CSCE 463/612 Networks and Distributed Processing Fall 2024

Application Layer V

Dmitri Loguinov
Texas A&M University

September 24, 2024

Homework #2

TX ID	flags
nQuestions	nAnswers
nAuthority	nAdditional

questions (variable size)
answers (variable size)
authority (variable size)
additional (variable size)

- Unlike HTTP, all fields are binary
  - Make sure to refresh pointer usage
- Question format:

str_1 size	str_1	…	str_n size	str_n	0	Query type	Query class
1 byte			1 byte			2 bytes	2 bytes

question

- Create structs for fixed headers
  - Fill in the values (flags: DNS_QUERY and DNS_RD, nQuestions = 1)
  - Allocate memory for the packet
  - Write question into buffer

```
class QueryHeader {
    u_short type;
    u_short class;
};
```

```
class FixedDNSheader {
    u_short ID;
    u_short flags;
    u_short questions;
    ...
};
```

Homework #2

- High-level operation for DNS questions:

```
char packet [MAX_DNS_LEN];                    // 512 bytes is max
char host[] = “www.google.com”;
int pkt_size = strlen(host) + 2 + sizeof(FixedDNSheader) + sizeof(QueryHeader);
// fixed field initialization
FixedDNSheader *dh = (FixedDNSheader *) packet;
QueryHeader *qh = (QueryHeader*) (packet + pkt_size - sizeof(QueryHeader));
dh->ID = ...
dh->flags = ...
...
qh->type = ...
qh->class = ...

// fill in the question
MakeDNSquestion (dh + 1, host);
// transmit to Winsock
sendto (sock, packet, ...);
```

- If packet is incorrectly formatted, you will usually get no response; use Wireshark to check outgoing packets

Homework #2

```
class DNSanswerHdr {
          u_short type;
          u_short class;
          u_int ttl;
          u_short len;
};
```

- Formation of questions:

```
makeDNSquestion (char* buf, char *host) {
     while(words left to copy){
          buf[i++] = size_of_next_word;
          memcpy (buf+i, next_word, size_of_next_word);
          i += size_of_next_word;
     }
     buf[i] = 0;     // last word NULL-terminated
}
```

- All answers start with an RR name, followed by a fixed *DNS answer header*, followed by the answer itself
  - Uncompressed answer (not common)

```
0x3 “irl” 0x2 “cs” 0x4 “tamu” 0x3 “edu” 0x00
<DNSanswerHdr> <ANSWER>
```

  - Compressed (2 upper bits 11, next 14 bits jump offset)

```
0xC0 0x0C <DNSanswerHdr> <ANSWER>
```

- For type-A questions, the answer is a 4-byte IP

Homework #2

- To check the header
  - Hex printout on screen
  - Wireshark

```
class DNSanswerHdr {
          u_short type;
          u_short class;
          u_int ttl;
          u_short len;
};
```

- What is `sizeof(DNSanswerHdr)`?
  - The actual size is 10 bytes, but the compiler will align/pad it to 4-byte boundary (so 12)
- Remember to change struct packing of all classes that define binary headers to 1 byte

```
#pragma pack(push,1)
// define headers here
#pragma pack(pop)
```

- Caveats (must be properly handled):
  - Exceeding array boundaries on jumps
  - Infinite looping on compressed answers

Homework #2

- How to check if compressed and read 14-bit offset?
  - Suppose array `char *ans` contains the reply packet
  - The answer begins within this array at position `curPos`

1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x

14 bits

```
char *ans;  // points to reply buffer
if (ans [curPos] >= 0xC0)
    // compressed; so jump
else
    // uncompressed, read next word
```

```
char *ans;  // points to reply buffer
if ( (ans [curPos] >> 6) == 3)
    // compressed; so jump
else
    // uncompressed, read next word
```

```
// computing the jump offset
int off = ( (ans[curPos] & 0x3F) << 8) + ans[curPos + 1];
```

- The first two checks will generally fail
  - Use only unsigned chars when reading buffer!

