Topics in Database Systems: Data Management in Peer-to-Peer Systems

Unstructured Peer-to-Peer Systems

1. What is a P2P System?
   - Multiple sites (at edge)
   - Distributed resources
   - Sites are autonomous (different owners)
   - Sites are both clients and servers
   - Sites have equal functionality

What is P2P Purity

Unstructured Peer-to-Peer Systems

Based on
"Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems"  
Karl Aberer & Manfred Hauswirth 
ICDE02 Tutorial

"Architectures and Algorithms for Internet-Scale (P2P) Data Management"  
Joe Hellerstein 
VLDB 2004 Tutorial

"Open Problems in Data-Sharing Peer-to-Peer Systems",  

Περιέχεις για 29/3
1. Θα διορθώσετε το προηγούμενο σύνολο μέχρι τη Δευτέρα (28/3)
2. Θα απαντήσετε σε 2-3 ερωτήσεις πάνω στη σημερινή ύλη (στη Μέρος 2) (Θα ανακοινωθούν αύριο)

Περιέχεις για 8/3
What is P2P?
- Every participating node acts as both a client and a server ("servent").
- Every node "pays" its participation by providing access to (some of) its resources.
- Properties:
  - no central coordination
  - no central database
  - no peer has a global view of the system
  - global behavior emerges from local interactions
  - all existing data and services are accessible from any peer
  - peers are autonomous
  - peers and connections are unreliable

Overlay Networks
- P2P applications need to:
  - Track identities & (IP) addresses of peers
    - May be many!
    - May have significant Churn (update rate)
    - 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.”

Overlay Networks
- 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

P2P Cooperation Models
- Centralized model
  - Global index held by a central authority
    - Single point of failure
  - Direct contact between requestors and providers
  - Example: Napster
- Decentralized model
  - Examples: Freenet, Gnutella
  - No global index, no central coordination, global behavior emerges from local interactions, etc.
  - Direct contact between requestors and providers (Gnutella) or mediated by a chain of intermediaries (Freenet)
- Hierarchical model
  - Introduction of “super-peers”
  - Mix of centralized and decentralized model
  - Example: DNS

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?
  - Hard to beat hosted/managed services
  - p2p Google appears to be infeasible
  - [Li, et al., IPTPS 03]
- Most Resilient? Hmmm.
  - In principle more resistant to DoS attacks, etc.
  - Take, Chord: A node entering multiple times in the ring with different identities, control much of the traffic
  - 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, P2P conference
    - (classical venues (DB: VLDB, SIGMOD, ICDE DC: ICDCS, etc)

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.

Distributed Databases

- Fragmenting large databases (e.g., relational) over physically distributed nodes
- Efficient processing of complex queries (e.g., SQL) by decomposing them
- Efficient update strategies (e.g., lazy vs. eager)
- Consistent transactions (e.g., 2 phase commit)
- Normally approaches rely on central coordination

Main P2P Design Requirements

- Resource discovery
- Managing updates
- Scalability
- Robustness and fault tolerance
- Trust assessment and management

Distributed Databases vs. Peer-to-Peer

- Data distribution is a key issue for P2P systems
- Distribution Transparency
- Data Allocation and Fragmentation
- Advanced (SQL?) Query Processing
- Transactions
Usage Patterns to position P2P

Discovering information is the predominant problem
- Occasional discovery: search engines
  - E.g., new town — where is the next car rental?
- Notification: event-based systems
  - Notification for (correlated) events (event patterns)
  - E.g., notify me when my stocks drop below a threshold
- Regular discovery: P2P systems
  - Find certain type of information on a regular basis
  - E.g., search for MP3 files of Jethro Tull regularly
- Continuous information feed: push systems
  - Subscription to a certain information type
  - E.g., sports channel, updates are sent as soon as available

What is P2P?

