IP becomes "just" the low-level transport
 - All the IP routing is opaque
- Control over naming and routing is powerful
 - And as we'll see, brings networks into the database era

Many New Challenges

- Relative to other parallel/distributed systems
 - Partial failure
 - Churn
 - Few guarantees on transport, storage, etc.
 - Huge optimization space
 - Network bottlenecks & other resource constraints
 - No administrative organizations
 - Trust issues: security, privacy, incentives
- Relative to IP networking
 - Much higher function, more flexible
 - Much less controllable/predictable

Why Bother? Not the Gold Standard

- Given an infinite budget, would you go p2p?
- Highest performance? No.
 - Hard to beat hosted/managed services
 - p2p Google appears to be infeasible [Li, et al. IPTPS 03]
- Most Resilient? Hmmmm.
 - In principle more resistant to DoS attacks, etc.
 - Today, still hard to beat hosted/managed services
 - Geographically replicated, hugely provisioned
 - People who "do it for dollars" today don't do it p2p

Why Bother II: Positive Lessons from Filestealing

- P2P enables *organic scaling*
 - Vs. the top few killer services -- no VCs required!
 - Can afford to "place more bets", try wacky ideas
- Centralized services engender scrutiny
 - Tracking users is trivial
 - Provider is liable (for misuse, for downtime, for local laws, etc.)
- Centralized means business
 - Need to pay off startup & maintenance expenses
 - Need to protect against liability
 - Business requirements drive to particular short-term goals
 - *Tragedy of the commons*

Why Bother III? Intellectual motivation

- Heady mix of theory and systems
 - Great community of researchers have gathered
 - Algorithms, Networking, Distributed Systems, Databases
 - Healthy set of publication venues
 - IPTPS workshop as a catalyst
 - Surprising degree of collaboration across areas
 - In part supported by NSF Large ITR (project IRIS)
 - UC Berkeley, ICSI, MIT, NYU, and Rice

Infecting the Network, Peer-to-Peer

- The Internet is hard to change.
- But Overlay Nets are easy!
 - P2P is a wonderful "host" for infecting network designs
 - The "next" Internet is likely to be very different
 - "Naming" is a key design issue today
 - Querying and data independence key tomorrow?
- Don't forget:
 - The Internet was originally an overlay on the telephone network
 - There is no money to be made in the bit-shipping business
- A modest goal for DB research:
 - Don't query the Internet.

Infecting the Network, Peer-to-Peer

Be the Internet.

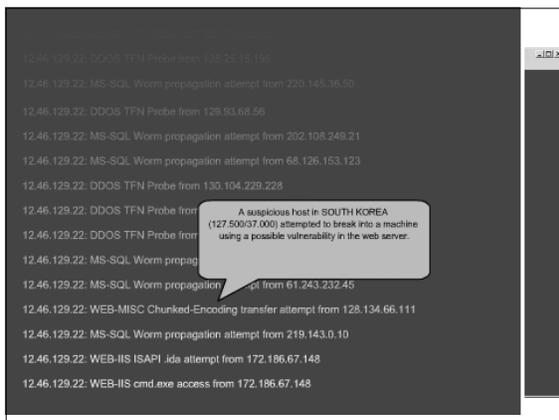
- A modest goal for DB research:
 - Don't query the Internet.

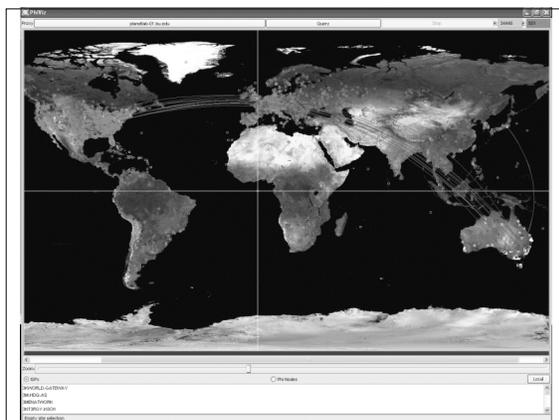
Some Guiding Applications

- φ
 - Intel Research & UC Berkeley
- LOCKSS
 - Stanford, HP Labs, Sun, Harvard, Intel Research
- LiberationWare

ϕ : Public Health for the Internet

- Security tools focused on "medicine"
 - Vaccines for Viruses
 - Improving the world one patient at a time
- Weakness/opportunity in the "Public Health" arena
 - Public Health: population-focused, community-oriented
 - Epidemiology: incidence, distribution, and control in a *population*
- ϕ : A New Approach
 - Perform *population-wide measurement*
 - Enable massive sharing of data and query results
 - The "Internet Screensaver"
 - Engage end users: education and prevention
 - Understand risky behaviors, at-risk populations.
- Prototype running over *PIER*





φ Vision: Network Oracle

- Suppose there existed a Network Oracle
 - Answering questions about current Internet state
 - Routing tables, link loads, latencies, firewall events, etc.
 - How would this change things
 - Social change (Public Health, safe computing)
 - Medium term change in distributed application design
 - Currently distributed apps do some of this on their own
 - Long term change in network protocols
 - App-specific custom routing
 - Fault diagnosis
 - Etc.

LOCKSS: Lots Of Copies Keep Stuff Safe



- Digital Preservation of Academic Materials
- Librarians are scared with good reason
 - Access depends on the fate of the publisher
 - Time is unkind to bits after decades
 - Plenty of enemies (ideologies, governments, corporations)
- Goal: *Archival* storage and access

LOCKSS Approach



- Challenges:
 - Very low-cost hardware, operation and administration
 - No central control
 - Respect for access controls
 - A long-term horizon
- Must anticipate and degrade gracefully with
 - Undetected bit rot
 - Sustained attacks
 - Esp. Stealth modification
- Solution:
 - P2P auditing and repair system for replicated docs

LiberationWare

- Take your favorite Internet application
 - Web hosting, search, IM, filesharing, VoIP, email, etc.
 - Consider using centralized versions in a country with a repressive government
 - Trackability and liability will prevent this being used for free speech
 - Now consider p2p
 - Enhanced with appropriate security/privacy protections
 - Could be the medium of the next Tom Paines
- Examples: FreeNet, Publius, FreeHaven
 - p2p storage to avoid censorship & guarantee privacy
 - PKI-encrypted storage
 - Mix-net privacy-preserving routing

"Upleveling": Network Data Independence

SIGMOD Record, Sep. 2003

Recall Codd's Data Independence

- Decouple app-level API from data organization
 - Can make changes to data layout without modifying applications
 - Simple version: location-independent names
 - Fancier: declarative queries

"As clear a paradigm shift as we can hope to find in computer science"
- C. Papadimitriou

The Pillars of Data Independence

- Indexes
 - Value-based lookups have to compete with direct access
 - Must adapt to shifting data distributions
 - Must guarantee performance
- Query Optimization
 - Support declarative queries beyond lookup/search
 - Must adapt to shifting data distributions
 - Must adapt to changes in environment

B-Tree

Join Ordering,
AM Selection,
etc.

Generalizing Data Independence

- A classic “level of indirection” scheme
 - Indexes are exactly that
 - Complex queries are a richer indirection
- The key for data independence:
 - It’s all about *rates of change*
- Hellerstein’s Data Independence Inequality:
 - Data independence matters when

$$d(\text{environment})/dt \gg d(\text{app})/dt$$

Data Independence in Networks

$$d(\text{environment})/dt \gg d(\text{app})/dt$$

- In databases, the RHS is unusually small
 - This drove the relational database revolution
- In extreme networked systems, LHS is unusually high
 - And the applications increasingly complex and data-driven
 - Simple indirections (e.g. local lookaside tables) insufficient

The Pillars of Data Independence

- Indexes
 - Value-based lookups have to compete with direct access
 - Must adapt to shifting data distributions
 - Must guarantee performance
- Query Optimization
 - Support declarative queries beyond lookup/search
 - Must adapt to shifting data distributions
 - Must adapt to changes in environment

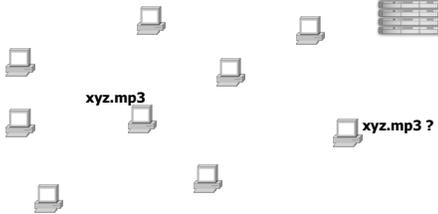
| | |
|--------|---|
| B-Tree | Content-Addressable Overlay Networks (DHTs) |
|--------|---|

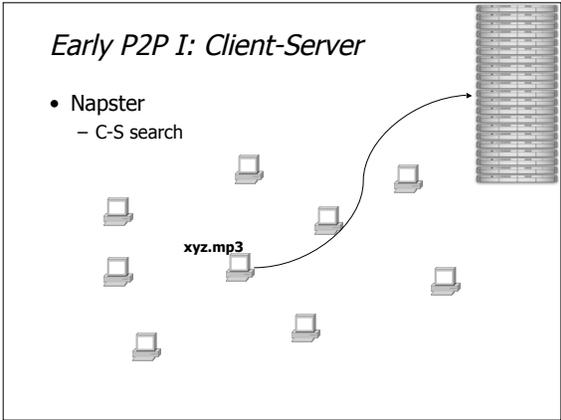
| | |
|-----------------------------------|------------------------------|
| Join Ordering, AM Selection, etc. | Multiquery dataflow sharing? |
|-----------------------------------|------------------------------|