Homework #2

- Note that jumps may appear mid-answer

  0x3 “irl” 0xC0 0x22 <DNSanswerHdr> <ANSWER>
- Jumps may be nested, but must eventually end with a 0-length word
  - Need to remember the position following the very first jump so that you can come back to read DNSanswerHdr
- Replies may be malicious or malformatted
  - Homework must avoid crashing
- If AAAA (IPv6) answers are present, skip
  - Use `DNSanswerHdr::len` to jump over unknown types
- Caution with TAMU VPN
  - Malformed packets are filtered out

Chapter 2: Roadmap

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

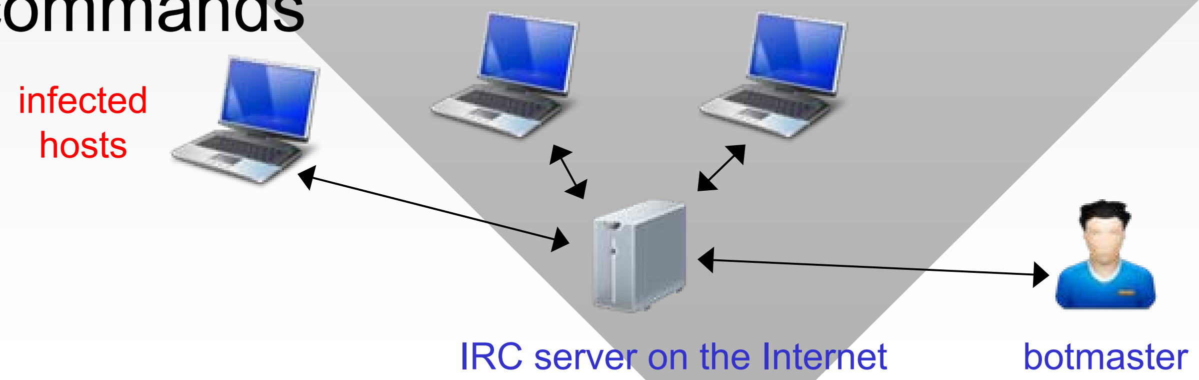
- SMTP, POP3, IMAP

2.5 DNS (extras)

2.6 P2P file sharing

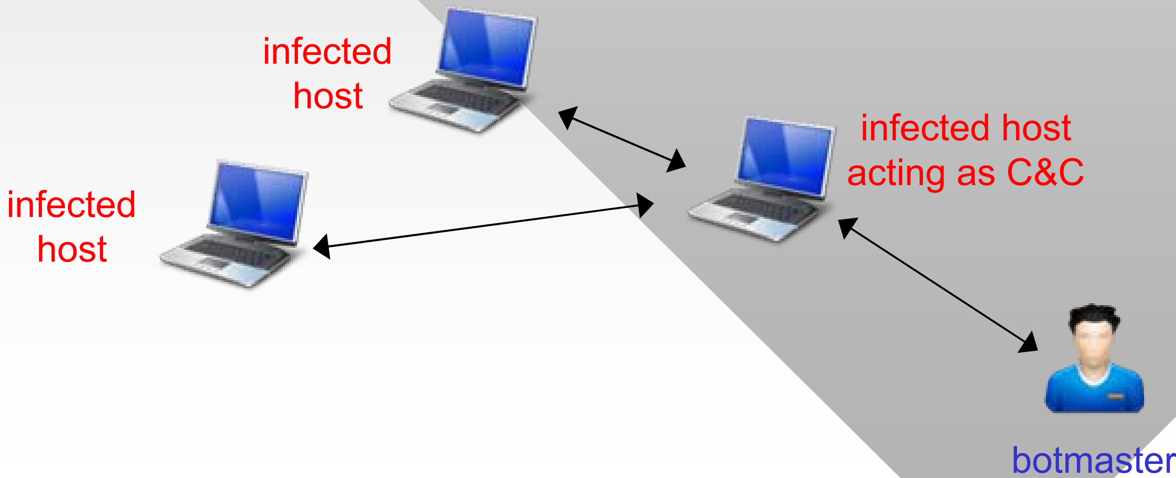
Domain Flux

- Viruses, trojan horses, rootkits, and various malware affect millions of computers today
- Years ago, viruses mostly performed pranks or corrupted data, but this has changed
  - Modern attacks are often driven by financial gains
- Infected hosts are organized into botnets
  - Large collection of computers under control of a botmaster
- Early botnets used IRC (Internet Relay Chat) to send and receive commands



