Network Applications:
Smart Switch (Load Balancer)
Application Overlays (P2P)

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

10/12/2017
Outline

- Admin and recap
- Multi-servers
- Application overlays (peer-to-peer networks)
Assignment three status and questions.
SocketAddress address  
  = new InetSocketAddress(args[0], port);
AsynchronousSocketChannel client  
  = AsynchronousSocketChannel.open();
Future<Void> connected  
  = client.connect(address);
ByteBuffer buffer = ByteBuffer.allocate(100);

// wait for the connection to finish
connected.get();

// read from the connection
Future<Integer> future = client.read(buffer);

// do other things...

// wait for the read to finish...
future.get();

// flip and drain the buffer
buffer.flip();
WritableByteChannel out  
  = Channels.newChannel(System.out);
try {
    out.write(buffer);
} catch (IOException ex) {
    System.err.println(ex);
}

@Override
public void completed(Integer result, ByteBuffer buffer) 
{
    buffer.flip();
    WritableByteChannel out  
        = Channels.newChannel(System.out);
    try {
        out.write(buffer);
    } catch (IOException ex) {
        System.err.println(ex);
    }
}

@Override
public void failed(Throwable ex, ByteBuffer attachment) {
    System.err.println(ex.getMessage());
}

ByteBuffer buffer = ByteBuffer.allocate(100);
CompletionHandler<Integer, ByteBuffer> handler = new LineHandler();
channel.read(buffer, buffer, handler);
Recap: Operational Analysis— Fundamental Laws

- **Utilization law:** \( U_i = X_i S_i \)
- **Forced flow law:** \( X_i = V_i X \)
- **Bottleneck device:** Largest \( D_i = V_i S_i \)
- **Little’s Law:** \( Q_i = X_i R_i \)
- **Bottleneck bound of interactive response (for the given closed model):**
  \[
  X(N) \leq \min \left\{ \frac{1}{D_{\max}}, \frac{N}{D+Z} \right\}
  \]
  \[
  R(N) \geq \max \{D, ND_{\max} - Z\}
  \]
Summary: High-Perf. Network Server

- Avoid blocking (so that we can reach bottleneck throughput)
  - Introduce threads

- Limit unlimited thread overhead
  - Thread pool, async io

- Shared variables
  - Synchronization (lock, synchronized)

- Avoid busy-wait
  - Wait/notify; FSM; asynchronous channel/Future/Handler

- Extensibility/robustness
  - Language support/Design for interfaces

- System modeling and measurements
  - Queueing analysis, operational analysis
Recap: Multiple Servers

- Why
  - Scalability
    - Scaling beyond single server throughput
      - FB 12M HTTP/sec in 2013
    - Scaling beyond single geo location latency
  - Redundancy and fault tolerance
  - System/software architecture

- Challenges when designing multiple servers
Recap: Direction Mechanisms

- Cluster1 in US East
- Cluster2 in US West
- Cluster2 in Europe
Recap: CNAME based DNS Name

- Typical design
  - Use `cname` to create aliases, e.g.,
  - `cname`: e2466.dscg.akamaiedge.net
    - why two levels in the name?
Two-Level Direction

- high-level DNS determines proximity, directs to low-level DNS;
  Input: dscg.akamaiedge.net & and client IP,
  Output: region (low-level) DNS

- low-level DNS: who manages a close-by cluster of servers with different IP addresses
  Input: e2466.dscg.akamaiedge.net & and client IP
  Output: specific servers
Akamai Load Direction

If the directed edge server does not have requested content, it goes back to the original server (source).
Two-Level DNS Mapping Alg

- **High-level**
  - Typically geo-location matching from client to region

- **Low-level**
  - Typically secret, e.g., details of Akamai algorithms are proprietary
  - Typical goal: load balancing among servers
Akamai Local DNS LB Alg

- A Bin-Packing algorithm (column 12 of Akamai Patent) every T second
  - Compute the load to each publisher k (called serial number)
  - Sort the publishers from increasing load
  - For each publisher, associate a list of random servers generated by a hash function
    - Hash range may be increasing, e.g., first hash [0,1], second [0, 3]
  - Assign the publisher to the first server that does not overload
Experimental Study of Akamai Load Balancing

Methodology

- 2-months long measurement
- 140 PlanetLab nodes (clients)
  - 50 US and Canada, 35 Europe, 18 Asia, 8 South America, the rest randomly scattered
- Every 20 sec, each client queries an appropriate CNAME for Yahoo, CNN, Fox News, NY Times, etc.