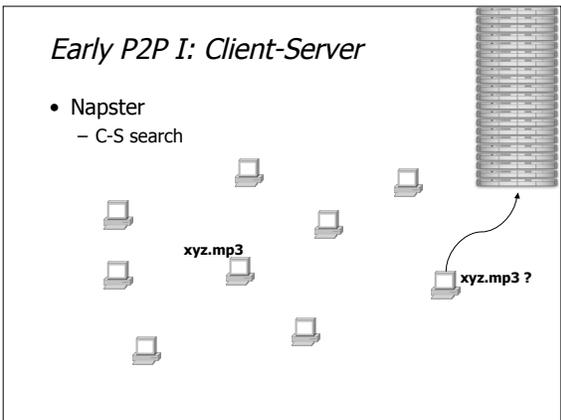
Early P2P

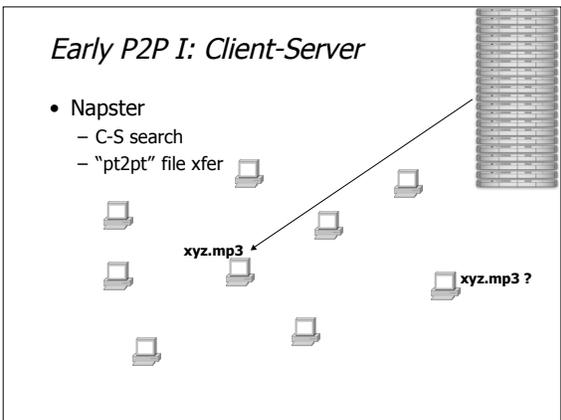
Early P2P I: Client-Server

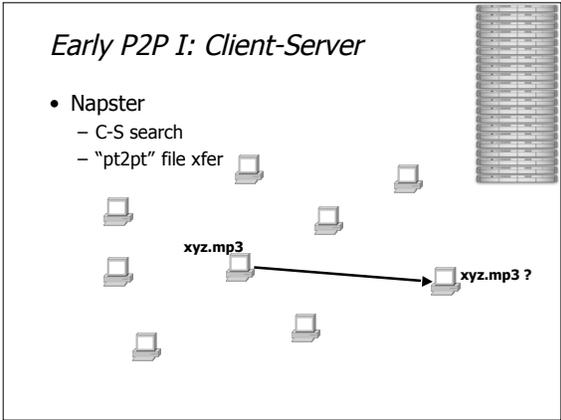
- Napster

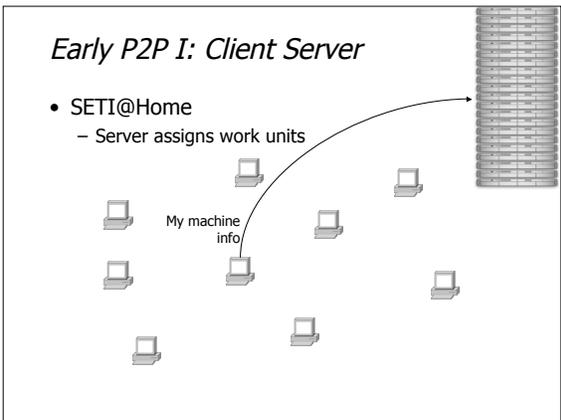


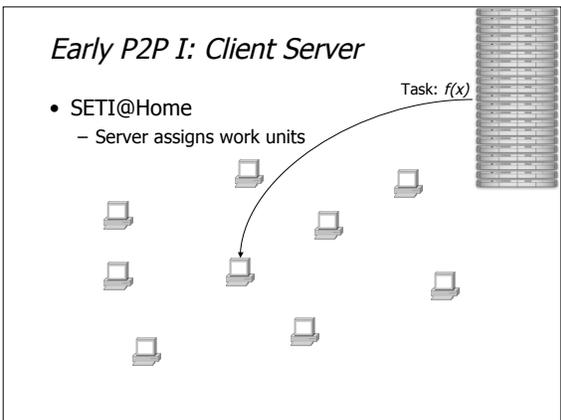


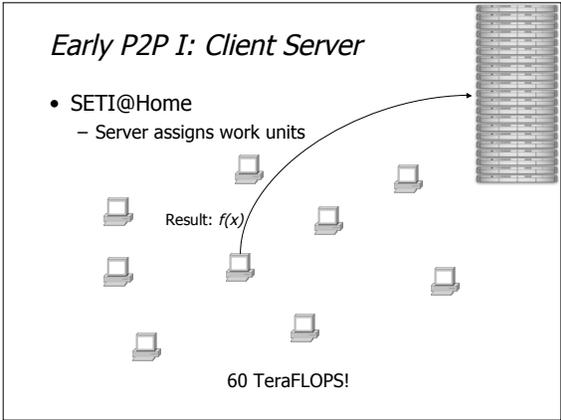


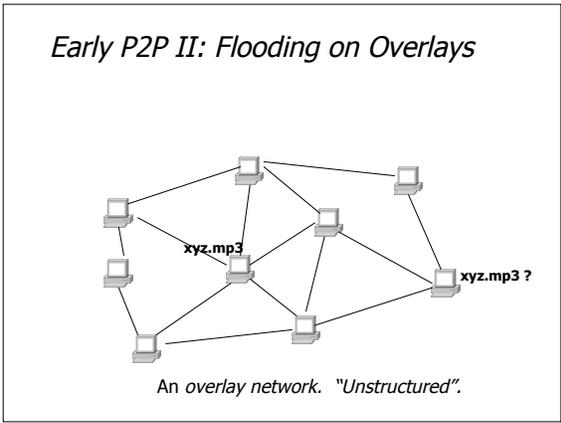


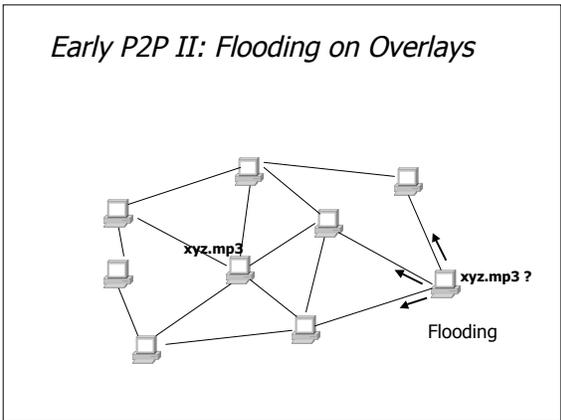




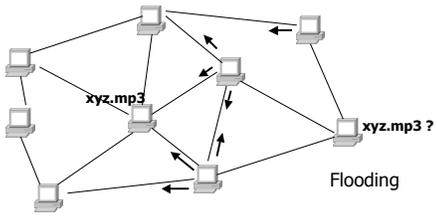




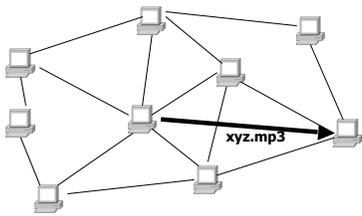




Early P2P II: Flooding on Overlays

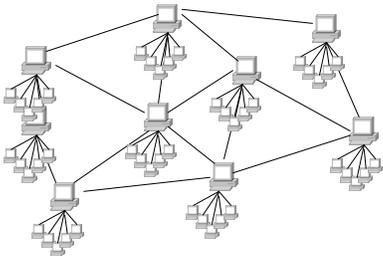


Early P2P II: Flooding on Overlays



Early P2P II.v: "Ultrappeers"

- Ultrappeers can be installed (KaZaA) or self-promoted (Gnutella)



Hierarchical Networks (& Queries)

- IP
 - Hierarchical name space (www.vldb.org, 141.12.12.51)
 - Hierarchical routing
 - Autonomous Systems correlate with name space (though not perfectly)
 - Astrolabe [Birman, et al. TOCS 03]
 - OLAP-style aggregate queries down the IP hierarchy
- DNS
 - Hierarchical name space ("clients" + hierarchy of servers)
 - Hierarchical routing w/aggressive caching
 - 13 managed "root servers"
 - IrisNet [Deshpande, et al. SIGMOD 03]
 - Xpath queries over (selected) DNS (sub)-trees.
- Traditional pros/cons of Hierarchical data mgmt
 - Works well for things aligned with the hierarchy
 - Esp. physical locality a la Astrolabe
 - Inflexible
 - No data independence!

Commercial Offerings

- JXTA
 - Java/XML Framework for p2p applications
 - Name resolution and routing is done with floods & superpeers
 - Can always add your own if you like
- MS WinXP p2p networking
 - An unstructured overlay, flooded publication and caching
 - "does not yet support distributed searches"
- Both have some security support
 - Authentication via signatures (assumes a trusted authority)
 - Encryption of traffic
- Groove
 - Platform for p2p "experience". IM and asynch collab tools.
 - Client-serverish name resolution, backup services, etc.

Lessons and Limitations

- Client-Server performs well
 - But not always feasible
 - Ideal performance is often not the key issue!
- Things that flood-based systems do well
 - Organic scaling
 - Decentralization of visibility and liability
 - Finding popular stuff
 - Fancy *local* queries
- Things that flood-based systems do poorly
 - Finding unpopular stuff [Loo, et al VLDB 04]
 - Fancy *distributed* queries
 - Vulnerabilities: data poisoning, tracking, etc.
 - Guarantees about anything (answer quality, privacy, etc.)