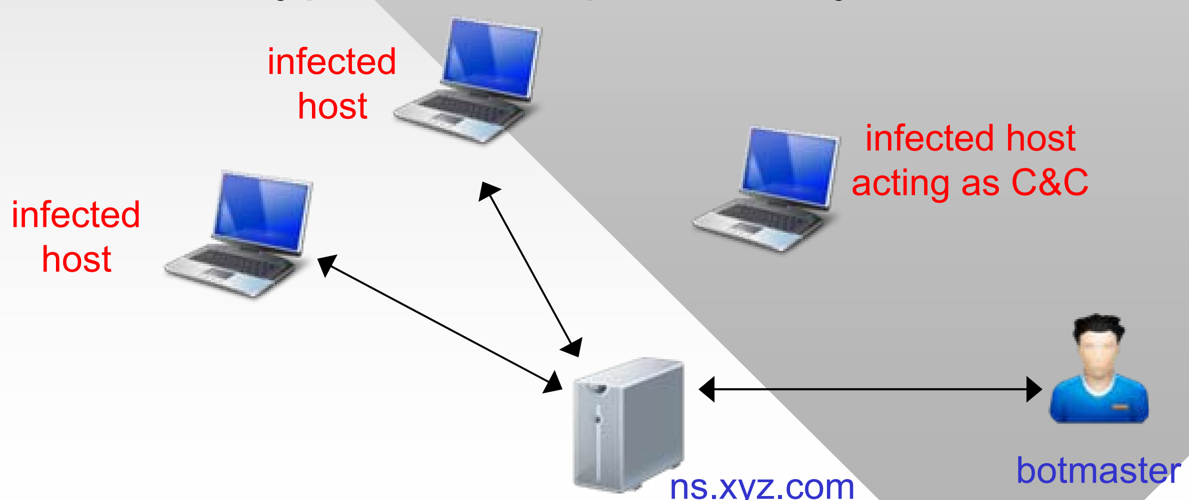
Domain Flux 2

- Eventually, ISPs started blocking IRC traffic
  - Also, IRC servers were easy targets for shutdown and filtering (e.g., detection of encrypted commands and botnet channels)
- New generation of botnets uses dynamically changing rendezvous points called C&C (command & control)
  - Stealthy because C&C's IP can rapidly change over time
  - Main problem: how does the botnet find the current C&C?



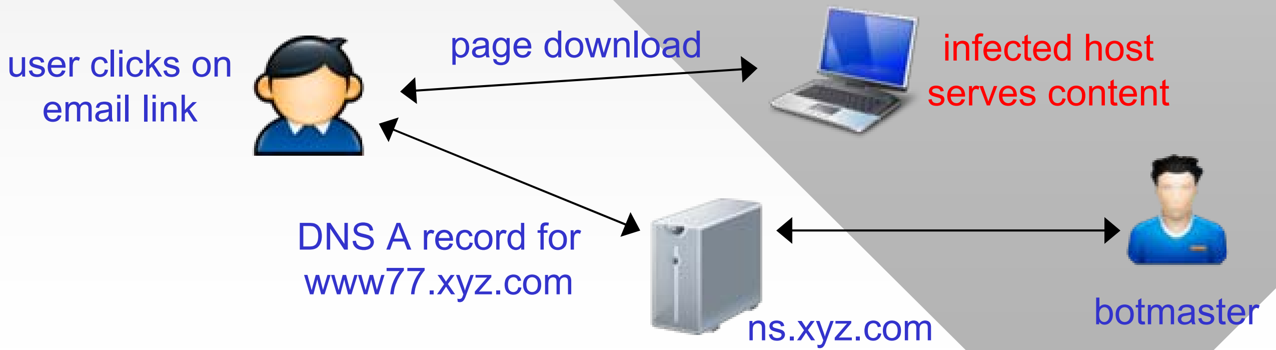
Domain Flux 3

- Fast flux is a method for discovering the IP address of C&C and other resources the botnet may need
  - Botmaster registers a domain (say xyz.com) and controls the DNS server ns.xyz.com
- Botnet contacts nameserver ns.xyz.com and obtains the current IP of the C&C (or multiple ones)
  - Performs a type-A lookup inside xyz.com



Domain Flux 4

- Main defense against botnet traffic is blocking communication with the C&C
  - Fast Flux makes it harder since the C&C changes over time and is load-balanced across several hosts
  - When C&C is blocked, botnet learns other locations quickly
- Fast flux can also be used to serve phishing content
  - Suppose email arrives to user with a link to www77.xyz.com
  - Botnet uses DNS to serve this request from a variety of compromised hosts



Domain Flux 5

Nowadays, TLD servers auto-detect fastflux and block suspected domains in conjunction with the registrar