See http://www.aqualab.cs.northwestern.edu/publications/Ajsu06DBA.pdf
Server Pool: to Yahoo

Client 1: Berkeley

Client 2: Purdue

Target: a943.x.a.yimg.com (Yahoo)
Server Diversity for Yahoo

Majority of PL nodes see between 10 and 50 Akamai edge-servers.

Nodes far away from Akamai hot-spots.
Server Pool: Multiple Akamai Hosted Sites

![Chart showing the number of Akamai Web Replicas for different clients.](image-url)
Load Balancing Dynamics

CDF (prob < X)

Inter-redirection time (s)

Berkeley to Yahoo
Korea to Yahoo
Brazil to Yahoo

Berkeley
Brazil
Korea
Redirection Effectiveness: Measurement Methodology

Akamai Low-Level DNS Server

Planet Lab Node

10 Best Akamai Replica Servers
Do redirections reveal network conditions?

- $\text{Rank} = r_1 + r_2 - 1$, where $r_i$ is rank of server $i$
  - 16 means perfect correlation
  - 0 means poor correlation

- Brazil is poor
- MIT and Amsterdam are excellent

Percentage of time Akamai's selection is better or equal to rank

Rank of Akamai's selection
A system named Cartographer (written in Python) processes measurement data and configures the DNS maps of individual DNS servers (open source tinydns)
Discussion

- Advantages of using DNS for using multiple servers (LB)
  - Leveraging existing DNS features (e.g., cname, hierarchy name for natural hierarchical redirection)
  - Leveraging existing DNS deployment/optimization

- Disadvantages of using DNS
  - Distributed caching may lead to slow response
  - Only in the unit of IP addresses
Recap: Direction Mechanisms

Cluster1 in US East

Cluster2 in US West

Cluster2 in Europe

DNS name1

DNS name2

IP1

IP2

IPn

App

Load balancer

proxy

Load balancer

servers
Outline

- Recap
- Multiple network servers
  - Basic issues
  - Load direction
    - DNS (IP level)
    - Load balancer/smart switch (sub-IP level)
Smart Switch: Big Picture
VIP Clustering

Goals
server load balancing
failure detection
access control filtering
priorities/QoS
request locality
transparent caching

What to switch/filter on?
L3 source IP and/or VIP
L4 (TCP) ports etc.
L7 URLs and/or cookies
L7 SSL session IDs
Load Balancer (LB): Basic Structure

Problem of the basic structure?
**Problem**

- Client to server packet has VIP as destination address, but real servers use RIPv
  - if LB just forwards the packet from client to a real server, the real server drops the packet
  - reply from real server to client has real server IP as source -> client will drop the packet

```plaintext
state: listening
address: {RealIP.6789, *:*}  
completed connection queue: C1; C2
sendbuf: 
recvbuf:

state: 
address: {VIP:6789, 198.69.10.10.1500}
sendbuf: 
recvbuf: 
...
...
```

```
state: 
address: {RealIP:6789, 198.69.10.10.1500}
sendbuf: 
recvbuf: 
...
...
```
Solution 1: Network Address Translation (NAT)

- LB does rewriting/translation
- Thus, the LB is similar to a typical NAT gateway with an additional scheduling function
Example Virtual Server via NAT

Table 1: an example of virtual server rules

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Virtual IP Address</th>
<th>Port</th>
<th>Real IP Address</th>
<th>Port</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>80</td>
<td>172.16.0.2</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>172.16.0.3</td>
<td>8000</td>
<td>2</td>
</tr>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>21</td>
<td>172.16.0.3</td>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>
LB/NAT Flow

Request packet

source ip: 206.183.42.40
destination ip: 192.168.0.100

VIP: 192.168.0.100

active load balancer

backup load balancer

primary heartbeat
backup heartbeat

Server 1
RIP: 192.168.0.205

Server n
RIP: 192.168.0.206
LB/NAT Flow
LB/NAT Advantages and Disadvantages

- **Advantages:**
  - Only one public IP address is needed for the load balancer; real servers can use private IP addresses
  - Real servers need no change and are not aware of load balancing

- **Problem**
  - The load balancer must be on the critical path and hence may become the bottleneck due to load to rewrite request and response packets
    - Typically, rewriting responses has more load because there are more response packets
Goal: LB w/ Direct Reply
LB with Direct Reply: Implication

Direct reply  Each real server uses VIP as its IP address

Address conflict: multiple devices w/ the same IP addr.
Why IP Address Matters?