Unstructured P2P Systems
- Napster
- Gnutella
- Freenet

The P2P Cloud

Early P2P

Early P2P I: Client-Server
- Napster
Early P2P I: Client-Server

- Napster
  - C-S search
  - "pt2pt" file xfer

xyz.mp3

Early P2P I: Client-Server

- Napster
  - C-S search
  - "pt2pt" file xfer

xyz.mp3

Early P2P I: Client-Server

- SETI@Home
  - Server assigns work units

Task: f(x)
Early P2P I: Client Server

- **SETI@Home**
  - Server assigns work units

Result: \( f(x) \)

60 TeraFLOPS!

---

**More on Napster: A brief History**

- May 1999: Napster Inc. files shared service founded by Shawn Fanning and Sean Parker
- Dec 7, 1999: Recording Industry Association of America (RIAA) sues Napster for copyright infringement
- April 13, 2000: Heavy metal rock group Metallica sues Napster for copyright infringement
- April 27, 2000: Rapper Dr. Dre sues Napster
- May 3, 2000: Metallica's attorney claims 335,000 Internet users illegally share Metallica's songs via Napster
- July 26, 2000: Court orders Napster to shut down
- Oct 31, 2000: Bertelsmann becomes a partner and drops lawsuit
- Feb 12, 2001: Court orders Napster to cease trading copyrighted songs and to prevent subscribers to gain access to content on its search index that could potentially infringe copyrights
- Feb 20, 2001: Napster offers $1 billion to record companies (rejected)
- March 2, 2001: Napster installs software to satisfy the order

---

**Napster: System Architecture**

- Central (virtual) database which holds an index of offered MP3/WMA files
- Clients(!) connect to this server, identify themselves (account) and send a list of MP3/WMA files they are sharing (C/S)
- Other clients can search the index and learn from which clients they can retrieve the file (P2P)
- Combination of client/server and P2P approaches
- First time users must register an account

---

**Napster: Communication Model**

**Napster: The Protocol [Drscholl01]**

- The protocol was never published openly and is rather complex and inconsistent
- OpenNap have reverse engineered the protocol and published their findings
- TCP is used for C/S communication
- Messages to/from the server have the following format:
  - length specifies the length of the data portion
  - type defines the message type
  - data: the transferred data
- Many fields are in plain ASCII, in many cases enclosed in double quotes (e.g., filenames such as “song.mp3” or client ids such as “nap v0.8”)

---

**Sample Messages - 1**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S Error message</td>
<td>&lt;message&gt;</td>
</tr>
<tr>
<td>2</td>
<td>C Login</td>
<td>&lt;nick&gt;&lt;pwd&gt;&lt;port&gt;&lt;client info&gt;&lt;link type&gt;</td>
</tr>
<tr>
<td>3</td>
<td>S Login ack</td>
<td>&lt;user’s email&gt;</td>
</tr>
<tr>
<td>5</td>
<td>S Add upgrade</td>
<td>&lt;version&gt;&lt;http-hostname:filename&gt;</td>
</tr>
<tr>
<td>6</td>
<td>C New user login</td>
<td>&lt;nick&gt;&lt;pwd&gt;&lt;port&gt;&lt;client info&gt;&lt;speed&gt;&lt;email address&gt;</td>
</tr>
<tr>
<td>100</td>
<td>C Client notification of shared file</td>
<td>&lt;filename&gt;&lt;md5&gt;&lt;size&gt;&lt;bitrate&gt;&lt;frequency&gt;&lt;duration&gt;</td>
</tr>
<tr>
<td>200</td>
<td>C Search request</td>
<td>[FILENAME CONTAINS &quot;artist name&quot;]</td>
</tr>
<tr>
<td>201</td>
<td>S Search response</td>
<td>&lt;filename&gt;&lt;md5&gt;&lt;size&gt;&lt;bitrate&gt;&lt;frequency&gt;&lt;duration&gt;&lt;nick&gt;&lt;ip address&gt;</td>
</tr>
<tr>
<td>202</td>
<td>S End of search response</td>
<td>(empty)</td>
</tr>
</tbody>
</table>
### Sample Messages - 2

