## CS 555: Distributed Systems
## [P2P Systems: Gnutella & BitTorrent]

Shrideep Pallickara
Computer Science
Colorado State University



## Frequently asked questions from the previous class survey



## Topics covered in this lecture

- Unstructured P2P Systems
- Strategies for effective search in unstructured P2P systems
- Gnutella
- BitTorrent



## Unstructured P2P Systems



## Structured P2P systems [Summary]

- Overall **global policy** governing
  - Topology of the network
  - Placements of objects
  - Routing functions to locate objects
- There is a specific **distributed data structure** that underpins
  - Associated Overlay
  - Algorithms that operate on it to route messages



## Structured P2P systems [Summary]

- Because of the structure, algorithms are
  - Efficient
  - Offer *time-bounds* on object location
- BUT involve **costly maintenance** of underlying structures
  - In highly dynamic environments

## Unstructured P2P systems [1/2]

- Target the maintenance argument
- No overall control on
  - Topology
  - Placements of objects within the network
- Overlay is created in an ***ad hoc*** manner
  - Each node joins network by following simple, local rules to establish connectivity



## Unstructured P2P systems [2/2]

- A new joining node will establish contact with a set of *neighbor* nodes
  - These neighbors will be connected to further neighbors, etc.
- The network is fundamentally **decentralized** and self-organizing
  - Resilient to failures



## Locating objects in unstructured P2P systems

- Requires a search of the resultant network topology
- No guarantees of being able to find the object
  - Performance will also be unpredictable
  - There is a risk of generating excessive message traffic to locate objects



## Pros and Cons

| | Structured P2P | Unstructured P2P systems |
|---|---|---|
| Advantages | Guaranteed to locate objects with bounds on this operation. Low message overhead | Self-organizing and naturally resilient to failures |
| Disadvantages | Maintain complex overlay structures that are difficult and costly in dynamic settings | Probabilistic<br>Cannot offer absolute guarantees on locating objects |



## STRATEGIES FOR EFFECTIVE SEARCH IN UNSTRUCTURED P2P SETTINGS



## Sharing in unstructured P2P networks

- All nodes in the network offer files to the greater environment
- Problem of locating a file?
  - Maps onto a *search of the whole network*
- CAVEAT:
  - If implemented naively, could result in **flooding** the network with requests

## Refinements for search in unstructured P2P systems

- Expanded ring search
- Random walks
- Gossiping
- Replication



## Refinements for search in unstructured P2P systems: Expanded Ring Search

- Initiating node carries out a series of searches with *increasing values* in the TTL field
- A significant number of searches will likely be satisfied locally (proximate peers)
  - Expand the scope of search only if requests fail in the neighborhood



## Refinements for search in unstructured P2P systems: Random Walks

- Initiating node sets of a number of **walkers**
- Walkers follow *random pathways* through the interconnected graph
  - Over the unstructured network



## Refinements for search in unstructured P2P systems: Gossiping

- Node sends request to a neighbor with a certain probability
- Requests propagate through the network in a manner that is similar to **viral propagations**
  - Such gossip protocols are also referred to as *epidemic protocols*



## Refinements for search in unstructured P2P systems: Gossiping

- Probabilities may either be
  - **Fixed** for a given network
  - Computed **dynamically** based on:
    - Past experience
    - Current context



## Refinements for search in unstructured P2P systems: Replication

- **Replicate** content across a number of peers
- *Probability* of efficient discovery for particular files is enhanced
- Replications can be for
  - The entire file
  - Fragments thereof

## GNUTELLA



## Gnutella

- Launched in 2000
- One of the most dominant and influential peer-to-peer file sharing applications



## Gnutella: Early Versions (0.4)

- Every node forwarded a request to **each** of its neighbors
- Neighbors, in turn, passed this on to their neighbors
  - Until a match was found
- This is **flooding**



## Gnutella: Early Versions (0.4)

- Search was **constrained** with a *time-to-live* (TTL) field limiting the number of hops
- At the time of Version 0.4, average peer connectivity was 5 neighbors per-node



## GNUTELLA
## VERSION 0.6 AND LATER



## Addressing deficiencies in scaling: Hybrid Architecture [1/2]

- Move away from classic P2P where all nodes are equal
- Some nodes are elected as **ultrapeers**
  - Form the heart of the network
- Other nodes take on the role of **leaf nodes**
- Peers still cooperate to offer service

## Addressing deficiencies in scaling: Hybrid Architecture [2/2]

- Leaves connect to a small number of **ultrapeers**
- Ultrapeers are *densely connected* to other ultrapeers
- Effect?
  - Dramatically *reduces* the maximum number of hops for exhaustive search



## Query Routing Protocol [1/2]

- Designed to **reduce** the number of queries issued by nodes
- **Exchange information** about files contained on nodes
- **Forward queries** down paths where the system thinks there will be a positive outcome



## Query Routing Protocol [2/2]

- Does not share information about files directly
- Protocol produces **set of numbers**
  - By *hashing on individual words* in a file-name
  - For e.g., "`Gone with the wind`" will be represented as `<36, 789, 452, 132>`
- Each node produces a **Query Routing Table**
  - Contains hash values representing *each of the files* contained on that node
  - Sends it to all its associated ultrapeers



## Query Routing Protocol: Ultrapeers

- Ultrapeers produce their own Query Routing Table
  - **Union** of all entries *from all connected leaves*; together with entries for files at that ultrapeer
- The ultrapeer then **exchanges** its Query Routing Table with other ultrapeers