- Each network interface card listens to an assigned MAC address.
- A router is configured with the range of IP addresses connected to each interface (NIC).
- To send to a device with a given IP, the router needs to translate IP to MAC (device) address.
- The translation is done by the Address Resolution Protocol (ARP).
ARP Protocol

- ARP is “plug-and-play”:
  - nodes create their ARP tables without intervention from net administrator

- A broadcast protocol:
  - Router broadcasts query frame, containing queried IP address
    - all machines on LAN receive ARP query
  - Node with queried IP receives ARP frame, replies its MAC address
ARP in Action

- Router broadcasts ARP broadcast query: who has VIP?
- ARP reply from LB: I have VIP; my MAC is $MAC_{LB}$
- Data packet from R to LB: destination MAC = $MAC_{LB}$
ARP and race condition:
- When router R gets a packet with dest. address VIP, it broadcasts an Address Resolution Protocol (ARP) request: who has VIP?
- One of the real servers may reply before load balancer

Solution: configure real servers to not respond to ARP request
The virtual IP address is shared by real servers and the load balancer.

Each real server has a non-ARPing, loopback alias interface configured with the virtual IP address, and the load balancer has an interface configured with the virtual IP address to accept incoming packets.

The workflow of LB/DR is similar to that of LB/NAT:
- the load balancer directly routes a packet to the selected server
  - the load balancer simply changes the MAC address of the data frame to that of the server and retransmits it on the LAN (how to know the real server’s MAC?)
- When the server receives the forwarded packet, the server determines that the packet is for the address on its loopback alias interface, processes the request, and finally returns the result directly to the user
Advantages:
- Real servers send response packets to clients directly, avoiding LB as bottleneck

Disadvantages:
- Servers must have non-arp alias interface
- The load balancer and server must have one of their interfaces in the same LAN segment
- Considered by some as a hack, not a clean architecture
Example Implementation of LB

- An example open source implementation is Linux virtual server (linux-vs.org)
  - Used by
    - www.linux.com
    - sourceforge.net
    - wikipedia.org
    - Many commercial LB servers from F5, Cisco, ...

- More details please read chapter 2 of Load Balancing Servers, Firewalls, and Caches
Problem of the Load Balancer Architecture

One major problem is that the LB becomes a single point of failure (SPOF).
Solutions

- **Redundant load balancers**
  - E.g., two load balancers (a good question to think offline)

- **Fully distributed load balancing**
  - e.g., Microsoft Network Load Balancing (NLB)
**Microsoft NLB**

- No dedicated load balancer
- All servers in the cluster receive all packets
- Key issue: one and only one server processes each packet
  - All servers within the cluster simultaneously run a mapping algorithm to determine which server should handle the packet. Those servers not required to service the packet simply discard it.
  - **Mapping (ranking) algorithm**: computing the “winning” server according to host priorities, multicast or unicast mode, port rules, affinity, load percentage distribution, client IP address, client port number, other internal load information

Discussion

- Compare the design of using Load Balancer vs Microsoft NLB
Recap: Direction Mechanisms

- DNS name1
  - IP1
    - Cluster1 in US East
  - IP2
    - Cluster2 in US West
  - Load balancer
  - proxy
  - servers
- DNS name2
  - IPn
  - Cluster2 in Europe
  - Load balancer
  - App

- Rewrite
- Direct reply
- Fault tolerance
Outline

- Admin and recap
- Multiple servers
  - Overview
  - Basic mechanisms
  - Example: YouTube (offline read)
02/2005: Founded by Chad Hurley, Steve Chen and Jawed Karim, who were all early employees of PayPal.

10/2005: First round of funding ($11.5 M)

03/2006: 30 M video views/day

07/2006: 100 M video views/day

11/2006: acquired by Google

10/2009: Chad Hurley announced in a blog that YouTube serving well over 1 B video views/day (avg = 11,574 video views/sec)
Pre-Google Team Size

- 2 Sysadmins
- 2 Scalability software architects
- 2 feature developers
- 2 network engineers
- 1 DBA
- 0 chefs
while (true) {
    identify_and_fix_bottlenecks();
    drink();
    sleep();
    notice_new_bottleneck();
}
YouTube Major Components

- Web servers
- Video servers
- Thumbnail servers
- Database servers
YouTube: Web Servers

- **Components**
  - Netscaler load balancer; Apache; Python App Servers; Databases

- **Python**
  - Web code (CPU) is not bottleneck
    - JIT to C to speedup
    - C extensions
    - Pre-generate HTML responses
  - Development speed more important
YouTube: Video Popularity

How to design a system to handle highly skewed distribution?