<table>
<thead>
<tr>
<th>Type</th>
<th>C/S</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>203</td>
<td>C</td>
<td>Download request</td>
<td>&lt;nick&gt; &lt;filename&gt;</td>
</tr>
<tr>
<td>204</td>
<td>S</td>
<td>Download ack</td>
<td>&lt;nick&gt; &lt;ip&gt; &lt;port&gt; &lt;filename&gt; &lt;md5&gt; &lt;linespeed&gt;</td>
</tr>
<tr>
<td>206</td>
<td>S</td>
<td>Peer to download not available</td>
<td>&lt;nick&gt; &lt;filename&gt;</td>
</tr>
<tr>
<td>209</td>
<td>S</td>
<td>Hotlist user signed on</td>
<td>&lt;user&gt; &lt;speed&gt;</td>
</tr>
<tr>
<td>211</td>
<td>C</td>
<td>Browse a user's files</td>
<td>&lt;nick&gt;</td>
</tr>
<tr>
<td>212</td>
<td>S</td>
<td>Browse response</td>
<td>&lt;nick&gt; &lt;filename&gt; &lt;md5&gt; &lt;size&gt; &lt;bit rate&gt; &lt;frequency&gt; &lt;time&gt;</td>
</tr>
<tr>
<td>213</td>
<td>S</td>
<td>End of browse list</td>
<td>&lt;nick&gt; [&lt;ip address&gt;]</td>
</tr>
<tr>
<td>500</td>
<td>C</td>
<td>Push file to me (firewall problem)</td>
<td>&lt;nick&gt; &lt;filename&gt;</td>
</tr>
<tr>
<td>501</td>
<td>S</td>
<td>Push ack (to other client)</td>
<td>&lt;nick&gt; &lt;ip address&gt; &lt;port&gt; &quot;&lt;filename&gt;&quot; &lt;md5&gt; &lt;speed&gt;</td>
</tr>
</tbody>
</table>

---

### Client-Client Communication - 1

- Normal download (A downloads from B):
  - A connects to B's IP address/port as specified in the 204 message returned by the server (response to 203)
  - B sends the ASCII character "1"
  - A sends the string "GET"
  - B sends <mynick> <filename> <offset>
  - B returns the file size (not terminated by any special character!) or an error message such as "FILE NOT SHARED"
  - A notifies the server that the download is ongoing via a 218 message; likewise B informs the server with a 220 message
  - Upon successful completion A notifies the server with a 219 message; likewise B informs the server with a 221 message

---

### Client-Client Communication - 2

- Firewalled download (A wants to download from B who is behind a firewall):
  - A sends a 500 message to the server which in turn sends a 501 message (holding A's IP address and data port) to B
  - B connects A according to the 501 message
  - A sends the ASCII character "1"
  - B sends the string "SEND"
  - B sends <mynick> <filename> <size>
  - A returns the byte offset at which the transfer should start (plain ASCII characters) or an error message such as "INVALID REQUEST"
  - A notifies the server that the download is ongoing via a 218 message; likewise B informs the server with a 220 message
  - Upon successful completion A notifies the server with a 219 message; likewise B informs the server with a 221 message

---

### Napster: Further Services

- Additionally to its search/transfer features the Napster client offers:
  - A chat program that allows users to chat with each others in forums based on music genre, etc.
  - A audio player to play MP3 files from inside Napster
  - A tracking program to support users in keeping track of their favorite MP3s for later browsing
  - Instant messaging service

- Most of the message types in the protocol deal with hotlist, chat room, and instant messages

---

### Napster: Summary

- (Virtually) centralized system
  - Single point of failure ⇒ limited fault tolerance
  - Limited scalability (server farms with load balancing)
- Protocol is complicated and inconsistent
- Querying is fast and upper bound for the duration can be given
- "Topology is known"
- Reputation of peers is not addressed
- Many add-on services users like

---

### Early P2P II: Flooding on Overlays

- An overlay network. "Unstructured"
Gnutella: A brief History
- Developed in a 14 days “quick hack” by Nullsoft (winamp)
- Originally intended for exchange of recipes
- Timeline:
  - Published under GNU General Public License on the Nullsoft web server
  - Taken off after a couple of hours by AOL (owner of Nullsoft)
  - This was enough to “infect” the Internet
    - Gnutella protocol was reverse engineered from downloaded versions of the original Gnutella software
    - Third-party clients were published and Gnutella started to spread

