CSCI-1680
CDN & P2P

Chen Avin

Based partly on lecture notes by Scott Shenker and John Jannotti and Rodrigo Fonseca
Last time

- DNS & DHT
- Today: P2P & CND
  - P2P Benefits
  - Bit Torrent & Skype
  - Caching & Content Distribution Networks
Content distribution networks

• **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

• **option 1:** single, large “mega-server”
  – single point of failure
  – point of network congestion
  – long path to distant clients
  – multiple copies of video sent over outgoing link

....quite simply: this solution *doesn’t scale*
Content distribution networks

• **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

• **option 2:** store/serve multiple copies of videos at multiple geographically distributed sites (**CDN**)
  – **enter deep:** push CDN servers deep into many access networks
    • close to users
    • used by Akamai, 1700 locations
  – **bring home:** smaller number (10’s) of larger clusters in POPs near (but not within) access networks
    • used by Limelight
CDN: “simple” content access scenario

Bob (client) requests video http://netcinema.com/6Y7B23V
- video stored in CDN at http://KingCDN.com/NetC6y&B23V

2. Resolve http://netcinema.com/6Y7B23V via Bob’s local DNS
3. netcinema’s authoritative DNS returns URL http://KingCDN.com/NetC6y&B23V
4&5. Resolve http://KingCDN.com/NetC6y&B23V via KingCDN’s authoritative DNS, which returns IP address of KingCDN server with video
6. Request video from KINGCDN server, streamed via HTTP
CDN cluster selection strategy

• **challenge:** how does CDN DNS select “good” CDN node to stream to client
  – pick CDN node geographically closest to client
  – pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)
  – IP anycast

• **alternative:** let **client** decide - give client a list of several CDN servers
  – client pings servers, picks “best”
  – Netflix approach
How Akamai works

- Akamai has cache servers deployed close to clients
  - Co-located with many ISPs
- Challenge: make same domain name resolve to a proxy close to the client
- Lots of DNS tricks. BestBuy is a customer
  - Delegate name resolution to Akamai (via a CNAME)
- From Brown:
  
  dig www.bestbuy.com
  ;; ANSWER SECTION:
  www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
  a1105.b.akamai.net. 20 IN A 198.7.236.240
  - Ping time: 2.53ms
- From Berkeley, CA:
  a1105.b.akamai.net. 20 IN A 198.189.255.200
  a1105.b.akamai.net. 20 IN A 198.189.255.207
  - Ping time: 3.20ms
DNS Resolution

dig www.bestbuy.com

;; ANSWER SECTION:
www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
a1105.b.akamai.net. 20 IN A 198.7.236.240

;; AUTHORITY SECTION:
b.akamai.net. 1101 IN NS n1b.akamai.net.
b.akamai.net. 1101 IN NS n0b.akamai.net.

;; ADDITIONAL SECTION:
n0b.akamai.net. 1267 IN A 24.143.194.45
n1b.akamai.net. 2196 IN A 198.7.236.236

- **n1b.akamai.net** finds an edge server close to the client’s local resolver
- Uses knowledge of network: BGP feeds, traceroutes. *Their secret sauce*…
What about the content?

- Say you are Akamai
  - Clusters of machines close to clients
  - Caching data from many customers
  - Proxy fetches data from *origin* server first time it sees a URL

- Choose cluster based on client network location

- How to choose server within a cluster?

- If you choose based on client
  - Low hit rate: N servers in cluster means N cache misses per URL
Consistent Hashing  [Karger et al., 99]

- URLs and Caches are mapped to points on a circle using a hash function
- A URL is assigned to the closest cache clockwise
- Minimizes data movement on change!
  - When a cache is added, only the items in the preceding segment are moved
  - When a cache is removed, only the next cache is affected
Consistent Hashing \cite{Karger et al., 99}

- Minimizes data movement
  - If 100 caches, add/remove a proxy invalidates \(\sim 1\%\) of objects
  - When proxy overloaded, spill to successor

- Can also handle servers with different capacities.

  How?
  - Give bigger proxies more random points on the ring

<table>
<thead>
<tr>
<th>Object</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
</tr>
</tbody>
</table>
Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

**examples:**
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)
Peer-to-Peer Systems

- **How did it start?**
  - A killer application: file distribution
  - Free music over the Internet! (*not exactly legal…*)

- **Key idea: share storage, content, and bandwidth of individual users**
  - Lots of them

- **Big challenge: coordinate all of these users**
  - In a scalable way (not NxN!)
  - With changing population (aka *churn*)
  - With no central administration
  - With no trust
  - With large heterogeneity (content, storage, bandwidth,...)
3 Key Requirements

• P2P Systems do three things:
  • Help users determine what they want
    – Some form of search
    – P2P version of Google
  • Locate that content
    – Which node(s) hold the content?
    – P2P version of DNS (map name to location)
  • Download the content
    – Should be efficient
    – P2P form of Akamai
**Question:** how much time to distribute file (size $F$) from one server to $N$ peers?

