



## Peer-to-Peer File Sharing

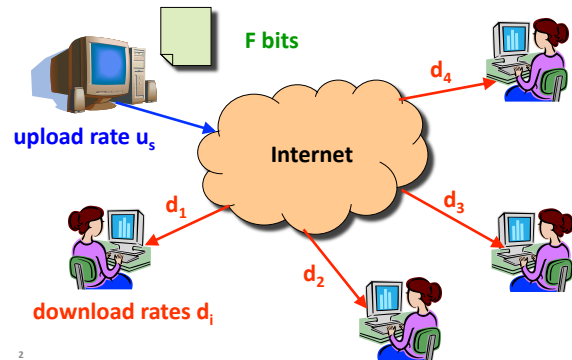
Mike Freedman

COS 461: Computer Networks

Lectures: MW 10-10:50am in Architecture N101

<http://www.cs.princeton.edu/courses/archive/spr13/cos461/>

## Server Distributing a Large File



## Server Distributing a Large File

- Sending an  $F$ -bit file to  $N$  receivers
  - Transmitting  $NF$  bits at rate  $u_s$
  - ... takes at least  $NF/u_s$  time
- Receiving the data at the slowest receiver
  - Slowest receiver has download rate  $d_{min} = \min_i \{d_i\}$
  - ... takes at least  $F/d_{min}$  time
- Download time:  $\max\{NF/u_s, F/d_{min}\}$

3

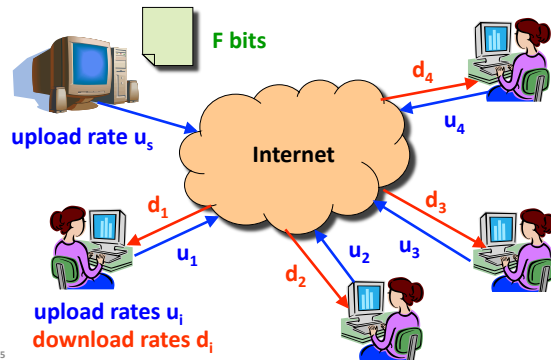
## Speeding Up the File Distribution

- Increase the server upload rate
  - Higher link bandwidth at the server
  - Multiple servers, each with their own link
- Alternative: have the receivers help
  - Receivers get a copy of the data
  - ... and redistribute to other receivers
  - To reduce the burden on the server



4

## Peers Help Distributing a Large File



5

## Peers Help Distributing a Large File

- **Components of distribution latency**
  - Server must send each bit: min time  $F/u_s$
  - Slowest peer must receive each bit: min time  $F/d_{min}$
- **Upload time using all upload resources**
  - Total number of bits:  $NF$
  - Total upload bandwidth  $u_s + \sum_i(u_i)$
- **Total:  $\max\{F/u_s, F/d_{min}, NF/(u_s + \sum_i(u_i))\}$**

6

## Peer-to-Peer is Self-Scaling

- **Download time grows slowly with N**
  - Client-server:  $\max\{NF/u_s, F/d_{min}\}$
  - Peer-to-peer:  $\max\{F/u_s, F/d_{min}, NF/(u_s + \sum_i(u_i))\}$
- **But...**
  - Peers may come and go
  - Peers need to find each other
  - Peers need to be willing to help each other

7

## Locating the Relevant Peers

- **Three main approaches**
  - Central directory (Napster)
  - Query flooding (Gnutella)
  - Hierarchical overlay (Kazaa, modern Gnutella)
- **Design goals**
  - Scalability
  - Simplicity
  - Robustness
  - Plausible deniability

8

## Peer-to-Peer Networks: Napster


- **Napster history: the rise**
  - 1/99: Napster version 1.0
  - 5/99: company founded
  - 12/99: first lawsuits
  - 2000: 80 million users
- **Napster history: the fall**
  - Mid 2001: out of business due to lawsuits
  - Mid 2001: dozens of decentralized P2P alternatives
  - 2003: growth of pay services like iTunes



Shawn Fanning,  
Northeastern freshman

9

## Napster Directory Service

- **Client contacts Napster (via TCP)** 
  - Provides a list of music files it will share
  - ... and Napster's central server updates the directory
- **Client searches on a title or performer**
  - Napster identifies online clients with the file
  - ... and provides their IP addresses
- **Client requests the file from the chosen supplier**
  - Supplier transmits the file to the client
  - Both client and supplier report status to Napster

10

## Napster Properties

- **Server's directory continually updated**
  - Always know what music is currently available
  - Point of vulnerability for legal action
- **Peer-to-peer file transfer**
  - No load on the server
  - Plausible deniability for legal action (but not enough)
- **Bandwidth**
  - Suppliers ranked by apparent bandwidth and response time

11

## Napster: Limitations of Directory

- **File transfer is decentralized, but locating content is highly centralized**
  - Single point of failure
  - Performance bottleneck
  - Copyright infringement
- **So, later P2P systems were more distributed**
  - Gnutella went to the other extreme...