Gnutella: System Architecture
- No central server
  - cannot be sued (Napster)
- Constrained broadcast
  - Every peer sends packets it receives to all of its peers (typically 4)
    - Life-time of packets limited by time-to-live (TTL) (typically set to 7)
    - Packets have unique ids to detect loops
- Hooking up to the Gnutella systems requires that a new peer knows at least one Gnutella host
  - gnutellahosts.com:6346
  - Outside the Gnutella protocol specification
Gnutella: Protocol Message Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Contained Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Pong</td>
<td>Response to a ping</td>
<td>IP address and port # of responding servant; number and total kb of files shared</td>
</tr>
<tr>
<td>Query</td>
<td>Search request</td>
<td>Minimum network bandwidth of responding servant; search criteria</td>
</tr>
<tr>
<td>QueryHit</td>
<td>Returned by servants that have the requested file</td>
<td>IP address, port # and network bandwidth of responding servant; number of results and result set</td>
</tr>
<tr>
<td>Push</td>
<td>File download requests for servants behind a firewall</td>
<td>Servent identifier, index of requested file; IP address and port to send file to</td>
</tr>
</tbody>
</table>

Gnutella: Meeting Peers (Ping/Pong)

The Protocol behind: Descriptors

- Meeting
  - GNUTELLA CONNECT/0.4
  - GNUTELLA OK

  "Descriptor header" (general packet header)
  - Descriptor ID: 16 byte unique id
  - Payload descriptor: packet type (e.g., 0x00 = Ping)
  - TTL: the number of times the descriptor will be forwarded
  - Hops: TTL(i) = TTL(i+1) + Hops(i)
  - Payload length: the length of the descriptor immediately following this header

Gnutella: Searching (Query/QueryHit/GET)

<table>
<thead>
<tr>
<th>Payload</th>
<th>TTL</th>
<th>Hops</th>
<th>Payload Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte offset</td>
<td>0</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

- Descriptor ID: 16 byte unique id
- Payload descriptor: packet type (e.g., 0x00 = Ping)
- TTL: the number of times the descriptor will be forwarded
- Hops: TTL(i) = TTL(i+1) + Hops(i)
- Payload length: the length of the descriptor immediately following this header

Ping/Pong Descriptors

- Ping (0x00): Descriptor header with payload 0x00
- Pong (0x01):
  - Port: on which the responding host can accept connections
  - IP address: of the responding host
  - Number of files shared
  - Number of kilobytes shared

Gnutella: Query Descriptor

- Query (0x80):
  - Minimum speed: the minimum network bandwidth of the servent (in kb/s) that should respond to this query
  - Search criteria: a null (i.e., 0x00) terminated string; the maximum length of this string is bounded by the "Payload length" field of the descriptor header.
QueryHit Descriptor (0x81)

<table>
<thead>
<tr>
<th>Number of hits</th>
<th>Port</th>
<th>IP address</th>
<th>Speed</th>
<th>Result set</th>
<th>Servent identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte offset</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Port: on which the responding host can accept connections</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP address: of the responding host</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speed: of the responding host (in kb/s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Servent identifier: 16-byte string uniquely identifying the servent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result set (number of hits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

File index | File size | File name

- Byte offset
- Size of the file (in bytes)
- Double null (0x0000) terminated name of the file

File Downloads

- Out of band via simplified HTTP
- Connect to IP/address given in QueryHit

Example:

```plaintext
GET /get/2468/Foobar.mp3/ HTTP/1.0
Connection: Keep-Alive
Range: bytes=0
User-Agent: Gnutella

HTTP 200 OK
Server: Gnutella
Content-type: application/binary
Content-length: 4356789

<data>...
```

Free-riding on Gnutella [Adar00]

- 24 hour sampling period:
  - 70% of Gnutella users share no files
  - 50% of all responses are returned by top 1% of sharing hosts
- A social problem not a technical one
- Problems:
  - Degradation of system performance: collapse?
  - Increase of system vulnerability
  - "Centralized" (backbone) Gnutella ⇔ copyright issues?
- Verified hypotheses:
  - H1: A significant portion of Gnutella peers are free riders.
  - H2: Free riders are distributed evenly across domains.
  - H3: Often hosts share files nobody is interested in (are not downloaded)

