Today’s class

- Users are not interested in connecting to a specific computer, they are interested in downloading a specific content

- Content distribution
  - CDNs
  - Unstructured P2P
  - Structured P2P
Content distribution networks

- Primarily cache based systems
- Basic premise: It is faster and cheaper to get data that is closer to here than closer to there.
- “There” is the origin server
- “Here” can be:
  - Local browser cache
  - Client-side proxy
  - CDNs “cloud” proxies
Browser caching

Checks to see if the browser has a known fresh copy, and if not if the server has updated the page

- Uses a collection of headers for the checks
Consistency and Caching Directives

• Key issue is knowing when cached data is fresh/stale
  – Otherwise many connections or the risk of staleness

• Browsers typically use
  – Heuristics that check freshness once a “session” with GET
    If-Modified-Since and then assume it’s fresh the rest of the time
  – Use expiration on the header
  – Indicate whether some page is cacheable or not
Proxy caches

Proxy caches at the organization level
- Share proxy caches between many users
- Your browser has built-in support for this
More on proxy caches

• Does not work for encrypted traffic, dynamic web pages etc
• Provides a 2nd layer of caching over browser caches
• Allows companies to filter certain kinds of traffic
Content Distribution Networks (CDNs)

- What if we extend the idea of caching to the scale of the web?

- Web pages can be cached in several locations around the world
  - The web page will be returned from the "nearest" location
  - The "nearest" location may not necessarily be geographical nearest

- Rather than the client caching, it is the server that places cached copies everywhere
CDNs

CDNs caches at the scale of the web
Using the tree structure allows scaling

- Can distribute to many CDN nodes without requiring too much bandwidth

- The original server is not overloaded even if the number of clients increase

- Can increase the number of CDN nodes as needed
How to organize clients to use the correct CDN node?

- Replace proxy caches with CDN nodes
- Use mirroring
- DNS indirection
Implementing CDNs using DNs indirection

Directing clients to nearby CDN nodes with DNS:
- Client query returns local CDN node as response
- Local CDN node caches content for nearby clients and reduces load on the origin server
Example of a CDN: Akamai

- Founded in 1998, used DNS for indirection
- Incentive model: Businesses pay Akamai to deliver content.
- Several companies use Akamai: Facebook, Twitter, Hulu, Yahoo, Netflix, and many many more
- The Akamai model can also help against flash crowds
Recap

- Why is caching useful?

- Why does proxy caching work so well for Internet traffic?

- How is a browser cache different from a proxy cache different from a CDN?
P2P content distribution

- No need for large infrastructure support

- Popular to share music, videos, etc

- How it works
  - A publisher publishes a song and say I download the song
  - There are now two copies of the song, so you can either download the song from me or the original publisher
  - The next person can potentially download the song from three people: the publisher, me and you
  - And so on..
Peer-to-Peer Systems

- Quickly grown in popularity:
  - Dozens or hundreds of file sharing applications
  - In 2004:
    - 35 million adults used P2P networks – 29% of all Internet users in USA
    - Upset the music industry, drew college students, web developers, recording artists and universities into court
P2P content distribution is self scaling.

Instead of a download rate of $1/n$, downloads can scale with number of participants $n$.
BitTorrent

• 2002: B. Cohen debuted BitTorrent

• Key motivation: popular content
  • Popularity exhibits temporal locality (Flash Crowds)
  • E.g., Slashdot effect, release of movie/game

• Key reason for success: preventing free-loading

• Significant fraction of Internet traffic today
Main features of BitTorrent

• How does a peer find other peers that have content it wants to download?

• How is content downloaded?

• What is the incentive mechanism?
How does a peer find content?

• A user contacts a centralized tracker about content it needs

• The tracker provides the user access to the torrent file

• The torrent file is a small file that contains information about all servers that have the content

• The user downloads the torrent file to bootstrap the download
How is content downloaded?

- Large file divided into smaller pieces
  - Fixed-sized chunks
  - Typical chunk size of 16KB - 256 KB
- Allows simultaneous transfers between peers
  - Downloading chunks from different neighbors
  - Uploading chunks to other neighbors
BitTorrent: Simultaneous uploads and downloads
BitTorrent incentive

- Tit-for-Tat: A peer only uploads content to another peer if it gets good download performance from the peer.
- Peers randomly upload to other peers to see if there is a good match.
- Randomly uploading is essential. Why?
Recap

• How does P2P content distribution work?

• Why is BitTorrent called self-scaling?

• What is the incentive scheme used by BitTorrent?
Structured P2P

- How does a user know who has content?

- Bittorrent has a central server that has an index of servers and contents – too central

- Each server can maintain its own index with all contents it has – too distributed

- Distributed Hash Tables is a data structure that distributes the index, but in a way that is easy to search for content
What are DHTs?

• What is a hash table?

• Database has (key, value) pairs;
  – key: content ID; value: IP address

• Peers query DB with key
  – DB returns values that match the key
Properties of a DHT

How to assign values to keys such that

- Decentralized routing
- Small diameter
- Stable against nodes joining and leaving the network
DHT Identifiers

• Assign integer identifier to each peer in range \([0,2^n-1]\).

• To get integer keys and identifiers, hash name.
  – eg, key = h(“Led Zeppelin IV”)
  – eg, peer identifier = h(“128.119.245.24”)
  – This is why they call it a distributed “hash” table
How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1
Circular DHT

- Each peer *only* aware of immediate successor and predecessor.
- “Overlay network”
Shortcuts using finger table

- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Possible to design shortcuts so $O(\log N)$ neighbors, $O(\log N)$ messages in query

Who’s resp for key 1110?
Used of DHTs

- BTDigg: BitTorrent DHT search engine
- FAROO: Distributed web search engine