A Little Gossip

Gossip Protocols (Epidemic Algorithms)

- Originally targeted at database replication [Demers, et al. PODC '87]
 - Especially nice for unstructured networks
 - *Rumor-mongering*: propagate newly-received update to k random neighbors
- Extended to routing
 - Point-to-point routing [Vahdat/Becker TR, '00]
 - Rumor-mongering of queries instead of flooding [Haas, et al Infocom '02]
- Extended to aggregate computation [Kempe, et al, FOCS 03]
- Mostly theoretical analyses
 - Usually of two forms:
 - What is the "tipping point" where an epidemic infects the whole population? (Percolation theory)
 - What is the expected # of messages for infection?
- A Cornell specialty
 - Demers, Kleinberg, Gehrke, Halpern, ...

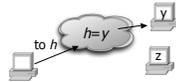
Structured Overlays: Distributed Hash Tables (DHTs)

DHT Outline

- High-level overview
- Fundamentals of structured network topologies
 - And examples
- One concrete DHT
 - Chord
- Some systems issues
 - Storage models & soft state
 - Locality
 - Churn management

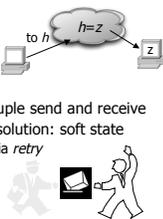
High-Level Idea: Indirection

- Indirection in space
 - Logical (content-based) IDs, routing to those IDs
 - "Content-addressable" network
 - Tolerant of *churn*
 - nodes joining and leaving the network



High-Level Idea: Indirection

- Indirection in space
 - Logical (content-based) IDs, routing to those IDs
 - "Content-addressable" network
 - Tolerant of *churn*
 - nodes joining and leaving the network
- Indirection in time
 - Want some scheme to temporally decouple send and receive
 - Persistence required. Typical Internet solution: *soft state*
 - Combo of persistence via *storage* and via *retry*
 - "Publisher" requests TTL on storage
 - Republishes as needed
- Metaphor: Distributed Hash Table



What is a DHT?

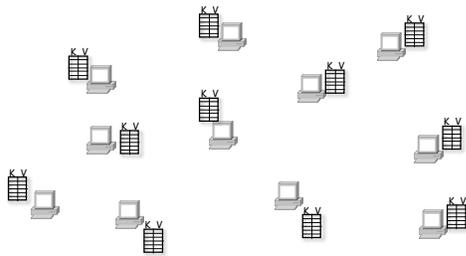
- Hash Table
 - data structure that maps "keys" to "values"
 - essential building block in software systems
- Distributed Hash Table (DHT)
 - similar, but spread across the Internet
- Interface
 - insert(key, value)
 - lookup(key)

How?

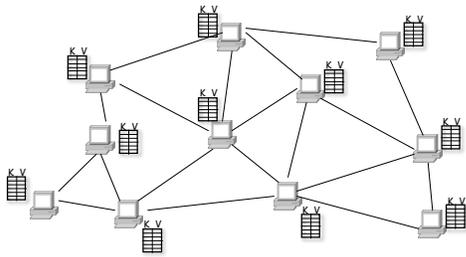
Every DHT node supports a single operation:

- Given *key* as input; route messages toward node holding *key*

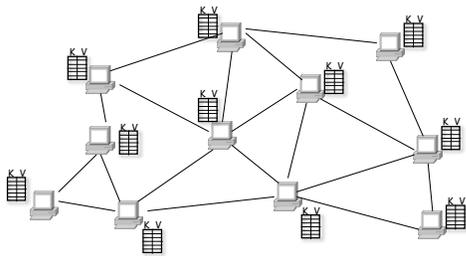
DHT in action



DHT in action

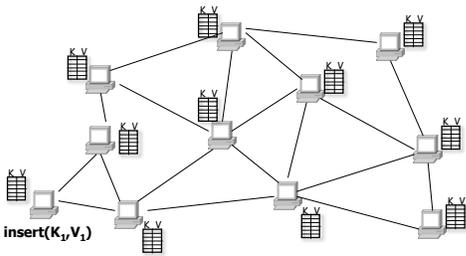


DHT in action



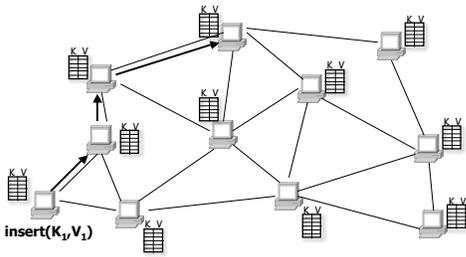
Operation: take *key* as input; route messages to node holding *key*

DHT in action: put()



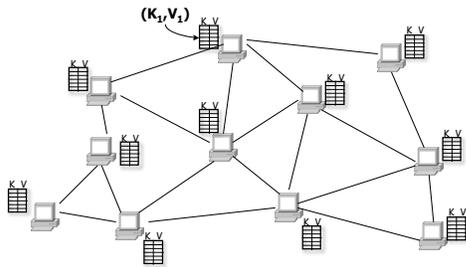
Operation: take *key* as input; route messages to node holding *key*

DHT in action: put()



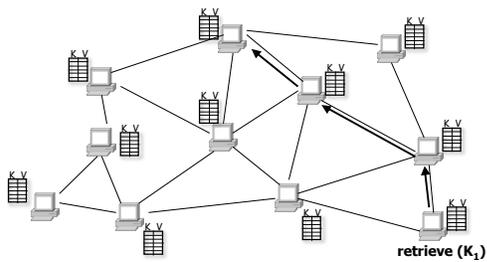
Operation: take *key* as input; route messages to node holding *key*

DHT in action: put()



Operation: take *key* as input; route messages to node holding *key*

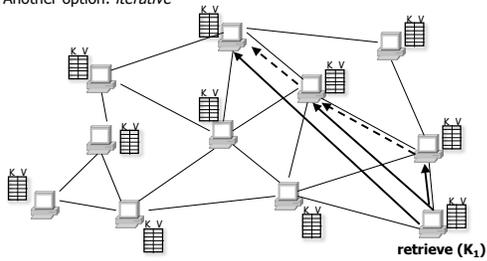
DHT in action: get()



Operation: take *key* as input; route messages to node holding *key*

Iterative vs. Recursive Routing

Previously showed recursive.
Another option: iterative



Operation: take *key* as input; route messages to node holding *key*

DHT Design Goals

- An "overlay" network with:
 - Flexible mapping of keys to physical nodes
 - Small network diameter
 - Small degree (fanout)
 - Local routing decisions
 - Robustness to churn
 - Routing flexibility
 - Decent locality (low "stretch")
- A "storage" or "memory" mechanism with
 - No guarantees on persistence
 - Maintenance via soft state

Peers vs Infrastructure

- Peer:
 - Application users provide nodes for DHT
 - Examples: filesharing, etc
- Infrastructure:
 - Set of managed nodes provide DHT service
 - Perhaps serve many applications
 - A p2p "incubator"?
 - We'll discuss this at the end of the tutorial

Library or Service

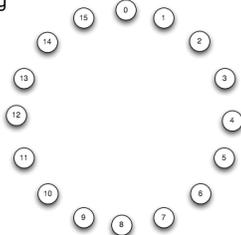
- Library: DHT code bundled into application
 - Runs on each node running application
 - Each application requires own routing infrastructure
- Service: single DHT shared by applications
 - Requires common infrastructure
 - But eliminates duplicate routing systems

DHT Outline

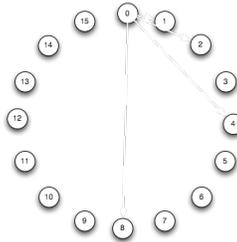
- High-level overview
- Fundamentals of structured network topologies
 - And examples
- One concrete DHT
 - Chord
- Some systems issues
 - Storage models & soft state
 - Locality
 - Churn management

An Example DHT: Chord

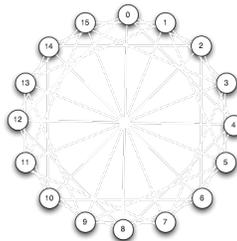
- Assume $n = 2^m$ nodes for a moment
 - A “complete” Chord ring
 - We’ll generalize shortly



An Example DHT: Chord

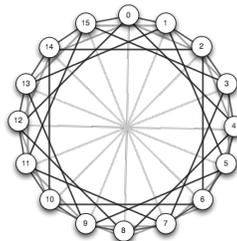


An Example DHT: Chord



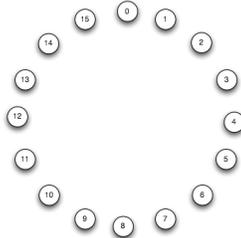
An Example DHT: Chord

- Overlaid 2^k -Gons



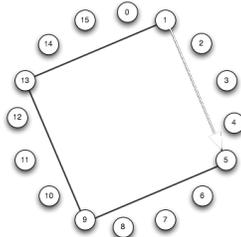
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



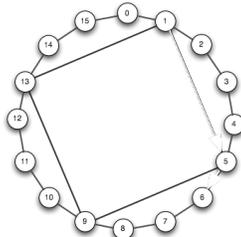
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



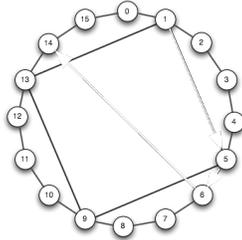
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



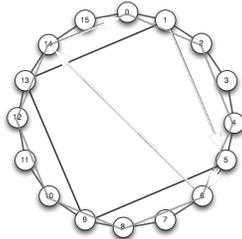
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



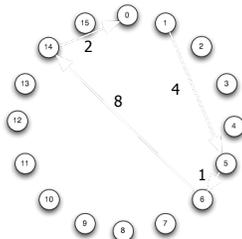
Routing in Chord

- At most one of each Gon
- E.g. 1-to-0



Routing in Chord

- At most one of each Gon
- E.g. 1-to-0
- What happened?
 - We constructed the binary number 15!
 - Routing from x to y is like computing $y - x \pmod n$ by summing powers of 2



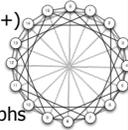
Diameter: $\log n$ (1 hop per gon type)
 Degree: $\log n$ (one outlink per gon type)

What is happening here? Algebra!