Free-riding Statistics - 1 [Adar00]

H1: Most Gnutella users are free riders
Of 33,335 hosts:
- 22,084 (66%) of the peers share no files
- 24,347 (73%) share ten or less files
- Top 1 percent (333) hosts share 37% (1,142,645) of total files shared
- Top 5 percent (1,667) hosts share 70% (1,142,645) of total files shared
- Top 10 percent (3,334) hosts share 87% (2,692,082) of total files shared

Free-riding Statistics - 2 [Adar00]

H3: Many servents share files nobody downloads
Of 11,585 sharing hosts:
- Top 1% of sites provide nearly 47% of all answers
- Top 25% of sites provide 98% of all answers
- 7,349 (63%) never provide a query response

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?

Categories of Queries [Sripanidkulchai01]

Categorized top 20 queries

Popularity of Queries [Sripanidkulchai01]

- Very popular documents are approximately equally popular
- Less popular documents follow a Zipf-like distribution (i.e., the probability of seeing a query for the ith most popular query is proportional to 1/ith)
- Access frequency of web documents also follows Zipf-like distributions ⇒ caching might also work for Gnutella

Topology of Gnutella [Jovanovic01]

- Power-law properties verified (“find everything close by”)
  - Backbone + outskirts

Why does it work? It’s a small World! [Hong01]

- Milgram: 42 out of 160 letters from Oregon to Boston (~ 6 hops)
- Watts: between order and randomness
  - short-distance clustering + long-distance shortcuts
Links in the small World [Hong01]

- "Scale-free" link distribution
- Scale-free: independent of the total number of nodes
- Characteristic for small-world networks
- The proportion of nodes having a given number of links n is:
  \[ P(n) = \frac{1}{n^k} \]
- Most nodes have only a few connections
- Some have a lot of links: important for binding disparate regions together

Freenet: Links in the small World [Hong01]

\[ P(n) \sim \frac{1}{n^{1.5}} \]

Freenet: "Scale-free" Link Distribution [Hong01]

Caching in Gnutella [Sripanidkulchai01]

- Average bandwidth consumption in tests: 3.5Mbps
- Best case: trace 2 (73% hit rate = 3.7 times traffic reduction)

Gnutella: New Measurements

[1] Stefan Saroiu, P. Mohsen Guoeddadle, Steven D. Gribble:
A Measurement Study of Peer-to-Peer File Sharing Systems,

Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design,

[3] Evangelos P. Markatos,

Making Gnutella-like P2P Systems Scalable,

[5] Qin Lu, Mei Cao, Edith Cohen, Kai Li, Scott Shenker:
Search and replication in unstructured peer-to-peer networks, CDS 2002: 84-95

Gnutella: Bandwidth Barriers

- Clip2 measured Gnutella over 1 month:
  - Typical query is 560 bits long (including TCP/IP headers)
  - 25% of the traffic are queries, 50% pings, 25% other
  - On average each peer seems to have 3 other peers actively connected
- Clip2 found a scalability barrier with substantial performance degradation if queries/sec > 10:
  - 10 queries/sec \* 560 bits/query
  - \* 4 (to account for the other 3 quarters of message traffic)
  - \* 3 simultaneous connections
  - \* 2000 bps
  - 10 queries/sec maximum in the presence of many dialup users
  - Won't improve (more bandwidth - larger files)
Gnutella: Summary

- Completely decentralized
- Hit rates are high
- High fault tolerance
- Adapts well and dynamically to changing peer populations
- Protocol causes high network traffic (e.g., 3.5Mbps). For example:
  - 4 connections C / peer, TTL = 7
  - 1 ping packet can cause packets
- No estimates on the duration of queries can be given
- Topology is unknown → algorithms cannot exploit it
- Free riding is a problem
- Reputation of peers is not addressed
- Simple, robust, and scalable (at the moment)

Hierarchical Networks (& Queries)

- DNS
  - Hierarchical name space ("clients" + hierarchy of servers)
  - Hierarchical routing w/aggressive caching
  - 13 managed "root servers"
- Traditional pros/cons of Hierarchical data mgmt
  - Works well for things aligned with the hierarchy
  - Esp. physical locality
  - 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

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 (e.g., caching)
  - 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.)

