Peer-to-Peer Systems

Thoshitha Gamage

Southern Illinois University Edwardsville

CS 447 · Week 5

[1]

[1] Lectures influenced by Dr. Indranil Gupta's (UIUC) lecture slides on P2P systems, and **Data Communication and Networking**, 5th ed., Ch.29

P2P Characteristics

- Application-level organization of the network to flexibly share resources
- Each user contributes resources to the system
- All nodes have the same functional capabilities and responsibilities (resource contribution can differ). Direct communication between peers
- Self-managing: Correct operation of nodes doesn't depend on any central administrator
- Can be designed to offer a limited degree of anonymity
- Can create systems where capacity automatically grow with the # of clients

Key issues

- Dealing with volatile resources -- owned and operated by a multitude of different users. No guarantee on uptime, connectivity, or fault-tolerance
- Choice of (data) placement and access algorithms across nodes that balance workload, ensure availability w/o undue overhead

IP vs Overlay Routing for P2P Applications

Why an additional application-level routing mechanism?

	IP	Overlay Routing
Address Space	Limited. IPv4 2^{32} IPv6 2^{128}	Very large and flat GUID namespace $(> 2^{128})$
Load Balancing	Router loads depend on topology and associated traffic patterns	Traffic patterns independent of topology
Network Dynamics	IP routing tables updated asynchronously on a best-effort basis. Order of 1 hour magnitude	Routing tables can be updated synchronously or asynchronously. Fractions-of-a-second delay
Fault-Tolerance	Designed into the IP network by its managers	Inexpensive n-fold replication
Target ID	IP-based one-to-one mapping	Messages routed to the nearest replica
Security and Anonymity	Addressing only secure with trusted nodes. No address owner anonymity	Secure even under limited trust. A limited degree of anonymity available

Structured vs Unstructured Overlays

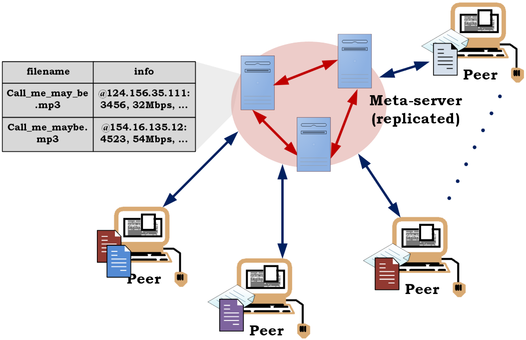
Structured Overlays

- Highly deterministic file placement. File insertion and deletion has some overhead
- Fast lookup
- Hash mapping based on a single characteristic (e.g. filename)
- Difficult to support range/keyword/attribute queries
- e.g. hypercube, mesh, de Bruijin graphs

Unstructured Overlays

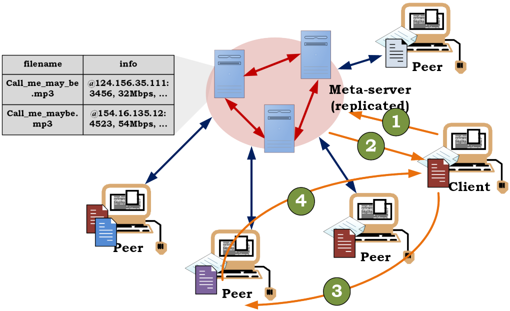
- Lose guidance for object search and storage
- Search mechanisms are ad-hoc, variants of flooding and random walk
- No structure on file placement
- Only local indexing used
- Easy peer join/departure
- Searching is costly (obviously) but supports range/keyword/attribute searching
- ``Best-effort'' search
- e.g. Gnutella

First Generation P2P: Napster



- A central meta-server maintains sharable object indexes. Peers store and provide access to actual objects
- Users (peers) are generally anonymous to each other. Meta-server provides the host/file to IP mapping
- Data-centric (not host-centric) search
- Meta-servers could be replicated for fault-tolerance

Napster Operations



1. Client requests file location from meta-server
2. Meta-server reply with a list of peers offering the file
3. Client fetch the file from the best peer (e.g. highest bandwidth)
4. Client becomes a peer for the file. Index updated

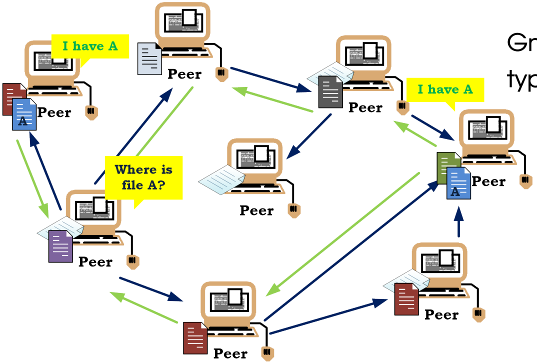
Napster Analysis

- Simple. $O(1)$ search
- Centralized indexing a potential source for congestion and bottleneck
- Potential SPoF (Can be improved through replication)
- No security: Plain text messages and passwords
- Subjected to many (historic) copyright violation law suites
- Applications that require replica consistency can hamper performance