- Benefits to serving HTTP content using fast flux
  - Difficult to trace IPs hosting content or block malicious URLs
  - Botnet is failure resilient -- if hosts are cleaned or go offline, there is automatic fail-over to other live hosts
  - Cheap in terms of bandwidth, simple to implement
- However, there is a problem
  - Suppose ISP, email filter (e.g., SpamAssasin), or the registrar block all references to xyz.com?
  - If xyz.com is taken down, the botnet freezes
- Domain flux aims to solve this issue
  - If current domain is blocked, botnet generates replacement domain names and tries to resolve them to find the C&C
  - More difficult to trace and block

Domain Flux 6

- Toy example:
  - Suppose botnet computers generate names using this sequence: 1.com, 2.com, 3.com, 5.com, 8.com, 13.com, etc.
  - Current domain name stays in effect until it is blocked
  - Initially, botmaster registers 1.com and 34.com
  - When 1.com gets blocked, the botnet automatically switches to 34.com, while botmaster registers 144.com, and so on
- In reality, the botnet goes through thousands of failed lookup attempts until it finds an active domain
  - Can be detected from a huge number of failed DNS queries
- Domains may be too random to be human-produced
  - If so, machine-learning algorithms can be used to detect infected hosts that are attempting domain flux

Domain Flux 7

- In some cases, reverse engineering the random generator allows one to predict future domain names
  - By registering these domains, botnets can be hijacked
  - Researchers have shown this is possible in B. Stone-Gross et al., “Your botnet is my botnet: Analysis of a botnet takeover,” ACM CCS, 2009.
- How large are botnets? Some examples:
  - BredoLab (2009): 30M hosts, 3.6B emails/day
  - Conficker (2008): 10.5M hosts, 10B emails/day
  - Cutwail (2007): 1.5M hosts, 74B emails/day
  - Torpig (paper above): 180K hosts (theft of 500K bank accounts, credit cards)
  - Avalanche (2008-2016): phishing botnet w/500K hosts

Chapter 2: Roadmap

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

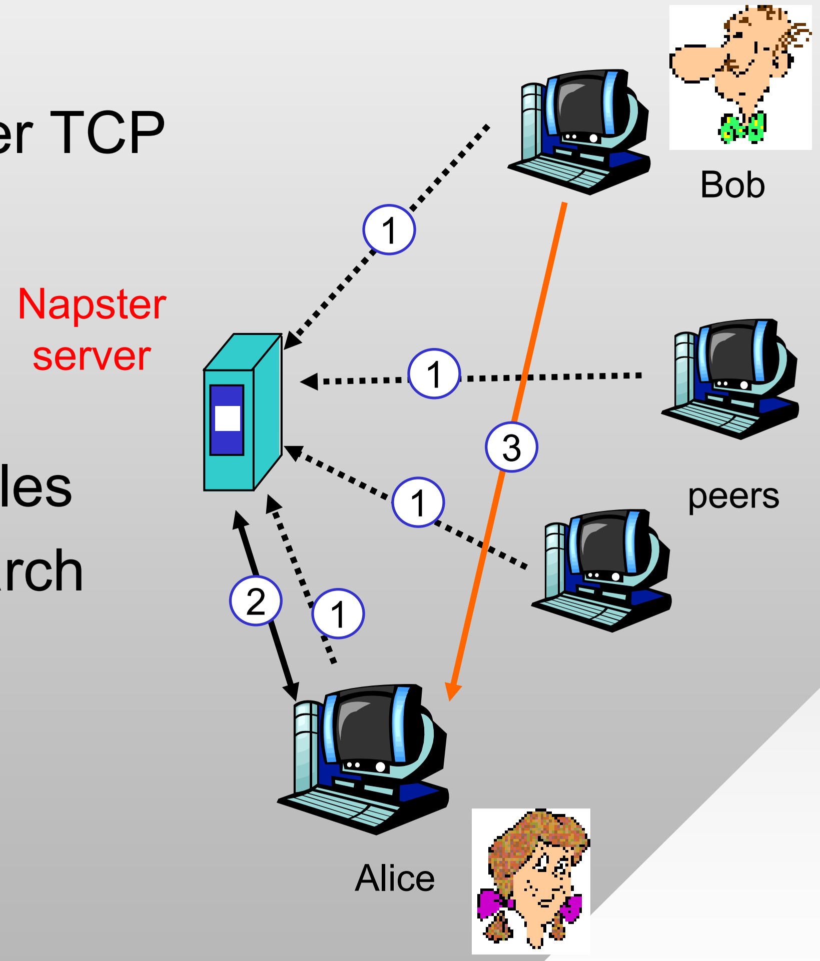
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P file sharing