Summary and Comparison of Approaches

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Search Type</th>
<th>Search Cost (messages)</th>
<th>Autonomy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gnutella</td>
<td>Breadth-first search on graph</td>
<td>String comparison</td>
<td>$2^{\frac{1}{7}(7-TTL)} \times C - 1$</td>
</tr>
<tr>
<td>FreeNet</td>
<td>Depth-first search on graph</td>
<td>String comparison</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Chord</td>
<td>Implicit binary search trees</td>
<td>Equality</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>CAN</td>
<td>d-dimensional space</td>
<td>Equality</td>
<td>$O(d \log n)$</td>
</tr>
<tr>
<td>P-Grid</td>
<td>Binary prefix trees</td>
<td>Prefix</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
More on Search

Search Options
- Query Expressiveness (type of queries)
- Comprehensiveness (all or just the first (or k) results)
- Topology
- Data Placement
- Message Routing

Comparison

<table>
<thead>
<tr>
<th></th>
<th>Gnutella</th>
<th>CAN</th>
<th>Others?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressiveness</td>
<td>★★★★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>Comprehensiveness</td>
<td>★★</td>
<td>★★★</td>
<td></td>
</tr>
<tr>
<td>Autonomy</td>
<td>★★★★</td>
<td>★★</td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>★</td>
<td>★★★</td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td>★★★★</td>
<td>★★</td>
<td></td>
</tr>
<tr>
<td>Topology</td>
<td>pwr law</td>
<td>grid</td>
<td></td>
</tr>
<tr>
<td>Data Placement</td>
<td>arbitrary</td>
<td>hashing</td>
<td></td>
</tr>
<tr>
<td>Message Routing</td>
<td>flooding</td>
<td>directed</td>
<td></td>
</tr>
</tbody>
</table>

Parallel Clusters

Other Open Problems besides Search: Security

- Availability (e.g., coping with DOS attacks)
- Authenticity
- Anonymity
- Access Control (e.g., IP protection, payments,...)

Trustworthy P2P

- Many challenges here. Examples:
  - Authenticating peers
  - Authenticating/validating data
    - Storing (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
**Authenticity**

- title: origin of species
- author: charles darwin
- date: 1859
- body: In an island far, far away ...

---

**More than Just File Integrity**

- title: origin of species
- author: charles darwin
- date: 1859
- body: In an island far, far away ...
- checksum

---

**More than Fetching One File**

- T=origin
- Y=1800
- A=darwin
- B=abcd

- T=origin
- Y=1859
- A=darwin
- B=abcd

- T=origin
- Y=1859
- A=darwin
- B=abcd

---

**Solutions**

- Authenticity Function $A(doc): T \text{ or } F$
  - at expert sites, at all sites?
  - can use signature $\text{expert} \rightarrow \text{sig(doc)} \rightarrow \text{user}$
- Voting Based
  - authentic is what majority says
- Time Based
  - e.g., oldest version (available) is authentic

---

**Added Challenge: Efficiency**

- Example: Current music sharing
  - everyone has authenticity function
  - but downloading files is expensive

- Solution: Track peer behavior

---

**Issues**

- Trust computations in dynamic system
- Overloading good nodes
- Bad nodes can provide good content sometimes
- Bad nodes can build up reputation
- Bad nodes can form collectives
- ...

---
Sample Results

Fraction of malicious peers

Security & Privacy

- Issues:
  - Anonymity
  - Reputation
  - Accountability
  - Information Preservation
  - Information Quality
  - Trust
  - Denial of service attacks

P2P Challenges

- Search ✓
- Resource Management
- Security & Privacy ✓

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 \( - p \sum \text{(published list)} > 0 \)
      - Recipient of publish request should check its 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

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 (e.g., caching)
- 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.)