12

## Peer-to-Peer Networks: Gnutella

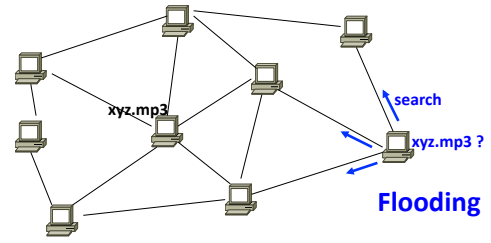
- **Gnutella history**
  - 2000: J. Frankel & T. Pepper released Gnutella
  - Soon after: many other clients (e.g., Morpheus, Limewire, Bearshare)
  - 2001: protocol enhancements, e.g., “ultrapeers”
- **Query flooding**
  - Join: contact a few nodes to become neighbors
  - Publish: no need!
  - Search: ask neighbors, who ask their neighbors
  - Fetch: get file directly from another node



gnutella.com

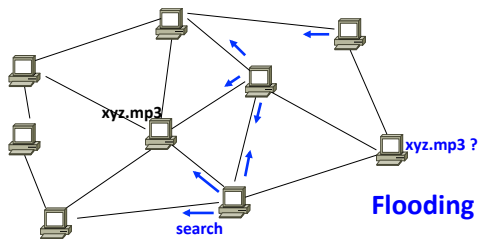
13

## Gnutella: Search by Flooding



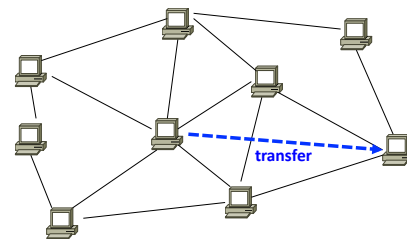
14

## Gnutella: Search by Flooding



15

## Gnutella: Search by Flooding



16

## Gnutella: Pros and Cons

- **Advantages**
  - Fully decentralized
  - Search cost distributed
  - Processing per node permits powerful search semantics
- **Disadvantages**
  - Search scope may be quite large
  - Search time may be quite long
  - High overhead, and nodes come and go often

17

## Lessons and Limitations

- **Client-Server performs well**
  - But not always feasible: Performance not often key issue!

For the following, you should choose a system that's  
(A) Flood-based (B) DHT-based (C) Either (D) Neither

- Organic scaling
- Decentralization of visibility and liability
- Finding popular stuff
- Finding unpopular stuff
- Fancy local queries
- Fancy distributed queries
- Prevent data poisoning
- Performance guarantees

18

## Lessons and Limitations

- **Client-Server performs well**
  - But not always feasible: Performance not often key issue!

For the following, you should choose a system that's  
(A) Flood-based (B) DHT-based (C) Both (D) Neither

- Organic scaling: C
- Decentralization of visibility and liability: C
- Finding popular stuff: A (maybe C)
- Finding unpopular stuff: B
- Fancy local queries: A
- Fancy distributed queries: D
- Prevent data poisoning: B (depends on query interface)
- Performance guarantees: B

19

## Peer-to-Peer Networks: KaZaA

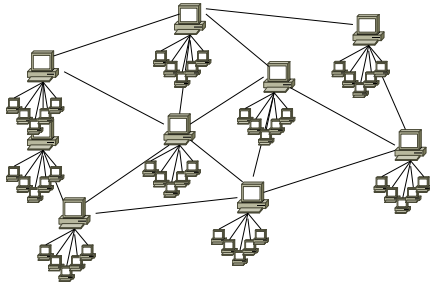
- **KaZaA history**
  - 2001: created by Dutch company (Kazaa BV)
  - Single network called FastTrack used by other clients as well
  - Eventually protocol changed so others could no longer use it
- **Super-node hierarchy**
  - Join: on start, the client contacts a super-node
  - Publish: client sends list of files to its super-node
  - Search: queries flooded among super-nodes
  - Fetch: get file directly from one or more peers