Hybrid P2P

- Napster (1999)
  - Application-layer protocol over TCP
  - Centralized directory server
- Sequence of steps
  - Connect to server, login
  - Upload your IP/port + list of files
  - Give server keywords for search
  - Select "best" answer (ping)
  - Download from peer
- Single point of failure
- Performance bottleneck
- Target for litigation due to copyright infringement

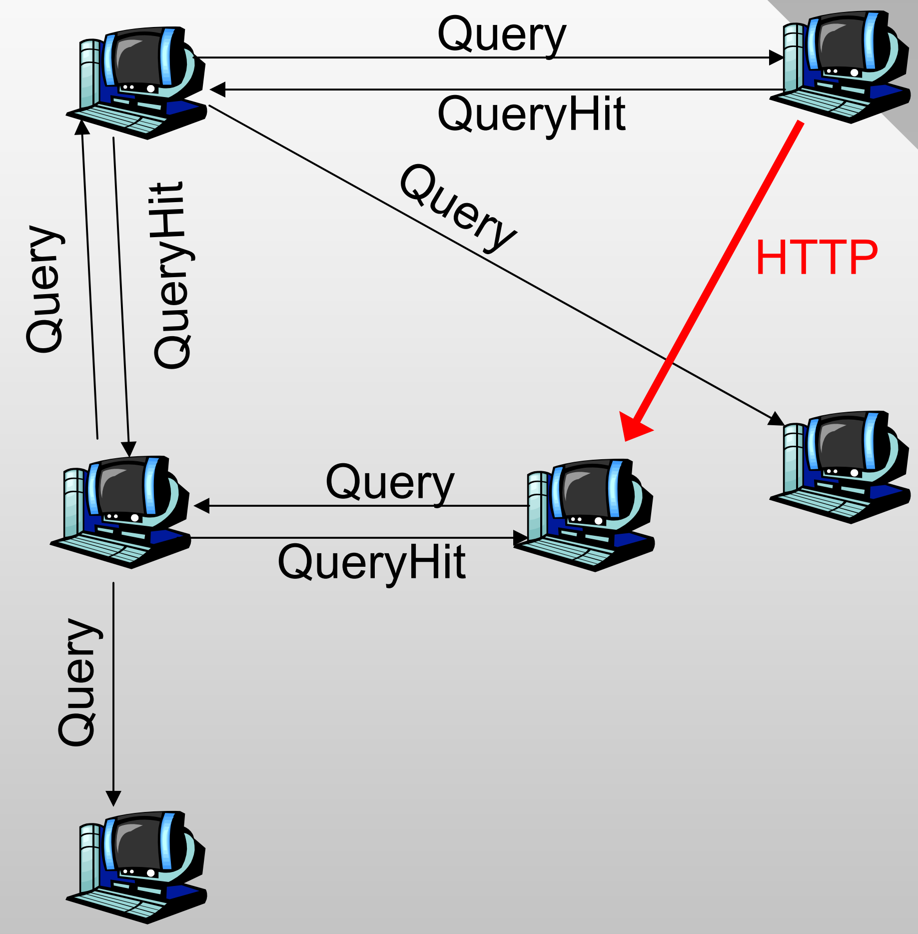


Decentralized P2P

- Napster folded in 2002
  - Other P2P systems took over (Gnutella, KaZaA, BitTorrent, eDonkey)
- Gnutella/0.4 (2001)
  - Public-domain protocol
  - Fully distributed design
- Many Gnutella clients implementing protocol
  - Limewire, Morpheus, BearShare