Fig. 8. Recently added YouTube videos rank by popularity

YouTube: Video Server Architecture

- Tiered architecture
  - CDN servers (for popular videos)
    - Low delay; mostly in-memory operation
  - YouTube servers (not popular 1-20 per day)
YouTube Redirection Architecture

YouTube servers

CDN server located in YouTube or Limelight network

(2) HTTP Redirect MSG

(1) HTTP Get MSG

(3) HTTP Get MSG

(4) Flash video stream

[Example of (1)]
Get /get_video?video_id=G_Y3y8escmA
HTTP/1.1

[Example of (2)]
HTTP/1.1 303 See other
Location: http://sjc-v110.sjc.youtube.com
/get_video?video_id=G_Y3y8escmA

Client
YouTube Video Servers

- Each video hosted by a mini-cluster consisting of multiple machines
- Video servers use the lighttpd web server for video transmission:
  - Apache had too much overhead (used in the first few months and then dropped)
  - Async io: uses epoll to wait on multiple fds
  - Switched from single process to multiple process configuration to handle more connections
Thumbnail Servers

- Thumbnails are served by a few machines
- Problems running thumbnail servers
  - A high number of requests/sec as web pages can display 60 thumbnails on page
  - Serving a lot of small objects implies
    - lots of disk seeks and problems with file systems inode and page caches
    - may ran into per directory file limit
  - Solution: storage switched to Google BigTable
Thumbnail Server Software Architecture

- **Design 1: Squid in front of Apache**
  - Problems
    - Squid worked for a while, but as load increased, performance eventually decreased: Went from 300 requests/second to 20
    - Under high loads Apache performed badly, changed to lighttpd

- **Design 2: lighttpd default**: By default, lighttpd uses a single thread
  - Problem: often stalled due to I/O

- **Design 3: switched to multiple processes contending on shared accept**
  - Problems: high contention overhead/individual caches
Thumbnails Server: lighttpd/aio
Example Server Systems

(a) iChat

(b) Google Plus Hangout

(c) Skype multi-party

http://eeweb.poly.edu/faculty/yongliu/docs/imc12.pdf
Scalability of Server-Only Approaches
Outline

- Admin and recap
- Multiple servers
- Application overlays
  - potential
An Upper Bound on Scalability

- Idea: use resources from both clients and the server
- Assume
  - need to achieve same rate to all clients
  - only uplinks can be bottlenecks
- What is an upper bound on scalability?
The Scalability Problem

- **Maximum throughput**

\[ R = \min \{ C_0, \frac{(C_0 + \sum C_i)}{n} \} \]

- The bound is theoretically approachable
Theoretical Capacity: upload is bottleneck

- Assume $c_0 > (c_0 + \Sigma c_i)/n$
- Tree $i$:
  - server $\rightarrow$ client $i$: $c_i/(n-1)$
  - client $i$ $\rightarrow$ other $n-1$ clients
- Tree $0$:
  - server has remaining
  - $c_m = c_0 - (c_1 + c_2 + \ldots + c_n)/(n-1)$
  - send to client $i$: $c_m/n$

$R = \min\{C_0, (C_0 + \Sigma C_i)/n\}$
Clients come and go (churns): maintaining the trees is too expensive
Each client needs N connections
Robustness
- Resistant to churns and failures

Efficiency
- A client has content that others need; otherwise, its upload capacity may not be utilized

Incentive: clients are willing to upload
- Some real systems nearly 50% of all responses are returned by the top 1% of sharing hosts
Discussion: How to handle the issues?