Created by a Northeastern University Freshman in 1999

Second Generation P2P: Gnutella

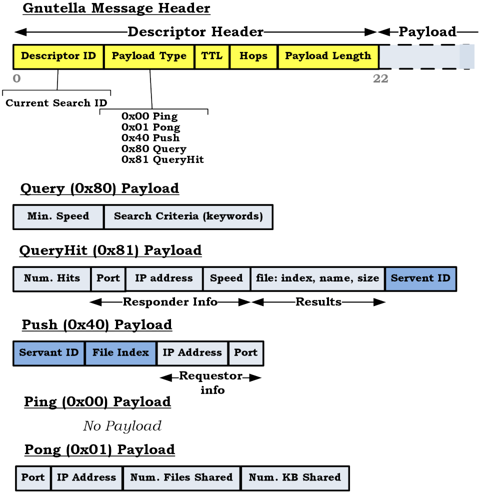
- Eliminates the central servers. Peers search and retrieve amongst themselves (unstructured!!)
- Clients also act as servers (a.k.a. **Servents**)



Gnutella has 5 main message types

- Query: Search
- QueryHit: Response to Query
- Ping: Probe network for other peers
- Pong: Reply to ping, contains address of another peer
- Push: Initiate file transfer

Gnutella Messages

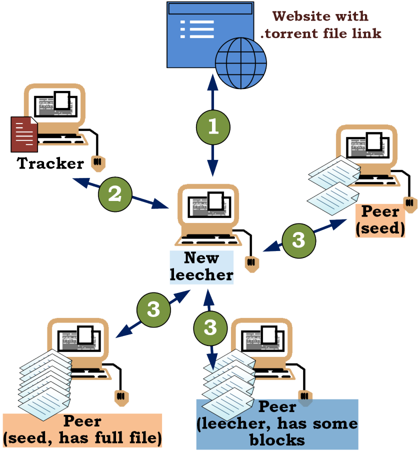


- Query Flooding, TTL-restricted, forwarded only once
- QueryHit routed on reverse path
- Requester sends Push to responder asking for file transfer. Why Push routed?
  - Responder can be behind a firewall. Push routed through HTTP GET
- Ping/Pong messages used to find peers
  - P2P systems **Churn** - peers join, leave, fail continuously
  - Periodic Ping messages used to discover active peers
  - Pings flood. Pongs reverse path routed

Gnutella Analysis

- High traffic overhead (~50%) for Ping/Pong messages
  - Multiplex, *cache*, and reduce ping/Pong frequency can help
- Repeated searchers with same keywords
  - Can be improved using cashing
  - Churning can cause freshness issues
- Require high bandwidth capable hosts
  - Modem-connected hosts can't support Gnutella
  - Can use a central (Oh no!) proxy server
  - Alternative: Create super peers (like in Skype or FastTrack)
- Large number of *freeloaders* - only download, never upload own files
- Flooding causes excessive traffic
  - Structured P2P can help in maintaining meta-information and intelligent routing

BitTorrent



1. Requester (new leecher) gets the tracker for the file from a torrent website
2. New leecher gets the peers from the tracker
   - Tracker receives heartbeats, joins, and leaves from peers per file basis
3. Leecher gets file blocks from peers
   - File split into (32KB - 256KB) blocks
   - Some peers might have all blocks (seeders), some peers might be might be leechers themselves but with required blocks

- Local Rarest First priority download -- prefer downloading least replicated blocks first
- Tit-for-Tat bandwidth usage -- Provide blocks to neighbors that provided best download rates

Distributed Hash Tables (DHT)

- Academic P2P with a structured search. Provides guaranteed lookup successes, provable bounds on search times, and scalability
- Allows insert, lookup, and delete objects with keys in a distributed manner

```
put(GUID, val) // insert val at GUID
val = get(GUID) // retrieve val from GUID
remove(GUID) // remove all references to GUID
```

- Key space distributed among peers. Each peer responsible for a portion
- Message routed among nodes looking the responsible peer. Can be a ring, tree, hypercube, etc..,
- DHT implementations: Chord, CAN, Pastry, Tapastry, Kademlia, Cassandra DHT, MS DHT, Amazon Dynamo, etc.