- Underlying group-theoretic structure
 - Recall a *group* is a set S and an operator \bullet such that:
 - S is closed under \bullet
 - Associativity: $(AB)C = A(BC)$
 - There is an *identity* element $I \in S$ s.t. $IX = XI = X$ for all $X \in S$
 - There is an inverse $X^{-1} \in S$ for each element $X \in S$ s.t. $XX^{-1} = X^{-1}X = I$
- The *generators* of a group
 - Elements $\{g_1, \dots, g_n\}$ s.t. application of the operator on the generators produces all the members of the group.
- Canonical example: $(\mathbb{Z}_n, +)$
 - Identity is 0
 - A set of generators: $\{1\}$
 - A different set of generators: $\{2, 3\}$

Cayley Graphs

- The *Cayley Graph* (S, E) of a group:
 - Vertices corresponding to the underlying set S
 - Edges corresponding to the *actions of the generators*
- (Complete) Chord is a Cayley graph for $(\mathbb{Z}_n, +)$
 - $S = \mathbb{Z} \bmod n$ ($n = 2^k$).
 - Generators $\{1, 2, 4, \dots, 2^{k-1}\}$
 - That's what the gons are all about!
- Fact: Most (complete) DHTs are Cayley graphs
 - And they didn't even know it!
 - Follows from parallel InterConnect Networks (ICNs)
 - Shown to be group-theoretic [Akers/Krishnamurthy '89]

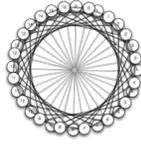


Note: the ones that aren't Cayley Graphs are *coset graphs*, a related group-theoretic structure

So...?

- Two questions:
 - How did this happen?
 - Why should you care?

How Hairy met Cayley

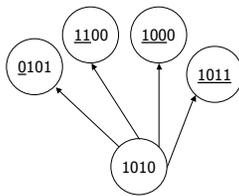


- What do you want in a structured network?
 - Uniformity of routing logic
 - Efficiency/load-balance of routing and maintenance
 - Generality at different scales
- Theorem: All Cayley graphs are *vertex symmetric*.
 - I.e. isomorphic under swaps of nodes
 - So routing from y to x looks just like routing from $(y-x)$ to 0
 - The routing code at each node is the same! Simple software.
 - Moreover, under a random workload the routing responsibilities (congestion) at each node are the same!
- Cayley graphs tend to have good degree/diameter tradeoffs
 - Efficient routing with few neighbors to maintain
- Many Cayley graphs are *hierarchical*
 - Made of smaller Cayley graphs connected by a new generator
 - E.g. a Chord graph on 2^{m+1} nodes looks like 2 interleaved (half-notch rotated) Chord graphs of 2^m nodes with half-notch edges
 - Again, code is nice and simple

Upshot

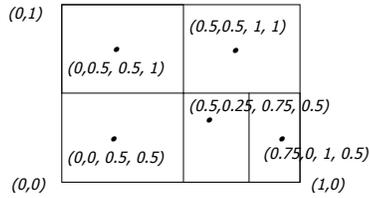
- Good DHT topologies will be Cayley/Coset graphs
 - A replay of ICN Design
 - But DHTs can use funky “wiring” that was infeasible in ICNs
 - All the group-theoretic analysis becomes suggestive
- Clean math describing the topology helps crisply analyze efficiency
 - E.g. degree/diameter tradeoffs
 - E.g. shapes of trees we’ll see later for aggregation or join
- Really no excuse to be “sloppy”
 - ISAM vs. B-trees

Pastry/Bamboo



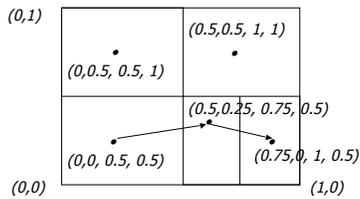
- Based on Plaxton Mesh [Plaxton, et al SPAA 97]
- Names are fixed bit strings
- Topology: Prefix Hypercube
 - For each bit from left to right, pick a neighbor ID with common flipped bit and common prefix
 - $\log n$ degree & diameter
- Plus a ring
 - For reliability (with k pred/succ)
- Suffix Routing from A to B
 - “Fix” bits from left to right
 - E.g. 1010 to 0001:
1010 \rightarrow 0101 \rightarrow 0010 \rightarrow 0000 \rightarrow 0001

CAN: Content Addressable Network



- Exploit multiple dimensions
- Each node is assigned a zone
- Nodes are identified by zone boundaries
- Join: chose random point, split its zone

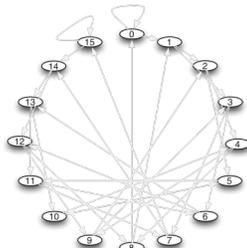
Routing in 2-dimensions



- Routing is navigating a d -dimensional ID space
 - Route to closest neighbor in direction of destination
 - Routing table contains $O(d)$ neighbors
- Number of hops is $O(dN^{1/d})$

Koorde

- DeBruijn graphs
 - Link from node x to nodes $2x$ and $2x+1$
 - Degree 2, diameter $\log n$
 - Optimal!
- Koorde is Chord-based
 - Basically Chord, but with DeBruijn fingers



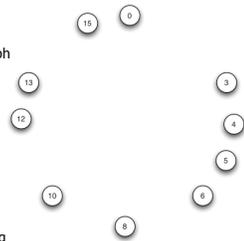
Note: Not vertex-symmetric!
Not a Cayley graph. But a coset graph of the "butterfly" topology.

Topologies of Other Oft-cited DHTs

- Tapestry
 - Very similar to Pastry/Bamboo topology
 - No ring
- Kademia
 - Also similar to Pastry/Bamboo
 - But the "ring" is ordered by the XOR metric
 - Used by the Overnet/eDonkey filesharing system
- Viceroy
 - An emulated Butterfly network
- Symphony
 - A randomized "small-world" network

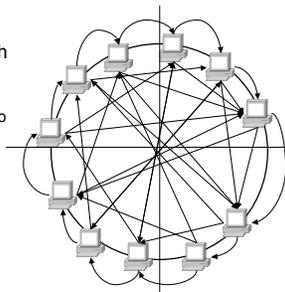
Incomplete Graphs: Emulation

- For Chord, we assumed 2^m nodes. What if not?
 - Need to "emulate" a complete graph even when incomplete.
 - Note: you've seen this problem before!
 - Litwin's Linear Hashing emulates hashables of length 2^m !
- DHT-specific schemes used
 - In Chord, node x is responsible for the range $[x, succ(x))$
 - The "holes" on the ring should be randomly distributed due to hashing
 - *Consistent Hashing* [Karger, et al. STOC 97]



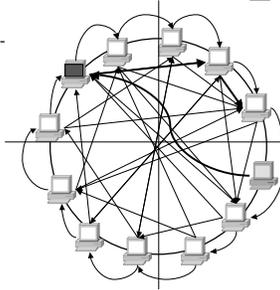
Chord in Flux

- Essentially never a "complete" chord graph
 - Maintain a "ring" of successor nodes
 - For redundancy, point to k successors
 - Point to nodes responsible for ID s at powers of 2
 - Sometimes called "fingers"
 - 1st finger is the successor



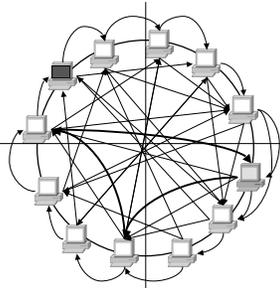
Joining the Chord Ring

- Need IP of some node
- Pick a random ID (e.g. SHA-1(IP))
- Send msg to current owner of that ID
 - That's your predecessor



Joining the Chord Ring

- Need IP of some node
 - Pick a random ID (e.g. SHA-1(IP))
 - Send msg to current owner of that ID
 - That's your predecessor
 - Update pred/succ links
 - Once the ring is in place, all is well!
 - Inform app to move data appropriately
 - Search to install "fingers" of varying powers of 2
 - Or just copy from pred/succ and check!
 - Inbound fingers fixed lazily
- Theorem: If consistency is reached before network *doubles*, lookups remain $\log n$



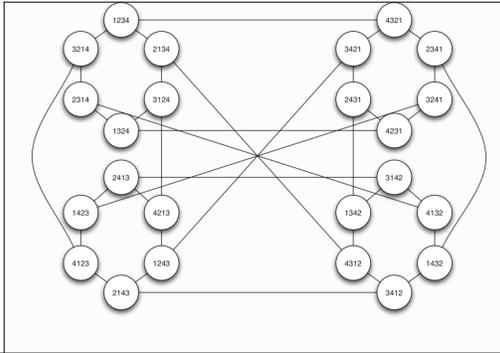
ICN Emulation

- At least 3 "generic" emulation schemes have been proposed
 - [Naor/Wieder SPAA '03]
 - [Abraham, et al. IPDPS '03]
 - [Manku PODC '03]
- As an exercise, funky ICN + emulation scheme = new DHT
 - IHOP: Internet Hashing on Pancake graphs [Ratajczak/Hellerstein '04]
 - Pancake graph¹ ICN + Abraham, et al. emulation.