- peer upload/download capacity is limited resource

---

**Diagram:**

- $u_s$: server upload capacity
- $d_i$: peer $i$ download capacity
- $u_i$: peer $i$ upload capacity
- $u_N$: server upload capacity
- $u_1$, $u_2$, $d_1$, $d_2$: peer upload/download capacities
- Network (with abundant bandwidth)
File distribution time: client-server

- **server transmission**: must sequentially send (upload) $N$ file copies:
  - time to send one copy: $F/u_s$
  - time to send $N$ copies: $NF/u_s$
- **client**: each client must download file copy
  - $d_{\text{min}} = \text{min client download rate}$
  - min client download time: $F/d_{\text{min}}$

\[
D_{c-s} \geq \max\{NF/u_s, F/d_{\text{min}}\}
\]

increases linearly in $N$
File distribution time: P2P

- **server transmission**: must upload at least one copy
  - time to send one copy: \( F/u_s \)
- **client**: each client must download file copy
  - min client download time: \( F/d_{\text{min}} \)
- **clients**: as aggregate must download \( NF \) bits
  - max upload rate (limiting max download rate) is \( u_s + \Sigma u_i \)

\[
D_{\text{P2P}} \geq \max\{F/u_s,F/d_{\text{min}},NF/(u_s + \Sigma u_i)\}
\]

Increases linearly in \( N \) …

… but so does this, as each peer brings service capacity
Client-server vs. P2P: example

client upload rate = $u$, $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$
Napster (1999)
Napster

• Search & Location: central server
• Download: contact a peer, transfer directly
• Advantages:
  – Simple, advanced search possible
• Disadvantages:
  – Single point of failure (technical and … legal!)
  – The latter is what got Napster killed
Gnutella: Flooding on Overlays (2000)

- Search & Location: flooding (with TTL)
- Download: direct

An “unstructured” overlay network
Gnutella: Flooding on Overlays

Flooding
Gnutella: Flooding on Overlays

Flooding
Gnutella: Flooding on Overlays

```
xyz.mp3
```
KaZaA: Flooding w/ Super Peers (2001)

- Well connected nodes can be installed (KaZaA) or self-promoted (Gnutella)
Voice-over-IP: Skype

- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- **P2P components:**
  - clients: skype peers connect directly to each other for VoIP call
  - super nodes (SN): skype peers with special functions
  - overlay network: among SNs to locate SCs
  - login server

![Skype architecture diagram]

- Skype clients (SC)
- supernode (SN)
- supernode overlay network
- Skype login server
P2P voice-over-IP: skype

**skype client operation:**

1. joins skype network by contacting SN (IP address cached) using TCP
2. logs-in (username, password) to centralized skype login server
3. obtains IP address for callee from SN, SN overlay
   - or client buddy list
4. initiate call directly to callee
Skype: peers as relays

- **problem**: both Alice, Bob are behind “NATs”
  - NAT prevents outside peer from initiating connection to insider peer
  - inside peer *can* initiate connection to outside

  - relay solution: *Alice, Bob maintain open connection to their SNs*
    - Alice signals her SN to connect to Bob
    - Alice’s SN connects to Bob’s SN
    - Bob’s SN connects to Bob over open connection Bob initially initiated to his SN
Lessons and Limitations

• **Client-server performs well**
  – But not always feasible

• **Things that flood-based systems do well**
  – Organic scaling
  – Decentralization of visibility and liability
  – Finding popular stuff
  – Fancy *local* queries

• **Things that flood-based systems do poorly**
  – Finding unpopular stuff
  – Fancy *distributed* queries
  – Vulnerabilities: data poisoning, tracking, etc.
  – Guarantees about anything (answer quality, privacy, etc.)
P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

**tracker**: tracks peers participating in torrent

**torrent**: group of peers exchanging chunks of a file

Alice arrives …
… obtains list of peers from tracker
… and begins exchanging file chunks with peers in torrent
P2P file distribution: BitTorrent

- **peer joining torrent:**
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)

- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- **churn:** peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent
BitTorrent: requesting, sending file chunks

**requesting chunks:**
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

**sending chunks: tit-for-tat**
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - “optimistically unchoke” this peer
  - newly chosen peer may join top 4
BitTorrent: tit-for-tat

1. Alice “optimistically unchokes” Bob
2. Alice becomes one of Bob’s top-four providers; Bob reciprocates
3. Bob becomes one of Alice’s top-four providers

higher upload rate: find better trading partners, get file faster!