eTorrent - writing P2P clients in Erlang

Analysis, Implementation, Philosophy

Jesper Louis Andersen
jesper.louis.andersen@gmail.com

Jun 9th, 2011

Overview

What is BitTorrent? You may already know...

P2P Ideas To consider in other projects...

eTorrent The Implementation...

Computing History

Computing History

- Main Frames

Computing History

- Main Frames
- Client / Server

Computing History

- Main Frames
- Client / Server
- Distributed Mainframes (dynamic server side HTTP)

Computing History

- Main Frames
- Client / Server
- Distributed Mainframes (dynamic server side HTTP)
- Client / Server distributed (JS+HTML5+...)
- CLOUD CLOUD CLOUD CLOUD (Buzzword Bingo!)

Joe Says:

"To make a fault-tolerant system you need at least two CLOUDS!"

Peer-to-peer: Make each client a client+server at the same time.

We are betting this is the future.
BitTorrent is a P2P protocol for content distribution.

HTTP vs BitTorrent

BitTorrent is about *Content distribution*. Some key differences:

HTTP

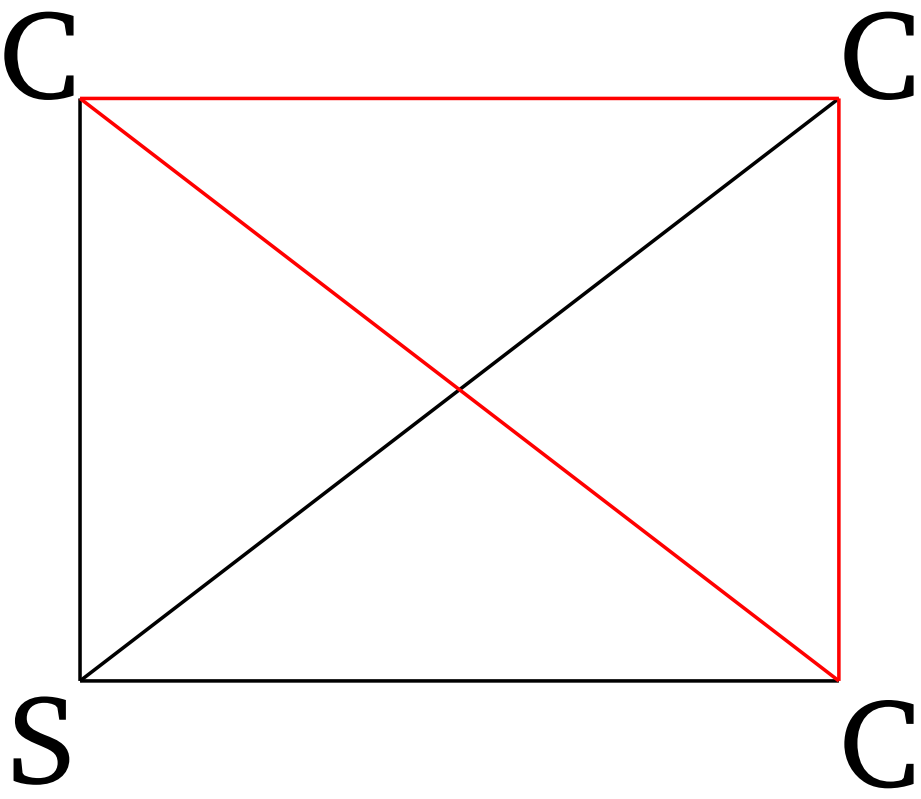
- Simple
- Stateless
- One-to-many
- "Serial"
- Upstream bandwidth heavy

BitTorrent

- Complex
- Stateful
- Peer-2-Peer
- "Concurrent"
- Upstream bandwidth scales proportionally with number of consumers

In BitTorrent everything is sacrificed for the last point.

BitTorrent Idea:



HTTP versus BitTorrent:

- Network 101: C is a set of clients. In a closed network

$$\sum_{c \in C} I_c \leq \sum_{c \in C} O_c$$

- A web server scales 1-to-n: n links, 1 upstream.
- BitTorrent scales: m-to-n: $\binom{n}{2} = \frac{n(n-1)}{2}$ links, n upstreams.

One Slide BitTorrent

- Want to distribute an array of bytes (i.e., a file)
- Utilize concurrency to do it!
- Three phases:
  1. Naming / Identity
  2. Discovery
  3. Exchange

Naming / Identity

- a `.torrent` file is a JSON-like structure, easily Rec. Descent parsed.
- Cut data into *pieces*, cryptographic checksum on each piece in torrent file.
- The torrent file is out fingerprint/DNA providing integrity.
- If we trust the torrent file, we don't have to trust peers.