¹Based on Bill Gates' only paper.
Trivia question: who was his advisor/co-author?

Pancake Topology



A "Generalized DHT"

- Pick your favorite InterConnection Network
 - Hypercube, Butterfly, DeBruijn, Chord, Pancake, etc.
- Pick an "emulation" scheme
 - To handle the "incomplete" case
- Pick a way to let new nodes choose IDs
 - And maintain load balance

PhD Thesis, Gurmeet Singh Manku, 2004

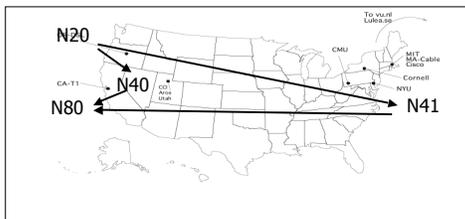
Storage Models for DHTs

- Up to now we focused on routing
 - DHTs as "content-addressable network"
- Implicit in the name "DHT" is some kind of storage
 - Or perhaps a better word is "memory" 
 - Enables indirection in time
 - But also can be viewed as a place to store things
- Soft state is the name of the game in Internet systems

A Note on Soft State

- A hybrid persistence scheme
 - Persistence via *storage & retry*
- Joint responsibility of publisher and storage node
 - Item published with a Time-To-Live (TTL)
 - Storage node attempts to preserve it for that time
 - Best effort
 - Publisher wants it to last longer?
 - Must republish it (or renew it)
- Must balance reliability and republishing overhead
 - Longer TTL = longer potential outage but less republishing
- On failure of a storage node
 - Publisher eventually republishes elsewhere
- On failure of a publisher
 - Storage node eventually “garbage collects”

Optimizing routing to reduce latency



- Nodes close on ring, but far away in Internet
- Goal: put nodes in routing table that result in few hops and low latency

Locality-Centric Neighbor Selection

- Much recent work [Gummadi, et al. SIGCOMM '03, Abraham, et al. SODA '04, Dabek, et al. NSDI 04, Rhea, et al. USENIX '04, etc.]
 - We saw flexibility in neighbor selection in Pastry/Bamboo
 - Can also introduce some randomization into Chord, CAN, etc.
- How to pick
 - Analogous to ad-hoc networks
 1. Ping random nodes
 2. Swap neighbor sets with neighbors
 - Combine with random pings to explore
 3. Provably-good algorithm to find nearby neighbors based on sampling [Karger and Ruhl 02]

Geometry and its effects

[Gummadi, et al. SIGCOMM '03]

- Some topologies allow more choices
 - Choice of neighbors in the neighbor tables (e.g. Pastry)
 - Choice of routes to send a packet (e.g. Chord)
 - Cast in terms of “geometry”
 - But really a group-theoretic type of analysis
- Having a ring is very helpful for resilience
 - Especially with a decent-sized “leaf set” (successors/predecessors)
 - Say $\sim \log n$

Handling Churn

- Bamboo [Rhea, et al, USENIX 04]
 - Pastry that doesn't go bad (?)
- Churn
 - Session time? Life time?
 - For system resilience, session time is what matters.
- Three main issues
 - Determining timeouts
 - Significant component of lookup latency under churn
 - Recovering from a lost neighbor in “leaf set”
 - Periodic, not reactive!
 - Reactive causes feedback cycles
 - Esp. when a neighbor is stressed and timing in and out
 - Neighbor selection again

Timeouts

- Recall Iterative vs. Recursive Routing
 - Iterative: Originator requests IP address of each hop
 - Message transport is actually done via direct IP
 - Recursive: Message transferred hop-by-hop
- Effect on timeout mechanism
 - Need to track latency of communication channels
 - Iterative results in direct $n \times n$ communication
 - Can't keep timeout stats at that scale
 - Solution: virtual coordinate schemes [Dabek et al. NSDI '04]
 - With recursive can do TCP-like tracking of latency
 - Exponentially weighted mean and variance
- Upshot: Both work OK up to a point
 - TCP-style does somewhat better than virtual coords at modest churn rates (23 min. or more mean session time)
 - Virtual coords begins to fail at higher churn rates

Complex Query Processing

DHTs Gave Us Equality Lookups

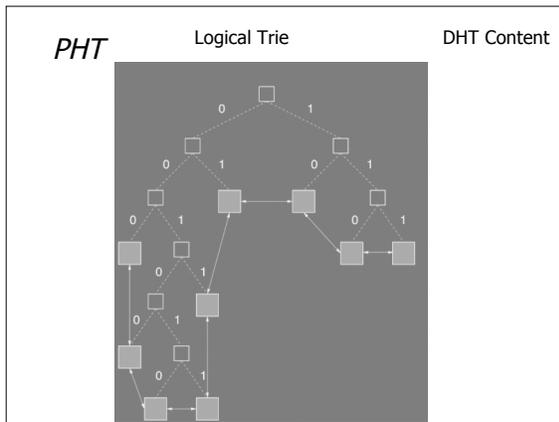
- What else might we want?
 - Range Search
 - Aggregation
 - Group By
 - Join
 - Intelligent Query Dissemination
- Theme
 - All can be built elegantly on DHTs!
 - This is the approach we take in PIER
 - But in some instances other schemes are also reasonable
 - I will try to be sure to call this out
 - The flooding/gossip strawman is always available

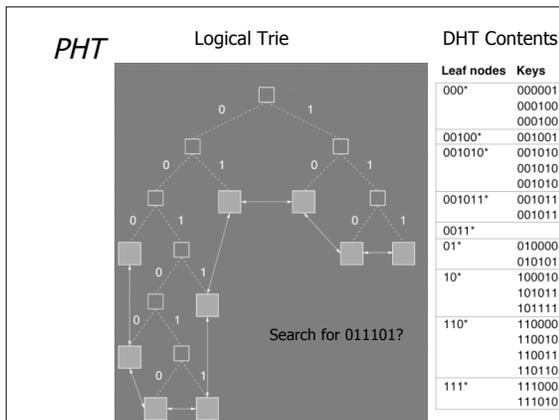
Range Search

- Numerous proposals in recent years
 - Chord w/o hashing, + load-balancing [Karger/Ruhl SPAA '04, Ganesan/Bawa VLDB '04]
 - Mercury [Bharambe, et al. SIGCOMM '04]. Specialized "small-world" DHT.
 - P-tree [Crainiceanu et al. WebDB '04]. A "wrapped" B-tree variant.
 - P-Grid [Aberer, CoopIS '01]. A distributed trie with random links.
 - (Apologies if I missed your favorite!)
- We'll do a very simple, elegant scheme here
 - Prefix Hash Tree (PHT). [Ratnasamy, et al '04]
 - Works over *any* DHT
 - Simple robustness to failure
 - Hints at generic idea: *direct-addressed distributed data structures*

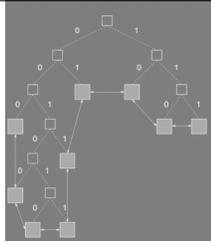
Prefix Hash Tree (PHT)

- Recall the *trie* (assume binary trie for now)
 - Binary tree structure with edges labeled 0 and 1
 - Path from root to leaf is a *prefix* bit-string
 - A key is stored at the minimum-distinguishing prefix (depth)
- PHT is a bucket-based trie addressed via a DHT
 - Modify trie to allow b items per leaf "bucket" before a split
 - Store contents of leaf bucket at DHT address corresponding to prefix
 - So far, not unlike Litwin's "Trie Hashing" scheme, but hashed on a DHT.
 - Punchline in a moment...





PHT Search



- Observe: The DHT allows *direct addressing* of PHT nodes
 - Can *jump* into the PHT at any node
 - Internal, leaf, or *below a leaf!*
 - So, can find leaf by binary search
 - $\log \log |D|$ search cost!
 - If you knew (roughly) the data distribution, even better
 - Moreover, consider a failed machine in the system
 - Equals a failed node of the trie
 - Can “hop over” failed nodes directly!
 - And... consider concurrency control
 - A link-free data structure: simple!

