Internet Technology

04. Peer-to-Peer Applications

Paul Krzyzanowski
Rutgers University
Spring 2013
Peer-to-Peer (P2P) Application Architectures

- No reliance on a central server
- Machines (peers) communicate with each other
- Pools of machines (peers) provide the service
- Goals
  - Robustness
    - Expect that some systems may be down
  - Self-scalability: the system can handle greater workloads as more peers are added
“If a million people use a web site simultaneously, doesn’t that mean that we must have a heavy-duty remote server to keep them all happy?

No; we could move the site onto a million desktops and use the Internet for coordination.

Could amazon.com be an itinerant horde instead of a fixed central command post? Yes.”

– David Gelernter

*The Second Coming – A Manifesto*
Peer-to-peer applications take up a large share of Internet traffic

**North America peak hour traffic**

<table>
<thead>
<tr>
<th></th>
<th>Upstream traffic</th>
<th>Downstream traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitTorrent</td>
<td>36.8%</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>HTTP</td>
<td>9.83%</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>Skype</td>
<td>4.76%</td>
<td></td>
</tr>
<tr>
<td>Netflix</td>
<td>4.51%</td>
<td></td>
</tr>
<tr>
<td>Netflix</td>
<td>33.0%</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>YouTube</td>
<td>14.8%</td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td>12.0%</td>
<td></td>
</tr>
<tr>
<td>BitTorrent</td>
<td>5.89%</td>
<td></td>
</tr>
</tbody>
</table>

Peer to Peer applications

• P2P targets diverse solutions
  – Cooperative computation
  – Communications (e.g., Skype)
  – Exchanges, digital currency (bitcoin)
  – DNS (including multicast DNS)
  – Content distribution (e.g., BitTorrent)
  – Storage distribution

Today, we’ll focus on file distribution
Four key primitives

• Join/Leave
  – How do you join a P2P system?
  – How do you leave it?
  – Who can join?

• Publish
  – How do you advertise content?

• Search
  – How do you find a file?

• Fetch
  – How do you download the file?

Strategies:
- Central server
- Flood the query
- Route the query
Example: Napster

• **Background**
  – Started in 1999 by 19-year-old college dropout Shawn Fanning
  – Built only for sharing MP3 files
  – Stirred up legal battles with $15B recording industry
  – Before it was shut down in 2001:
    • 2.2M users/day, 28 TB data, 122 servers
    • Access to contents could be slow or unreliable

• **Big idea:** Central directory, distributed contents
  – Users register files in a directory for sharing
  – Search in the directory to find files to copy
Napster: Overview

Napster is based on a central directory

• **Join**
  – On startup, a client contacts the central server

• **Publish**
  – Upload a list of files to the central server
  – These are the files you are sharing and are on your system

• **Search**
  – Query the sever
  – Get back one or more peers that have the file

• **Fetch**
  – Connect to the peer and download the file
Napster: Discussion

• Pros
  – Super simple
  – Search is handled by a single server
  – The directory server is a single point of control
    • Provides definitive answers to a query

• Cons
  – Server has to maintain state of all peers
  – Server gets all the queries
  – The directory server is a single point of control
    • No directory server, no Napster!
Example: Gnutella

• Background
  – Created by Justin Frankel and Tom Pepper (authors of Winamp)
  – AOL acquired their company, Nullsoft in 1999
  – In 2000, accidentally released gnutella
  – AOL shut down the project but the code was released

• Big idea: create fully distributed file sharing
  – Unlike Napster, you cannot shut down gnutella
Gnutella: Overview

Gnutella is based on query flooding

• **Join**
  – On startup, a client contacts at least one node
    • Asks who its friends are
  – These become its “connected nodes”

• **Publish**
  – No need to publish

• **Search**
  – Ask connected nodes. If they don’t know, they will ask their connected nodes, and so on…
  – Once/if the reply is found, it is returned to the sender

• **Fetch**
  – The reply identifies the peer; connect to the peer via HTTP & download
An overlay network is a virtual network formed by peer connections

- Any node might know about a small set of machines
- “Neighbors” may not be physically close to you
An **overlay network** is a virtual network formed by peer connections

- Any node might know about a small set of machines
- “Neighbors” may not be physically close to you
Gnutella: Search

Initial query sent to neighbors ("connected nodes")

Query: where is file X?
If a node does not have the answer, it forwards the query.