Discovery

- Find other clients to exchange pieces with.
- Centralized: Contact a *tracker* – web server keeping track of IP/Port pairs.
- Decentralized: Query a *Distributed Hash Table* for the IP/Port pair.

Exchange

- Make TCP connection.
- Handshake, Identify, Negotiate Extensions

Exchange

- Make TCP connection.
- Handshake, Identify, Negotiate Extensions
- The wire protocol is an asynchronous messaging protocol!

Exchange

- Make TCP connection.
- Handshake, Identify, Negotiate Extensions
- The wire protocol is an asynchronous messaging protocol!
- A peer can crash at any point!

Exchange

- Make TCP connection.
- Handshake, Identify, Negotiate Extensions
- The wire protocol is an asynchronous messaging protocol!
- A peer can crash at any point!
- Sounds familiar?

"BitTorrent is just a simple specialization of Erlang Process semantics"

Efficiency

- BitTorrent is extremely *efficient* (saturates)

Efficiency

- BitTorrent is extremely *efficient* (saturates)
- Economy strategy: To optimize yourself egoistically, you must help others.

Efficiency

- BitTorrent is extremely *efficient* (saturates)
- Economy strategy: To optimize yourself egoistically, you must help others.
- The network has an *emergent* behaviour: When each client optimizes itself, the network as a whole benefits.
- The strategy of whom downloads what from whom is not written down in code!

Etorrent – History

Etorrent - A bittorrent client implemented in Erlang

- Erlang/OTP implementation
- Initial Checkin, 27th Dec 2006
- Had first working version around early 2008
- 8 KSLOCs
- Two main developers: Magnus Klaar, Jesper Louis Andersen
- Contributions: Edward Wang, Adam Wolk, Maxim Treskin, Peter Lemenkov, and Tuncer Ayaz.

Why?

- Wanted to learn Erlang

Why?

- Wanted to learn Erlang
- Pick a project - parts not in brain

Why?

- Wanted to learn Erlang
- Pick a project - parts not in brain
- Map/Fold/Filter and FP in general is something I know

Why?

- Wanted to learn Erlang
- Pick a project - parts not in brain
- Map/Fold/Filter and FP in general is something I know
- OTP, Rebar, Common Test, ... not so much

Why?

- Wanted to learn Erlang
- Pick a project - parts not in brain
- Map/Fold/Filter and FP in general is something I know
- OTP, Rebar, Common Test, ... not so much
- Performance model!

Why?

- Wanted to learn Erlang
- Pick a project - parts not in brain
- Map/Fold/Filter and FP in general is something I know
- OTP, Rebar, Common Test, ... not so much
- Performance model!
- A *real* project is an excellent driver

Trying to do things right

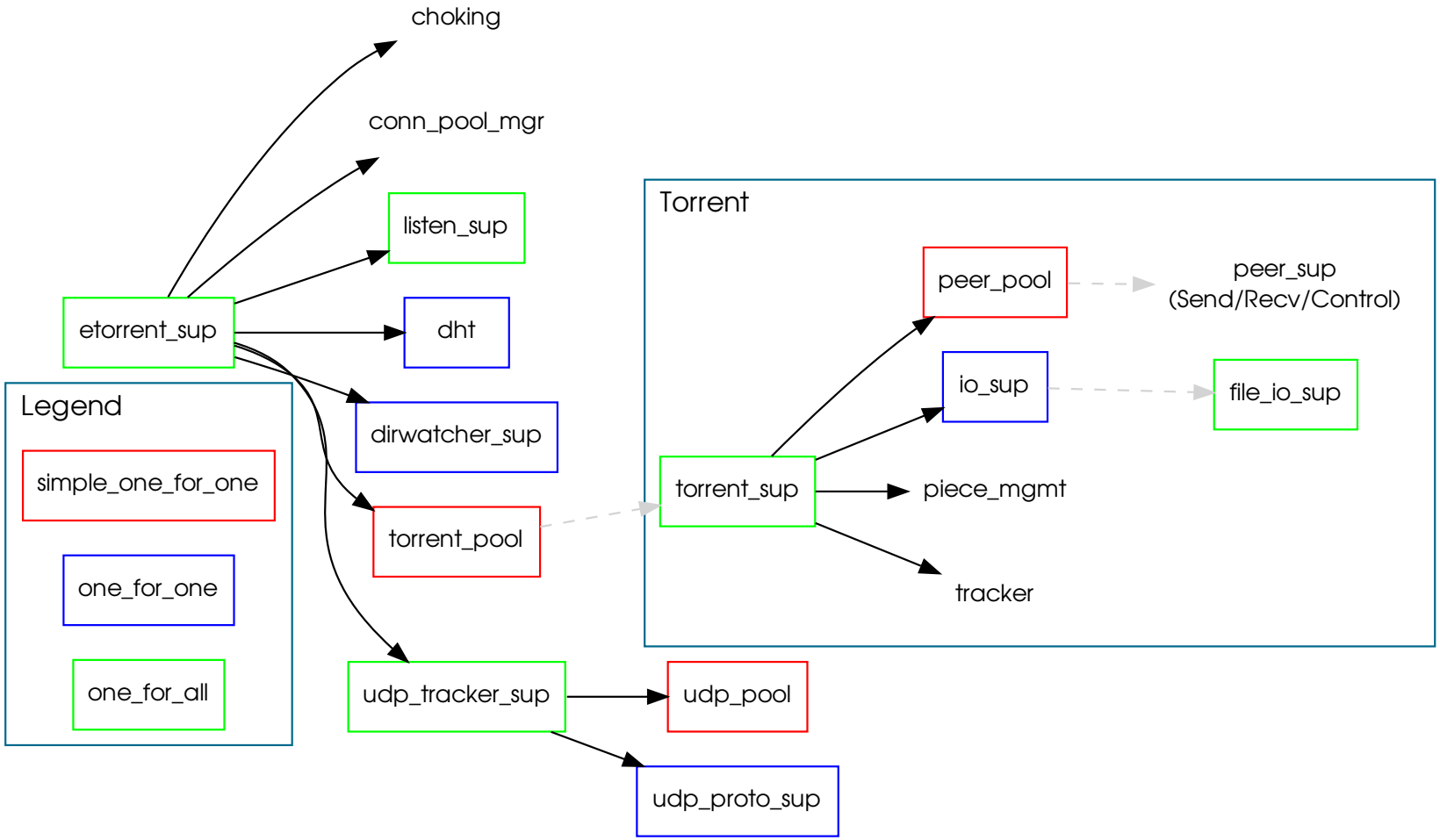
- Git – GitHub
- Well-documented
- Eunit, common test, dialyzer, rebar, QuickCheck/Proper
- OTP all the way
- I use the project for code examples all the time (because everything is in there somewhere)
- Excellent vehicle for explaining stuff on IRC – just drop a github link
- Achilles heel: No distribution yet
- UI is a directory - background operation
- Can be used as-an-application in other Erlang projects