Reusable Lessons from PHTs

- Direct-addressing a lovely way to emulate robust, efficient “linked” data structures in the network
- Direct-addressing requires regularity in the data space partitioning
 - *E.g.* works for regular space-partitioning indexes (tries, quad trees)
 - Not so simple for data-partitioning (B-trees, R-trees) or irregular space partitioning (kd-trees)

Aggregation

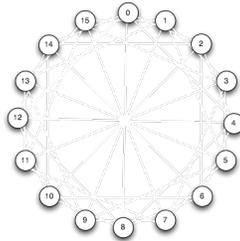
- Two key observations for DHTs
 - DHTs are multi-hop, so hierarchical aggregation can reduce BW
 - *E.g.*, the TAG work for sensornets [Madden, OSDI 2002]
 - DHTs provide tree construction in a very natural way
- But what if I don’t use DHTs?
 - Hold that thought!

An API for Aggregation in DHTs

- Uses a basic hook in DHT routing
 - When routing a multi-hop msg, intermediate nodes can intercept
- Idea
 - To aggregate in a DHT, pick an aggregating ID at random
 - All nodes send their tuples toward that ID
 - Nodes along the way intercept and aggregate before forwarding
- Questions
 - What does the resulting agg tree look like?
 - What shape of tree would be good?
- Note: tree-construction will be key to other tasks!

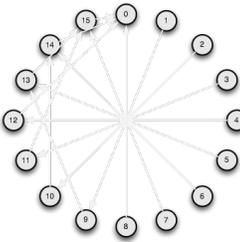
Consider Aggregation in Chord

- Everybody sends their message to node 0
- Assume greedy jumps (increasing Gon-order)
- Intercept messages and aggregate along the way



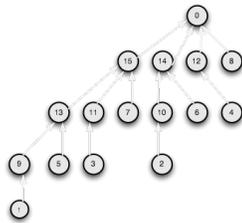
Consider Aggregation in Chord

- Everybody sends their message to node 0
- Assume greedy jumps (increasing Gon-order)
- Intercept messages and aggregate along the way



Consider Aggregation in Chord

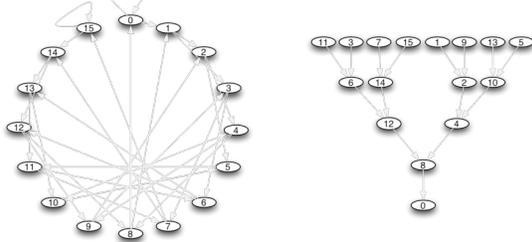
- Everybody sends their message to node 0
- Assume greedy jumps (increasing Gon-order)
- Intercept messages and aggregate along the way



Binomial Tree!!

Aggregation in Koorde

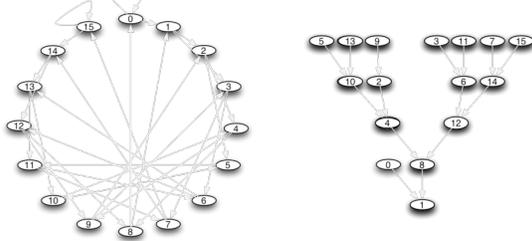
- Recall the DeBruijn graph:
 - Each node x points to $2x \bmod n$ and $(2x + 1) \bmod n$



(But note: not node-symmetric)

Aggregation in Koorde

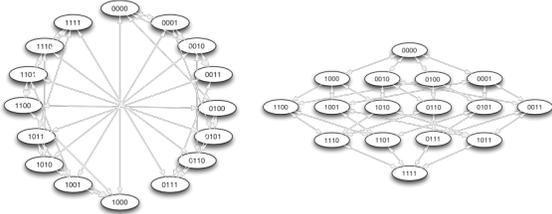
- Recall the DeBruijn graph:
 - Each node x points to $2x \bmod n$ and $(2x + 1) \bmod n$



(But note: not node-symmetric)

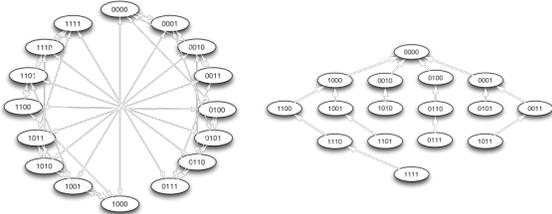
Aggregation in Pastry/Bamboo

- Depends on choice of neighbors
 - But if you flip exactly one bit each hop:



Aggregation in Pastry/Bamboo

- Depends on choice of neighbors
 - But if you flip exactly one bit:



Metrics for Aggregation Trees

- What makes a good/bad agg tree?
 - Number of edges? No!
 - Always $n-1$. With distributive/algebraic aggs, msg size is fixed.
 - Degree of fan-in
 - Affects congestion
 - Height
 - Determines latency
 - Predictability of subtree shape
 - Determines ability to control timing tightly
 - Stability in the face of churn
 - Changing tree shape while accumulating can result in errors
 - Subtree size distribution
 - Affects "jeopardy" of lost messages

So what if I don't have a DHT?

- Need another tree-construction mechanism
 - There are many in the NW literature (e.g. for multicast)
 - Require maintenance messages akin to DHTs
 - Do you maintain for the life of your query engine? Or setup/teardown as needed?
- Can pick a tree shape of your own
 - Not at the mercy of the DHT topologies
 - E.g. could do high fan-in trees to minimize latency
- As we noted before, we will reuse tree-construction for multiple purposes
 - It's handy that they're trivial in DHTs
 - But could reuse another scheme for multiple purposes as well
- Or, can do aggregation via gossip [Kempe, et al FOCIS '03]

Group By

- A piece of cake in a DHT
 - Every node sends tuples toward the hash ID of the grouping columns
 - An agg tree is naturally constructed per group
- Note nice dual-purpose use of DHT
 - Hash-based partitioning for parallel group by
 - Just like parallel DBMS (Gamma, the *Exchange* op in Volcano)
 - Agg tree construction in multi-hop overlay network

Hash Join

- We just did hash-based group by.
- Hash-based join is roughly the same deal, twice:
 - Given R,a Join S,b
 - Each node:
 - sends each R tuple toward H(R,a)
 - sends each S tuple toward H(S,b)
- Again, DHT gives
 - Hash-based partitioning for parallel hash join
 - Tree construction (no reduction along the way here, though)
- Note the resulting communication pattern
 - A tree is constructed *per hash destination!*
 - That's a lot of trees!
 - No big deal for the DHT -- it already had that topology there.

Fetch Matches Join

- Essentially a distributed index join
 - Name comes from R* (Mackert & Lohman)
- Given R.a Join S.b
 - Assume <S.b, tuple> was already "published" (indexed)
- For each tuple of R, query DHT for S tuples matching R.a
 - Each S.b value will get some subset of the nodes visiting it
 - So a lot of "partial" trees
 - Note: if S.b is *not* already indexed in the DHT via S.b, that has to happen on the fly
 - Half a hash join :-)

Symmetric Semi-Join and Bloom Join

- Query rewriting tricks from distributed DBs
- Semi-Joins a la SDD-1
 - But do it to both sides of the join
 - Rewrite R.a Join S.b as
 - (<S.ID,S.b> semi-join <R.id,R.a>) join R.a join S.b
 - Latter 2 joins can be Fetch Matches
- Bloom Joins a la R*
 - Requires a bit more finesse here
 - Aggregate R.a Bloom filters to a fixed hash ID. Same for S.b.
 - All the R.a Bloom filters are OR'ed, eventually multicasted to all nodes storing S tuples
 - Symmetric for S.b Bloom filter
 - Can in principle stream refining Bloom filters

Query Dissemination

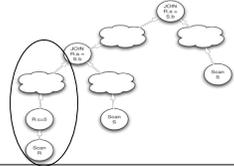
- How do nodes find out about a query?
 - Up to now we conveniently ignored this!
- Case 1: Broadcast
 - As far as we know, all nodes need to participate
 - Need to have a broadcast tree out of the query node
 - This is the opposite of an aggregation tree!
 - But how to instantiate it?
- Naïve solution: Flood
 - Each nodes sends query to all its neighbors
 - Problem: nodes will receive query multiple times
 - wasted bandwidth

SCRIBE

- Redundancy-free broadcast
- Upon joining the network, route a message to some canonical hash ID
 - Parent intercepts msg, makes a note of new child, discards message
 - At the end, each node knows its children, so you have a broadcast tree
 - Tree needs to deal with joins and leaves on its own; the DHT won't help.
 - MSR/Rice, NGC '01

Query Dissemination II

- Suppose you have a simple equality query
 - Select * From R Where R.c = 5
 - If R.c is already indexed in the DHT, can route query via DHT
- Query Dissemination is an “access method”
 - Basically the same as an index
- Can take more complex queries and disseminate sub-parts
 - Select * From R, S, T
 - Where R.a = S.b
 - And S.c = T.d
 - And R.c = 5



PIER



- Peer-to-Peer Information Exchange & Retrieval
 - Puts together many of the techniques described above
 - Aggressively uses DHTs
 - But agnostic to choice
 - Uses Bamboo, has worked on CAN and Chord
 - [Huebsch, et al. VLDB '03]
- Deployed
 - Running φ queries on ~400 nodes around the world (PlanetLab)
 - Simulated on up to 10K nodes
- Current Applications
 - Improved Filesharing
 - Internet Monitoring (φ)
 - Customizable Routing via Recursive Queries

<http://pier.cs.berkeley.edu>

DHTs in PIER

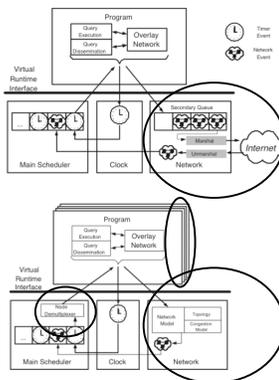
- PIER uses DHTs for:

| | |
|--|--------------|
| - Query Broadcast (TC) | DBMS Analogy |
| - Indexing (CBR + S) | Hash Index |
| - Range Indexing Substrate (CBR+S) | B+-Tree |
| - Hash-partitioned parallelism (CBR) | Exchange |
| - Hash tables for group-by, join (CBR + S) | HashJoin |
| - Hierarchical Aggregation (TC + S) | |

Key:
 TC = Tree Construction
 CBR = Content-Base Routing
 S = Storage

Native Simulation

- Entire system is event-driven
- Enables discrete-event simulation to be "slid in"
 - Replaces lowest-level networking & scheduler
 - Runs all the rest of PIER natively
- Very helpful for debugging a massively distributed system!