## Implications of exchanging the Query Routing Table

- Ultrapeers can determine which paths offer a **valid route** for a given query
  - Significantly reduces amount of unnecessary traffic
- Ultrapeer **forwards** a query to a node *only if a match is found*
  - Match indicates that the node has the file
  - Same check performed before forwarding query to another ultrapeer



## Avoid overloading the ultrapeers

- Nodes send query to *one* ultrapeer at a time
  - Wait for a specified time period
- ***Avoid reverse traversal*** of messages through the graph
  - Queries in Gnutella contain network address of the initiating ultrapeer
  - File sent directly (using UDP) to that ultrapeer

## BITTORRENT



## Bit Torrent: Traffic statistics

- In November 2004
  - Responsible for 25% of all Internet traffic
- February 2013
  - 3.35% of all worldwide bandwidth
  - > 50% of the 6% total bandwidth dedicated to file sharing



## BitTorrent

- Designed for downloading **large files**
- Not intended for real-time routing of content
- Relies on capabilities of ordinary user machines



## Bit Torrent: Key concepts

- Instead of downloading a file from a single source server
  - Users join a **swarm** of hosts to upload-to/download-from *simultaneously*
- Several basic commodity computers can replace large, customized servers
  - Without compromising on efficiency
  - In fact, lower bandwidth usage with swarms *prevents* large internet traffic spikes



## Segmented file transfer [1/2]

- File being transferred is divided into fixed-size segments called **chunks** (or pieces)
  - Chunks are of the same size throughout a single download (10MB file: 10 1MB chunks or 40 256KB chunks)
- Chunks are downloaded non-sequentially and rearranged into the correct order by BitTorrent



## Segmented file transfer [2/2]

- Advantages:
  - File transfers can be stopped at any time and *resumed*
    - Without loss of previously downloaded content
  - Clients seek out readily available chunks, rather than waiting for an unavailable (next in sequence) chunk

## BitTorrent: Protocol summary

- Splits files into fixed-sized **chunks**
- Chunks are then made available at various peers across the P2P network
- Clients can *download* a number of chunks **in parallel** from different sites
  - Reduces the burden on a particular peer to service the entire download



## The BitTorrent protocol

- When a file is made available in BitTorrent, a `.torrent` file is created
  - Holds **metadata** associated that file
- Metadata
  - The name and length of the file
  - Location of a **tracker** (URL)
    - Centralized server that manages download for that file
  - Checksum
    - Associated with each chunk
    - Generated using the SHA-1 algorithm



## Advantages of hashing chunks

- Each chunk has a cryptographic hash in the torrent descriptor
- Modifications of chunks can be reliably detected
  - Prevents accidental and malicious modifications
- If a node starts with an authentic/legitimate torrent descriptor?
  - It can verify the authenticity of the entire file that it receives



## The swarm or torrent for a particular file includes

- Tracker
- Seeders
- Leechers



## Trackers

- The use of trackers, compromises a core P2P principle
  - But *simplifies* the system
- Trackers are responsible for **tracking the download status** for a particular file



## The roles of participants in BitTorrent: Seeder

- Peer with a complete version of a file (i.e. with all its chunks) is known as a **seeder**
- Peer that initially creates the file, provides the initial seed for file distribution

## The roles of participants in BitTorrent: Leechers

- Peers that want to download a file are known as **leechers**
  - A given leecher, at any given time, contains a number of chunks for that file
- Once a leecher downloads all chunks for a file, it can become a seeder for subsequent downloads
  - Files **spread virally** based on demand



## When a peers wants to download a file

- Contacts the tracker
- Is given a **partial view** of the torrent
  - The set of peers that can support the download
  - The tracker does not participate in scheduling the downloads
    - Decentralized
- Chunks are requested and transmitted in **any order**



## Incentive mechanism: Tit-for-tat

- Gives downloading *preference* to peers who have previously uploaded to the site
  - Encourages concurrent uploads/downloads to make better use of bandwidth
- A peer supports downloads from $n$ simultaneous peers by **unchoking** these peers
  - Decisions based on rolling calculations of download rates



## Scheduling downloads

- **Rarest first** scheduling policy
- Peer prioritizes chunk that is *rarest* among its set of connected peers
- Ensures that chunks that are not widely available, spread rapidly



## How BitTorrent differs from a classic download

| | BitTorrent | Classic download |
|---|---|---|
| Connections | Many small data requests over different IP connections to different machines | One TCP connection to one machine |
| Download Order | Random or "rarest first" to ensure high-availability | Sequential |

** Allows BitTorrent to achieve lower cost, higher redundancy, and resistance to abuse



## BitTorrent: Advantages

- Advantages
  - Lower costs, greater redundancy, higher resistance to abuse or "flash crowds"
- Shortcomings
  - Non-contiguous download precludes progressive download
  - No streaming playback
    - Beta BitTorrent Streaming protocol available for testing in 2013

## BitTorrent: Shortcomings

- Downloads can take time to rise to full speed
  - May take time to enough peer connections to be established
  - Takes time for a node to receive data to become an effective uploader
- Regular (non-BitTorrent/traditional) downloads on the other hand:
  - Rise to full speed very quickly and maintain this speed throughout



## But how do you find a torrent?

- Browsing the web or by some other means
  - Open it with a BitTorrent client
- Client connects to trackers in the torrent file and finds peers
  - If swarm contains only the initial seeder, client connects directly to it and begins to request pieces



## Support for trackerless Torrents

- Azureus (now Vuze) supported this first
- Mainline BitTorrent provides a DHT based implementation
  - Mainline DHT
  - Kademlia-based Distributed Hash Table (DHT) used by BitTorrent clients



## The contents of this slide-set are based on the following references

- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 10]
- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van der Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273.* [Chapter 5]