- Robustness
- Efficiency
- Incentive
Example: BitTorrent

- A P2P file sharing protocol
- Created by Bram Cohen in 2004
  - Spec at bep_0003:
BitTorrent: Lookup

HTTP GET MYFILE.torrent

MYFILE.torrent

http://mytracker.com:6969/
S3F5YHG6FEB
FG5467HGF367
F456JI9N5FF4E
...

webserver

user
Metadata (.torrent) File Structure

- Meta info contains information necessary to contact the tracker and describes the files in the torrent
  - URL of tracker
  - file name
  - file length
  - piece length (typically 256KB)
  - SHA-1 hashes of pieces for verification
  - also creation date, comment, creator, ...
Tracker Protocol

- Communicates with clients via HTTP/HTTPS

- **Client GET request**
  - info_hash: uniquely identifies the file
  - peer_id: chosen by and uniquely identifies the client
  - client IP and port
  - numwant: how many peers to return (defaults to 50)
  - stats: e.g., bytes uploaded, downloaded

- **Tracker GET response**
  - interval: how often to contact the tracker
  - list of peers, containing peer id, IP and port
  - stats
Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol

Tracker Protocol
Robustness and efficiency: Piece-based Swarming

- Divide a large file into small blocks and request block-size content from different peers (why?)
  
  Block: unit of download

- If do not finish downloading a block from one peer within timeout (say due to churns), switch to requesting the block from another peer.
Detail: Peer Protocol

(Over TCP)

- Peers exchange bitmap representing content availability
  - bitfield msg during initial connection
  - have msg to notify updates to bitmap
  - to reduce bitmap size, aggregate multiple blocks as a piece

Local Peer

Remote Peer

BitField/have

![Diagram showing bitfield and have messages between local and remote peers.](image)

Piece
256KB

Incomplete Piece
**Peer Request**

- If peer A has a piece that peer B needs, peer B sends interested to A.

- unchoke: indicate that A allows B to request.

- request: B requests a specific block from A.

- piece: specific data

http://www.bittorrent.org/beps/bep_0003.html
Key Design Points

request:
- which data blocks to request?

unchoke:
- which peers to serve?
Request: Block Availability

- Request (local) **rarest first**
  - achieves the fastest replication of rare pieces
  - obtain something of value
Block Availability: Revisions

- When downloading starts (first 4 pieces): choose at random and request them from the peers
  - get pieces as quickly as possible
  - obtain something to offer to others

- Endgame mode
  - defense against the “last-block problem”: cannot finish because missing a few last pieces
  - send requests for missing pieces to all peers in our peer list
  - send cancel messages upon receipt of a piece
BitTorrent: Unchoke

- Periodically (typically every 10 seconds) calculate data-receiving rates from all peers
- Upload to \textit{(unchoke)} the fastest
  - constant number (4) of unchoking slots
  - partition upload bw equally among unchoked

commonly referred to as “tit-for-tat” strategy
Optimistic Unchoking

- Periodically select a peer at random and upload to it
  - typically every 3 unchoking rounds (30 seconds)

- Multi-purpose mechanism
  - allow bootstrapping of new clients
  - continuously look for the fastest peers (exploitation vs exploration)
**BitTorrent Fluid Analysis**

- Normalize file size to 1
- $x(t)$: number of downloaders (also known as leechers) who do not have all pieces at time $t$.
- $y(t)$: number of seeds in the system at time $t$.
- $\lambda$: the arrival rate of new requests.
- $\mu$: the uploading bandwidth of a given peer.
- $c$: the downloading bandwidth of a given peer, assume $c \geq \mu$.
- $\theta$: the rate at which downloaders abort download.
- $\gamma$: the rate at which seeds leave the system.
- $\eta$: indicates the effectiveness of downloader sharing, $\eta$ takes values in $[0, 1]$. 
System Evolution

\[
\frac{dx}{dt} = \lambda - \theta x(t) - \min\{cx(t), \mu(\eta x(t) + y(t))\}, \\
\frac{dy}{dt} = \min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t),
\]

Solving steady state:

\[
\frac{dx(t)}{dt} = \frac{dy(t)}{dt} = 0
\]

Define

\[
\frac{1}{\beta} = \max\left\{\frac{1}{c}, \frac{1}{\eta}\left(\frac{1}{\mu} - \frac{1}{\gamma}\right)\right\}
\]

\[
\bar{x} = \frac{\lambda}{\beta(1 + \frac{\theta}{\beta})}, \\
\bar{y} = \frac{\lambda}{\gamma(1 + \frac{\theta}{\beta})}.
\]
Q: How long does each downloader stay as a downloader?

\[ x = \frac{\lambda}{\beta(1 + \frac{\theta}{\beta})} \]

\[ T = \frac{1}{\theta + \beta} \]

\[ \frac{1}{\beta} = \max\left\{ \frac{1}{c}, \frac{1}{\eta}\left(\frac{1}{\mu} - \frac{1}{\gamma}\right) \right\} \]