Initial Tidbits from PIER Efforts

- "Multiresolution" simulation critical
 - Native simulator was hugely helpful
 - Emulab allows control over link-level performance
 - PlanetLab is a nice approximation of reality
- Debugging still very hard
 - Need to have a traced execution mode.
 - Radiological dye? Intensive logging?
- DB workloads on NW technology: mismatches
 - E.g. Bamboo aggressively changes neighbors for single-message resilience/performance
 - Can wreak havoc with stateful aggregation trees
 - E.g. returning results: SELECT * from Firewalls
 - 1 MegaNode of machines want to send you a tuple!
- A relational query processor w/o storage
 - Where's the metadata?

Storage Models & Systems

Traditional FileSystems on p2p?

- Lots of projects
 - OceanStore, FarSite, CFS, Ivy, PAST, etc.
- Lots of challenges
 - Motivation & Viability
 - Short & long term
 - Resource mgmt
 - Load balancing w/heterogeneity, etc.
 - Economics come strongly into play
 - Billing and capacity planning?
 - Reliability & Availability
 - Replication, server selection
 - Wide-area replication (+ consistency of updates)
 - Security
 - Encryption & key mgmt, rather than access control

Non-traditional Storage Models

- Very long term archival storage
 - LOCKSS
- Ephemeral storage
 - Palimpsest, OpenDHT

LOCKSS

[Maniatis, et al. SOSp '04]

- Digital Preservation of Academic Materials
 - Academic publishing is moving from paper to digital *leasing*
- Librarians are scared with good reason
 - Access depends on the fate of the publisher
 - Time is unkind to bits after decades
 - Plenty of enemies (ideologies, governments, corporations)
- Goal: Preserve access for local patrons, for a very long time

Protocol Threats

- Assume conventional platform/social attacks
- Mitigate further damage through protocol
- Top adversary goal: Stealth Modification
 - Modify replicas to contain adversary's version
 - Hard to reinstate original content after large proportion of replicas are modified
- Other goals
 - Denial of service
 - System slowdown
 - Content theft

The LOCKSS Solution

- Peer-to-peer auditing and repair system for replicated documents / no file sharing
- A peer periodically audits its own replica, by calling an opinion poll
- When a peer suspects an attack, it raises an alarm for a human operator
 - Correlated failures
 - IP address spoofing
 - System slowdown
- 2nd iteration of a deployed system

Sampled Opinion Poll

- Each peer holds
 - reference list of peers it has discovered
 - friends list of peers it knows externally
- Periodically (faster than rate of bit rot)
 - Take a sample of the reference list
 - Invite them to send a hash of their replica
- Compare votes with local copy
 - Overwhelming agreement (>70%) ☞ Sleep blissfully
 - Overwhelming disagreement (<30%) ☞ Repair
 - Too close to call ☞ Raise an alarm
- To repair, the peer gets the copy of somebody who disagreed and then reevaluates the same votes

Reference List Update

- Take out voters in the poll
 - So that the next poll is based on different group
- Replenish with some "strangers" and some "friends"
 - Strangers: Accepted nominees proposed by voters
 - Friends: From the friends list
 - The measure of favoring friends is called churn factor

LOCKSS Defenses

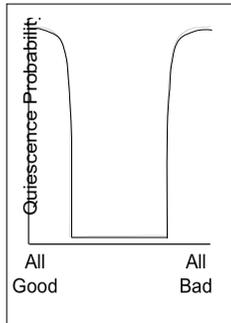
- Limit the rate of operation
- Bimodal system behavior
- Churn friends into reference list

Limit the rate of operation

- Peers determine their rate of operation autonomously
 - Adversary must wait for the next poll to attack through the protocol
- No operational path is faster than others
 - Artificially inflate “cost” of cheap operations
 - No attack can occur faster than normal ops

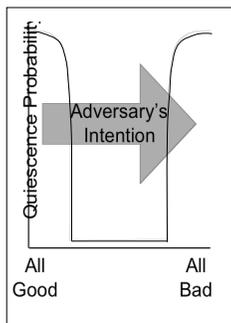
Bimodal System Behavior

- When most replicas are the same, no alarms
- In between, many alarms
- To get from mostly correct to mostly wrong replicas, system must pass through “moat” of alarming states



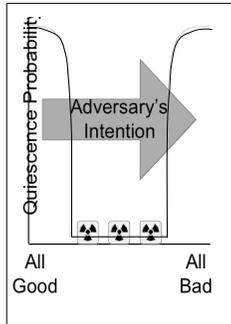
Bimodal System Behavior

- When most replicas are the same, no alarms
- In between, many alarms
- To get from mostly correct to mostly wrong replicas, system must pass through “moat” of alarming states



Bimodal System Behavior

- When most replicas are the same, no alarms
- In between, many alarms
- To get from mostly correct to mostly wrong replicas, system must pass through "moat" of alarming states



Churn Friends into Reference List

- Churn adjusts the bias in the reference list
- High churn favors friends
 - Reduces the effects of Sybil attacks
 - But offers easy targets for focused attack
- Low churn favors strangers
 - It offers Sybil attacks free reign
 - Bad peers nominate bad; good peers nominate some bad
 - Makes focused attack harder, since adversary can predict less of the poll sample
- Goal: strike a balance

Palimpsest [Roscoe & Hand, HotOS 03]

- Robust, available, secure *ephemeral* storage
- Small and very simple
- Soft-capacity – for service providers
- Congestion-based pricing
- Automatic space reclamation
- Flexible client and server policies

- *We'll ignore the economics*

Service Model for Ephemeral Storage

- For clients:
 - Data highly available for limited period of time
 - Secure from unauthorized readers
 - Resistant to DoS attacks
 - Tradeoff cost/reliability/performance
- For service providers:
 - Charging that makes economic sense
 - Capacity planning
 - Simplicity of operation and billing

How does it do this?

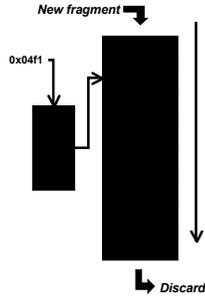
- To write a file:
 - Erasure code it
 - Route it through a network of simple block stores
 - Pay to store it
- Each block store is a fixed-length FIFO
 - Block stores may be owned by multiple providers
 - Block stores don't care who the users are
 - No one store needs to be trusted
 - Blocks are eventually lost off the end of the queue

Storing a file

- Each file has a *name* and a *key*.
- File Dispersal
 - Use a rateless code to spread blocks into fragments
 - Rabin's IDA over $GF(2^{16})$, 1024-byte blocks
- Fragment Encryption
 - Security, authenticity, identification
 - AES in Offset Codebook Mode
- Fragment Placement
 - Encrypt: $(SHA256(name) \oplus frag.id) \Rightarrow$ 256-bit ID
 - Send (fragment, ID) to a block store using DHT
 - Any DHT will do

What happens at the block store?

- Fixed-size (virtual) block stores
 - Use > 1 per node for scaling
- FIFO queue of fragments
- Indexed by fragment id
- Re-writing a fragment id moves to tail of queue
 - Note: fragment ID is *not* related to content (c.f. CFS)
- Block stores *ignore* user identity
 - No authentication needed



Retrieving a file

- Generate enough fragment IDs
- Request fragments from block stores
- Wait until n come back to you
- Decrypt and verify
- Invert the IDA
- *Voila!*

Unfortunately...

Files disappear

- This is a storage system which, in use, is *guaranteed* to forget *everything*
 - c.f. Elephant, Postgres, etc.
- Not a problem for us provided we know how long files stay around for
 - Can refresh files
 - Can abandon them
 - Note: there is no delete operation
- How do we do this?

Sampling the time constant

- Each block store has a *time constant* τ
 - How long fragment takes to reach end of queue
- Clients query block stores for τ
 - Operation piggy-backed on reads/writes
- Maintain exponentially-weighted estimate of *system* τ, τ_s
 - Fragment lifetimes Normally distributed around τ_s
- Use this to predict file lifetimes
 - Allows extensive application-specific tradeoffs

Security and Trust

Trustworthy P2P

- Many challenges here. Examples:
 - Authenticating peers
 - Authenticating/validating data
 - Stored (poisoning) and in flight
 - Ensuring communication
 - Validating distributed computations
 - Avoiding Denial of Service
 - Ensuring fair resource/work allocation
 - Ensuring privacy of messages
 - Content, quantity, source, destination
 - Abusing the power of the network
- We'll just do a sampler today

Free Riders

- Filesharing studies
 - Lots of people download
 - Few people serve files
- Is this bad?
 - If there's no incentive to serve, why do people do so?
 - What if there are strong disincentives to being a major server?