Chord

- A simple and elegant implementation of DHT for P2P
- Data items and nodes are m-bit identifiers, which creates a 2^m space arranged as a circle. Typically $m = 160$ (SHA-1 Hash function)

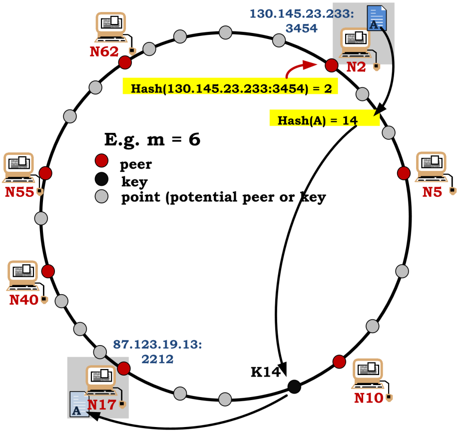
```
Peer ID : SHA-1(IP:port) → 𝒩
Key ID : SHA-1(data_to_store) → 𝒦
```

- Arithmetic in the space done using module 2^m
- Each peer has a **Finger Table** to resolve queries for **keys**. Basically a routing table
  - Each peer knows about m successors and one predecessor
  - Given a key $\mathcal{K}$, the finger table tells which peer $\mathcal{N}$ is responsible for it
  - i^{th} entry of peer n is the first peer with $\mathcal{N} \geq (n + 2^i) \mod 2^m$
  - Can also include Peer ID to IP:port mapping

Chord

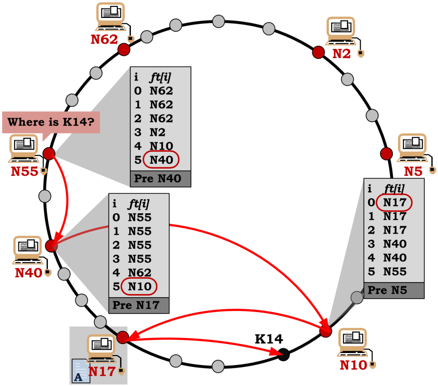
Example $m = 6$

- An example chord P2P with $m = 6$ with 7 current peers
- Peer N2 wants to share file A which has a hash value 14
- Closest peer to $K14$ is $N17$, thus $N2$ creates a reference to file A, its IP address, and port (and possibly other info) to $N17$ to be stored there
- Actual file is located at $N2$ but $N17$ holds the reference
    - Alternatively, a copy can be stored in $N17$ as well. Typically, each file is replicated at numerous peers anyway

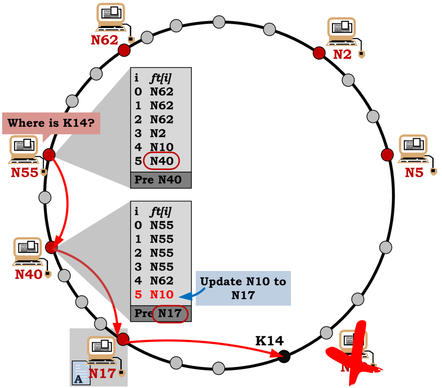


Chord Search

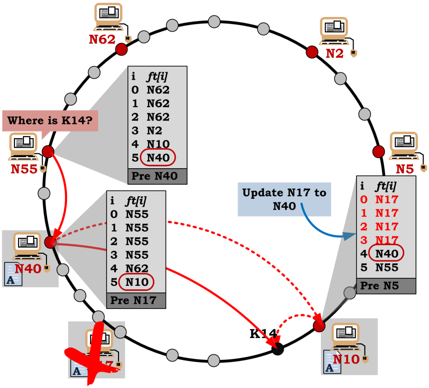
```
Lookup(key)
{
  if(node_is_responsible_for_key)
    return (node_id);
  else
    return find_successor(key);
}

find_successor(id)
{
  return find_closest_peer(id,ft[]
}
```



Chord Searching Under Failure

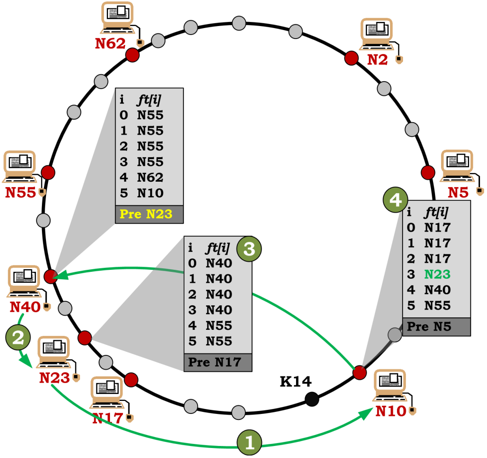


Keeping a track of multiple successors and a predecessor helps when peers fail



Files are also replicated among several neighbors to increase fault-tolerance

Chord New Peer Joining



- A new peer joining a chord P2P must have a reference to an existing peer

1. The new peer contacts the known peer with it's Peer ID
2. The known peer acts as the introducer and introduce the new peer to its immediate successor
3. The successor updates its finger table predecessor entry. The new node initializes the first successor and starts building the finger table by contacting other peers
4. All existing peers enter a stabilizing phase (read the Chord paper!!) and update their finger tables. Also, files are replicated as necessary

Chord Analysis

- File insertion and memory is also $O(log(N))$
- More structured than Gnutella

Performance

	Memory	Lookup Latency	# messages for a Lookup
Napster	$O(1)$ $(O(N))$@server	$O(1)$	$O(1)$
Gnutella	$O(N)$	$O(N)$	$O(N)$
Chord	$O(log(N))$	$O(log(N))$	$O(log(N))$

Next Week . . . Transport Layer