Query: where is file X?
Gnutella: Search

If a node does not have the answer, it forwards the query. Queries have a hop count (time to live).
Gnutella: Search

• Original protocol
  – Anonymous: you didn’t know if the request you’re getting is from the originator or the forwarder
  – Replies went through the same query path

• Downloads
  – Node connects to the server identified in the reply
  – If a connection is not possible due to firewalls, the requesting node can send a push request for the remote client to send it the file
Peers do not have equal capabilities

- Network upstream and downstream bandwidth
- Connectivity costs (willingness to participate)
- Availability
- Compute capabilities
Gnutella: Enhancements

• Optimizations
  – Requester’s IP address send in query to optimize reply
  – Every node is no longer equal
    • Leaf nodes & Ultraceepers
    • Leaf nodes connect to a small number of ultraceepers
    • Ultraceepers are connected to ≥ 32 other ultraceepers
    • route search requests

• Downloads
  – Node connects to the server identified in the reply
  – If a connection is not possible due to firewalls, the requesting node can send a push request for the remote client to send it the file
Gnutella: Summary

• Pros
  – Fully decentralized design
  – Searching is distributed
  – No control node – cannot be shut down
  – Open protocol

• Cons
  – Flooding is inefficient:
    • Searching may require contacting a lot of systems; limit hop count
  – Well-known nodes can become highly congested
  – In the classic design, if nodes leave the service, the system is crippled
Example: FastTrack/Kazaa

• Background
  – Kazaa & FastTrack protocol created in 2001
  – Team of Estonian programmers – same team that will later create Skype
  – Post-Napster and a year after Gnutella was released
  – FastTrack: used by others (Grokster, iMesh, Morpheus)
    • Proprietary protocol; Several incompatible versions

• Big idea: Some nodes are better than others
  – A subset of client nodes have fast connectivity, lots of storage, and fast processors
  – These will be used as supernodes (similar to gnutella’s ultrapeers)
  – Supernodes:
    • serve as temporary indexing servers for slower clients
    • Know other supernodes
Kazaa: Discussion

Selective flooding of queries

• Join
  – A client contacts a supernode

• Publish
  – Client sends a list of files to a supernode

• Search
  – Send a query to the supernode
  – Supernodes flood the query to other supernodes

• Fetch
  – Download the file from the peer with the content
Kazaa: Supernodes
Kazaa: publish a file

I have X
Kazaa: search

Supernodes answer for all their peers (ordinary nodes)
Kazaa: Summary

• Pros
  – Similar to improved Gnutella
  – Efficient searching via supernodes
  – Flooding restricted to supernodes

• Cons
  – Can still miss files
  – Well-known supernodes provide opportunity to stop service
BitTorrent

- **Background**
  - Introduced in 2002 by Bram Cohen
  - **Motivation**
    - Popular content exhibits temporal locality: *flash crowds*
      - E.g., slashdot effect, CNN on 9/11, new movies, new OS releases

- **Big idea:** allow others to download from you while you are downloading
  - Efficient fetching, not searching
  - Single publisher, many downloaders
BitTorrent: Overview

Enable downloads from peers

• Join
  – No need to join
    (seed registers with tracker server; peers register when they download)

• Publish
  – Create a torrent file that contains the metadata for the file and defines the initial tracker of the file.
  – Place the file with the content on some seed node and inform the tracker

• Search
  – Outside the BitTorrent protocol
  – Find the tracker for the file you want, contact it to get a list of peers with files

• Fetch
  – Download chunks of the file from our peers
  – At the same time, other peers may request chunks from you
BitTorrent: Publishing & Fetching

• To distribute a file
  – Create a .torrent file
    • Contains name, size, hash of each chunk, address of a tracker server.
  – Start a seed node, which contains initial copy of the full file
  – Start the tracker for the file
    • Tracker manages uploading & downloading of the content

• To get a file
  – Get a .torrent file
  – Contact tracker named in the file
    • Get the list of seeders and other nodes with portions of the file
    • Tracker will also announce you to others
  – Contact a random node for a list of file chunk numbers
  – Request a random block of the file

March 19, 2013
2013 Paul Krzyzanowski
Tracker identifies:
(1) initial system(s) that has 100% of the file (the seed)
(2) which machines have some chunks of the file downloaded

When a peer finished downloading a file, it may become a seed and remain online without downloading any content.
BitTorrent Summary

• Pros
  – Scales well; performs well when many participants
  – Gives peers an incentive to share
    • It is sometimes not possible to download without offering to upload

• Cons
  – Search is not a part of the protocol; relies on torrent index servers
  – Files need to be large for this to work well
  – Rare files do not offer distribution
  – A tracker needs to be running to bootstrap the downloads
The end