Simple Solution: Thresholds

- Many programs allow a threshold to be set
 - Don't upload a file to a peer unless it shares $> k$ files
- Problems:
 - What's k ?
 - How to ensure the shared files are interesting?

BitTorrent

- Server-based search
 - suprnova.org, chat rooms, etc. serve ".torrent" files
 - metadata including "tracker" machine for a file
- Bartered "Tit for Tat" download bandwidth
 - Download one (random) chunk from a storage peer, slowly
 - Subsequent chunks *bartered* with concurrent downloaders
 - As tracked by the tracker for the file
 - The more chunks you can upload, the more you can download
 - Download speed starts slow, then goes fast
 - Great for large files
 - Mostly videos, warez

One Slide on Game Theory

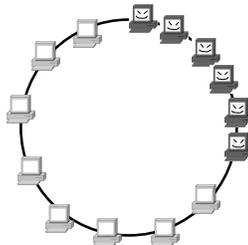
- Typical game theory setup
 - Assume self-interested (selfish) parties, acting autonomously
 - Define some benefit & cost functions
 - Parties make “moves” in the game
 - With resulting costs and benefits for themselves and others
 - A *Nash equilibrium*:
 - A state where no party increases its benefit by moving
 - Note:
 - Equilibria need not be unique nor equal
 - Time to equilibrium is an interesting computational twist
- *Mechanism Design*
 - Design the states/moves/costs/benefits of a game
 - To achieve particular globally-acceptable equilibria
 - I.e. selfish play leads to global good

DAMD P2P!

- Distributed Algorithmic Mechanism Design (DAMD)
 - A natural approach for P2P
- An Example: Fair-share storage [Ngan, et al., Fudico04]
 - Every node n maintains a *usage record*:
 - Advertised capacity
 - *Hosted list* of objects n is hosting (nodeID, objID)
 - *Published list* of objects people host for n (nodeID, objID)
 - Can publish if capacity - $p \cdot \sum(\text{published list}) > 0$
 - Recipient of publish request should check n 's usage record
 - Need schemes to authenticate/validate usage records
 - Selfish Audits: n periodically checks that the elements of its hosted list appear in published lists of publishers
 - Random Audits: n periodically picks a peer and checks all its hosted list items

Secure Routing in DHTs

- The “Sybil” attack [Douceur, IPTPS 02]
 - Register many times with multiple identities
 - Control enough of the space to capture particular traffic



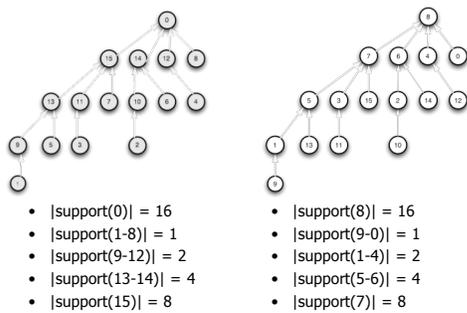
Squelching Sybil

- Certificate authority
 - Centralize one thing: the signing of ID certificates
 - Central server is otherwise out of the loop
 - Or have an “inner ring” of trusted nodes do this
 - Using practical Byzantine agreement protocols [Castro/Liskov OSDI '01]
- Weak secure IDs
 - ID = SHA-1(IP address)
 - Assume attacker controls a modest number of nodes
 - Before routing through a node, challenge it to produce the right IP address
 - Requires iterative routing

Redundant Computation

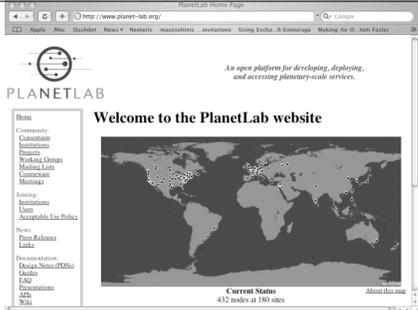
- Correctness via redundancy
 - An old idea (e.g. process pairs)
 - Applied in an adversarial environment
 - Using topological properties of DHTs
- Two Themes
 - Change “support” contents per peer across copies
 - Equalize “influence” of each peer

Example: Redundant Agg in Chord



$\log(n)$ roles w/binomial size distribution (avg = 3)

Joining the Fun



PlanetLab Home Page
An open platform for developing, deploying, and accessing planetary-scale services.

PLANETLAB

Welcome to the PlanetLab website

Home
Conferences
Institutions
Projects
Working Groups
Mailing Lists
Communities
Meetings

Security
Institutions
Name
Accessibility Use Policy

News
Press Releases
Links

Documentation
Domain Name (DNS) Info
FAQ
Downloads
API
Wiki

Current Status
432 nodes at 190 sites

About Us and

- Consortium of academia and industry
 - Catalyzed by Intel Research in 2002
 - Now hosted at Princeton U
 - 25% of SOOSP '03 papers used PlanetLab
- DB folks should get more involved!

OpenDHT

- A shared DHT service
 - The Bamboo DHT
 - Hosted on PlanetLab
 - Simple RPC API
 - You don't need to deploy or host to play with a *real* DHT!
- A playground for killer apps?
 - Needn't be as big as PIER!
 - Example: FreeDB replacement
- Research in sharing DHT svc!
 - ReDIR (Karp, et al. SPSS '04)
 - Recursive Distributed Rendezvous
 - Enables multiple apps on subsets of nodes
 - New resource mgmt scheme to do fair-share storage



OpenDHT: A Publicly Accessible DHT Service

OpenHash -> Open DHT

Open DHT is the new name for the OpenHash project. Same project, same goals, same people!

Project Overview

Open DHT is a publicly accessible DHT service; it allows any client code, running on any host on (key, value) pairs. No credentials or accounts are required to use the service, which clients invoke via RPC.

Open DHT runs on a collection of widely distributed infrastructure hosts (currently on PlanetLab).

What Does Open DHT Provide?

Open DHT makes it easy for developers everywhere to build and test new, broadly useful distributed applications. The high-availability (key, value) store provided by Open DHT takes much of the client-side rendezvous, and other communication models frequently employed by distributed applications.

Why a DHT Service?

Closing Thoughts

Much Fun to Be Had Here

- Potentially high-impact area
 - New classes of applications enabled
 - A useful question: “What apps need/deserve this scale”
 - Intensity of the scale keeps the research scope focused
 - Zero-administration, sub-peak performance, semantic homogeneity, etc.
 - A chance to reshape the Internet
 - More than just a packet delivery service
 - φ is an effort in this direction

Much Fun to Be Had Here

- Rich cross-disciplinary rallying point
 - Networks, algorithms, distributed systems, databases, economics, security...
 - Top-notch people at the table
 - Many publication venues to choose from
 - Including new ones like NSDI, IPTPS, WORLDS

Much Fun to Be Had Here

- DHT and similar overlays are a real breakthrough
 - Building block for data independence
 - Multiple metaphors
 - Hashable storage/index
 - Content-addressable routing
 - Topologically interesting tree construction
 - Each stimulates ideas for distributed computation
- Relatively solid DHT implementations available
 - Bamboo, OpenDHT (Intel & UC Berkeley)
 - Chord (MIT)

The DB Community Has Much to Offer

- Complex (multi-operator) queries & optimization
 - NW folks have tended to build single-operator "systems"
 - E.g. aggregation only, or multi-d range-search only
 - Adaptivity required
 - But may not look like adaptive QP in databases...
- Declarative language semantics
 - Deal with streaming, clock jitter and soft state!
- Data reduction techniques
 - For visualization, approximate query processing
- Bulk-computation workloads
 - Quite different from the ones the NW and systems folks envision
- Recursive query processing
 - The network *is* a graph!

Metareferences

- Your favorite search engine should find the inline refs
- Project IRIS has a lot of participants' papers online
 - <http://www.project-iris.org>
- IEEE Distributed Systems Online
 - <http://dsonline.computer.org/os/related/p2p/>
- O'Reilly OpenP2P
 - <http://www.openp2p.com>
- Karl Aberer's ICDE 2002 tutorial
 - <http://sirpeople.epfl.ch/aberer/Talks/ICDE2002-Tutorial.pdf>
- Ross/Rubenstein InfoCom 2003 tutorial
 - <http://cis.poly.edu/~ross/tutorials/P2PtutorialInfocom.pdf>
- PlanetLab
 - <http://www.planet-lab.org>
- OpenDHT
 - <http://www.opendht.org>

Some of the p2p DB groups

- PIER
 - <http://pier.cs.berkeley.edu>
- Stanford Peers
 - <http://www-db.stanford.edu/peers/>
- P-Grid
 - <http://www.p-grid.org/> (EPFL)
- Pepper
 - <http://www.cs.cornell.edu/database/pepper/pepper.htm>
- BestPeer (PeerDB)
 - <http://xena1.ddns.comp.nus.edu.sg/p2p/>
- Hyperion
 - <http://www.cs.toronto.edu/db/hyperion/>
- Piazza
 - <http://data.cs.washington.edu/p2p/piazza/>