20

## “Ultra/super peers” in KaZaA and later Gnutella

21



## KaZaA: Motivation for Super-Nodes

- **Query consolidation**
  - Many connected nodes may have only a few files
  - Propagating query to a sub-node may take more time than for the super-node to answer itself
- **Stability**
  - Super-node selection favors nodes with high up-time
  - How long you’ve been on is a good predictor of how long you’ll be around in the future

22

## Peer-to-Peer Networks: BitTorrent

- **BitTorrent history**
  - 2002: B. Cohen debuted BitTorrent
- **Emphasis on efficient fetching, not searching**
  - Distribute same file to many peers
  - Single publisher, many downloaders
- **Preventing free-loading**
  - Incentives for peers to contribute

23



## BitTorrent: Simultaneous Downloads

- **Divide file into many chunks (e.g., 256 KB)**
  - Replicate different chunks on different peers
  - Peers can trade chunks with other peers
  - Peer can (hopefully) assemble the entire file
- **Allows simultaneous downloading**
  - Retrieving different chunks from different peers
  - And uploading chunks to peers
  - Important for very large files

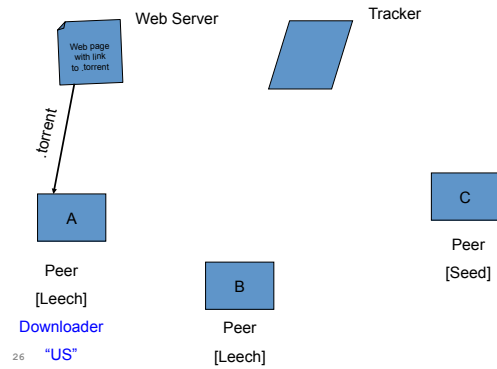
24

## BitTorrent: Tracker

- **Infrastructure node**
  - Keeps track of peers participating in the torrent
  - Peers registers with the tracker when it arrives
- **Tracker selects peers for downloading**
  - Returns a random set of peer IP addresses
  - So the new peer knows who to contact for data
- **Can have “trackerless” system**
  - Using distributed hash tables (DHTs)