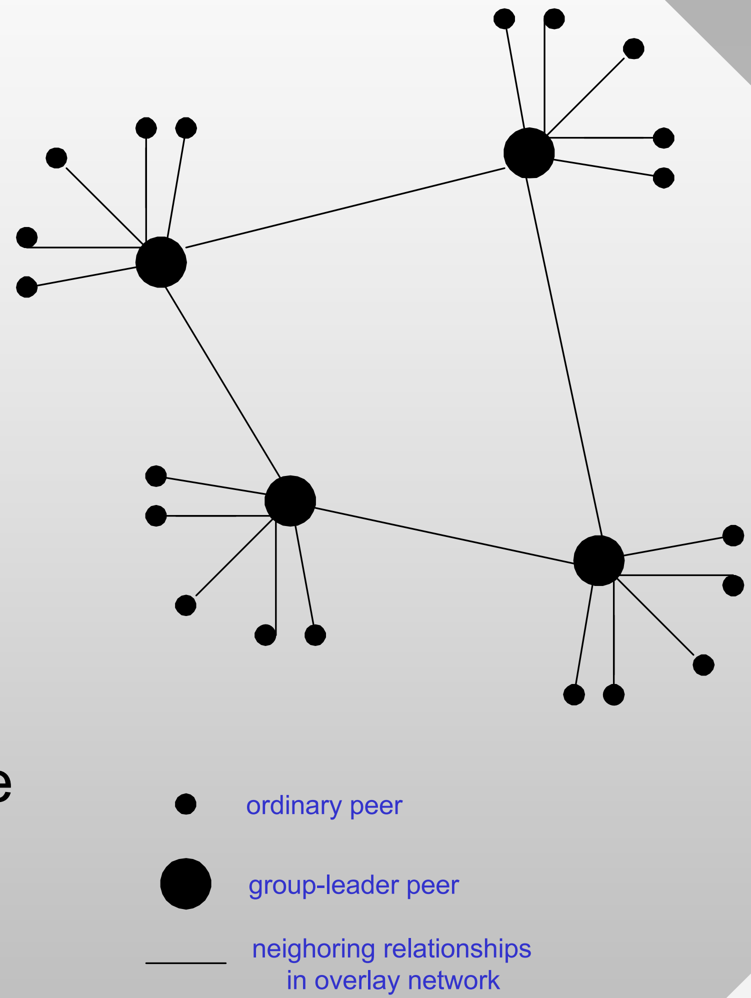
- How to find content?
- Idea: construct a graph
  - Edge between peer X and Y if there's a TCP connection between them
- All active peers and edges are called an overlay network
  - Peer typically connected to < 30 neighbors
- Search proceeds by flooding up to some depth
  - Limited-scope flooding

Decentralized P2P



- Queries are P2P
  - Inefficient due to huge volumes of traffic
  - Average degree k, depth of flood d, overhead $(k-1)^d$
- Downloads are P2P from a single user
  - Unreliable (peer departure or failure kills transfer)
  - Inefficient (asymmetry of upstream/downstream bandwidth)
- Join protocol (bootstrapping)
  - Find an entry peer X, flood its neighbors to obtain more candidates, establish connections to those who accept

Hierarchical P2P



- Gnutella/0.4 scaled to about 25K users and then choked
- Alternative construction proposed by KaZaA (2002)
  - Peer is either a group leader (supernode) or assigned to one
- Group leader tracks the content of all its children, acting like a mini-Napster
  - Peers query their group leaders, which flood the supernode graph until some number of matches found
  - Query-hits not routed, but sent directly to original supernode

Hierarchical P2P

- With 150 neighbors, this architecture is 150x more efficient than Gnutella/0.4 in message overhead
  - With 389M downloads as of 2008, KaZaA was more popular than Napster ever was, accounting for 50% of ISP bandwidth in some regions and running 3M concurrent users
- Gnutella/0.6 soon adopted the same structure
  - Scaled to 6.5M online users, 60M unique visitors per week
- Additional features
  - Hashed file contents to identify exact version of files
  - Upload and request queuing at each user, rate-limiting
  - Parallel downloads from multiple peers
  - Support for crawl requests that reveal neighbors

Other P2P

- Terminology: user holding a complete file is a seed
  - Traditional systems download only from seeds
  - Seed departs, transfer fails
- Idea: let non-seeds grab chunks from each other
  - Peers organize into a group (torrent) based on the file they're downloading
- Traditional systems download files sequentially
  - Starvation for final blocks
- Idea: maximize availability
  - Participants forced to serve chunks they have to others
  - Rarest chunk in torrent is always replicated first
- Known as BitTorrent (2001)
  - Protocol with many implementations
  - Requires trackers to keep torrent membership
  - Had more concurrent users that YouTube and Facebook combined
- Built-in incentives to share
  - Rate-limiting (choking) based on upload activity

Other P2P

- Tor (Onion Router)
  - Anonymity network of peers
- Each packet sent through a random chain of P2P nodes
  - Final user relays packet towards destination
  - Return packets processed similarly along reverse path
- Tor can be run thru an API
  - Extremely slow
  - Many exit points are known and blocked by Google
- Roughly 36M users

- Freenet
  - Anonymous information exchange, hiding identities of communicating parties
- Skype chat
  - Video streaming services either directly between users or relayed through non-firewalled peers
- Distributed Hash Tables
  - General class of P2P systems that map information into high-dimensional search space with guaranteed log(N) bounds on delay to find content