Building it:

- Async messaging, Check!
- Fault tolerance for error handling, Check!
- Built in Concurrency, Check!

- Each peer is independent.
- Some things happen on a Torrent-local scale.
- Some things happen on a global scale.

Etorrent Supervisor Tree:



On Being a Desktop Application

It is different:

On Being a Desktop Application

It is different:

- Case in point: Adobe Flash(tm)

On Being a Desktop Application

It is different:

- Case in point: Adobe Flash(tm)
- Must beat it Memory-wise and CPU-wise

Performance

- Long story short: Under load, we are competitive with C clients.
- For a single torrent we are beaten.
- Use more Memory (Erlangs data representation)
- Comparatively few optimizations has been added to the code base.

Performance

- Long story short: Under load, we are competitive with C clients.
- For a single torrent we are beaten.
- Use more Memory (Erlangs data representation)
- Comparatively few optimizations has been added to the code base.
- Robustness: If we run, we keep running. Fares better than most clients here.

Why can we compete?

- Erlang is flexible, we can try more things
- Erlang is productive, we can iterate faster
- Erlang is fault tolerant, we don't implement all the corner cases

Why can we compete?

- Erlang is flexible, we can try more things
- Erlang is productive, we can iterate faster
- Erlang is fault tolerant, we don't implement all the corner cases
- C programs have no abstraction, so they brute force
- We can choose the right Data Structure or Algorithm

Tricks employed

- No NIFs!
- Erlangs VM has 10+ years optimizations, push down

Tricks employed

- No NIFs!
- Erlangs VM has 10+ years optimizations, push down
- Use ETS, dict, array where applicable

Tricks employed

- No NIFs!
- Erlangs VM has 10+ years optimizations, push down
- Use ETS, dict, array where applicable
- Ignore everything but the critical path

Tricks employed

- No NIFs!
- Erlangs VM has 10+ years optimizations, push down
- Use ETS, dict, array where applicable
- Ignore everything but the critical path
- Do not show *any* mercy on the critical path: `ets:lookup_element/3` over `ets:lookup/2`
- Remember to measure on the critical path!

Fight unfair

- Change the algorithm, use fewer operations
- *Often* possible!

Fight unfair

- Change the algorithm, use fewer operations
- *Often* possible!
- Heuristics: The common case should be fast at the expense of everything else

Fight unfair

- Change the algorithm, use fewer operations
- *Often* possible!
- Heuristics: The common case should be fast at the expense of everything else
- Approximations: Don't go for optimal where near-optimal is equally good and much faster.

Repository

We use github for all code:

`http://www.github.com/jlouis`

Look for *etorrent*