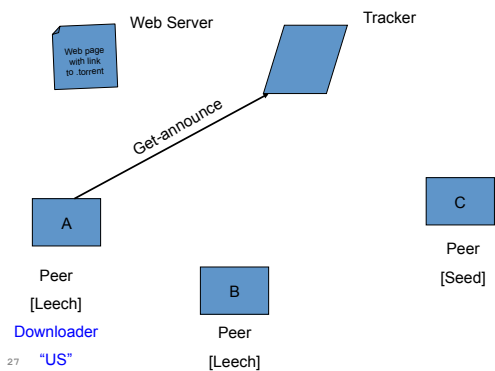
25

## BitTorrent: Overall Architecture



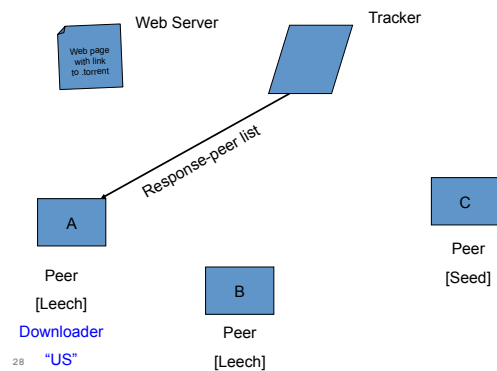
26

## BitTorrent: Overall Architecture



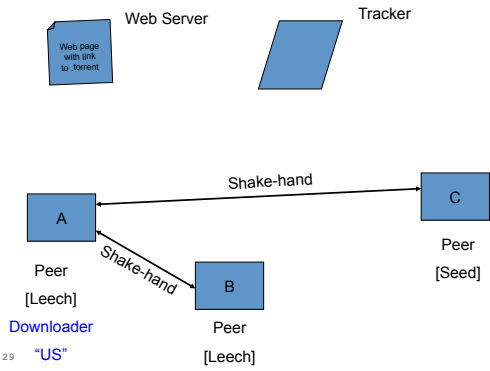
27

## BitTorrent: Overall Architecture

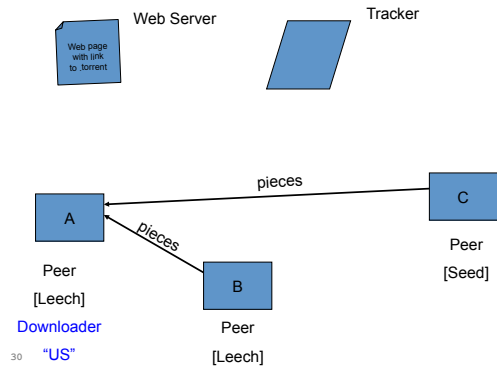


28

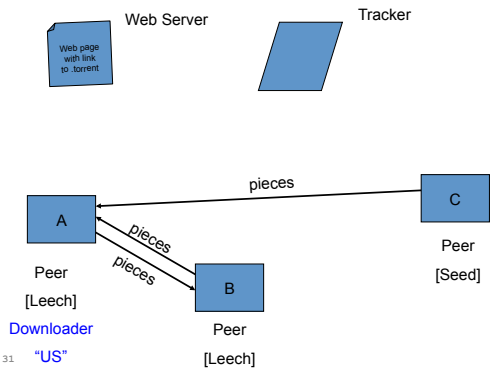
## BitTorrent: Overall Architecture



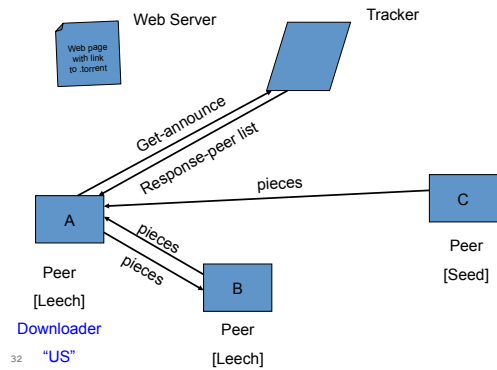
## BitTorrent: Overall Architecture



## BitTorrent: Overall Architecture



## BitTorrent: Overall Architecture





## BitTorrent: Chunk Request Order

- Which chunks to request?
  - Could download in order
  - Like an HTTP client does
- Problem: many peers have the early chunks
  - Peers have little to share with each other
  - Limiting the scalability of the system
- Problem: eventually nobody has rare chunks
  - E.g., the chunks need the end of the file
  - Limiting the ability to complete a download
- Solutions: random selection and rarest first

33

## BitTorrent: Rarest Chunk First

- Which chunks to request first?
  - Chunk with fewest available copies (i.e., rarest chunk)
- Benefits to the peer
  - Avoid starvation when some peers depart
- Benefits to the system
  - Avoid starvation across all peers wanting a file
  - Balance load by equalizing # of copies of chunks

34

## Free-Riding in P2P Networks

- Vast majority of users are free-riders
  - Most share no files and answer no queries
  - Others limit # of connections or upload speed
- A few “peers” essentially act as servers
  - A few individuals contributing to the public good
  - Making them hubs that basically act as a server
- BitTorrent prevent free riding
  - Allow the fastest peers to download from you
  - Occasionally let some free loaders download

35

## Bit-Torrent: Preventing Free-Riding

- Peer has limited upload bandwidth
  - And must share it among multiple peers
  - Tit-for-tat: favor neighbors uploading at highest rate
- Rewarding the top four neighbors
  - Measure download bit rates from each neighbor
  - Reciprocate by sending to the top four peers
- Optimistic unchoking
  - Randomly try a new neighbor every 30 seconds
  - So new neighbor has a chance to be a better partner

36

## BitTyrant: Gaming BitTorrent

- **BitTorrent can be gamed, too**
  - Peer uploads to top N peers at rate  $1/N$
  - E.g., if  $N=4$  and peers upload at 15, 12, 10, 9, 8, 3
  - ... peer uploading at rate 9 gets treated quite well
- **Best to be the  $N^{\text{th}}$  peer in the list, rather than 1<sup>st</sup>**
  - Offer just a bit more bandwidth than low-rate peers
  - And you'll still be treated well by others
- **BitTyrant software** <http://bittyrant.cs.washington.edu/>
  - Uploads at higher rates to higher-bandwidth peers

37

## Conclusions

- **Finding the appropriate peers**
  - Centralized directory (Napster)
  - Query flooding (Gnutella)
  - Super-nodes (KaZaA)
- **BitTorrent**
  - Distributed download of large files
  - Anti-free-riding techniques
- **Great example of how change can happen so quickly in application-level protocols**

38