eTorrent - writing P2P clients in Erlang

Analysis, Implementation, Philosophy

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

Mar, 2012

Overview

"And now for something completely different"

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.
Excellent vehicle for studying P2P ideas.

If we need a cloud which is decentralized, it is necessary.

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.

One Slide BitTorrent

- Want to distribute an array of bytes (i.e., a file)
- Utilize concurrency to do it!
- Split file into pieces, exchange them
- Key point 1: One process per peer – crash doesn't matter
- Key point 2: Once a piece passes integrity check, shove it to stable storage.

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

Etorrent – History

Etorrent - A bittorrent client implemented in Erlang

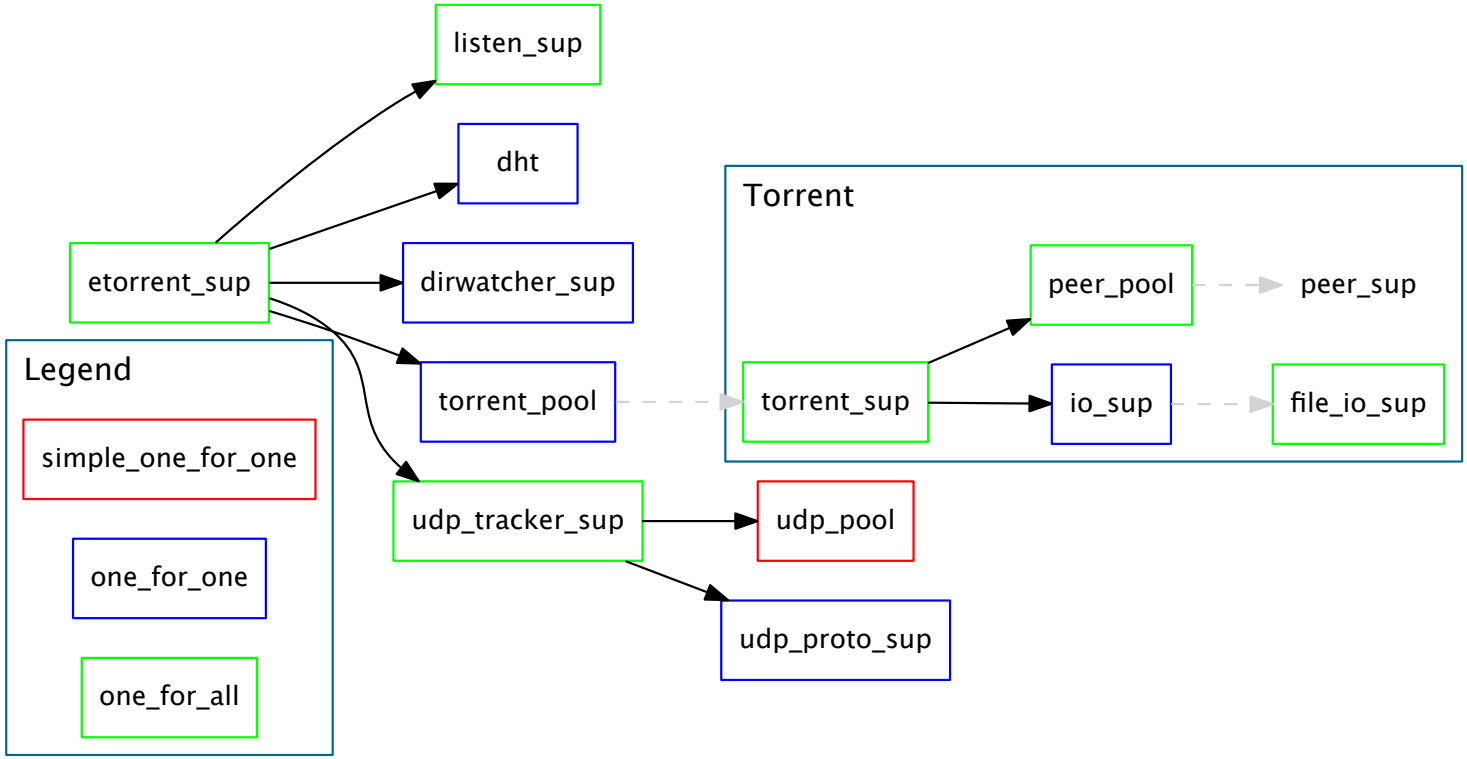
- Erlang/OTP implementation
- Initial Checkin, 27th Dec 2006
- Had first working version around early 2008
- 8 KSLOCs
- Two main developers: Magnus Klaar, Jesper Louis Andersen
- Contributions: Edward Wang, Adam Wolk, Maxim Treskin, Peter Lemenkov, Michael Uvarov and Tuncer Ayaz.

Building it:

- Async messaging!
- Fault tolerance and stable storage makes it robust!
- Built in Concurrency!

- Basic idea for contributions: Get them in, then get them right.
- The contributor is more important than the patch
- Follow Linus Torvalds: Your primary purpose is to get out of the way so people can do work.

Etorrent Supervisor Tree:



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.

New stuff:

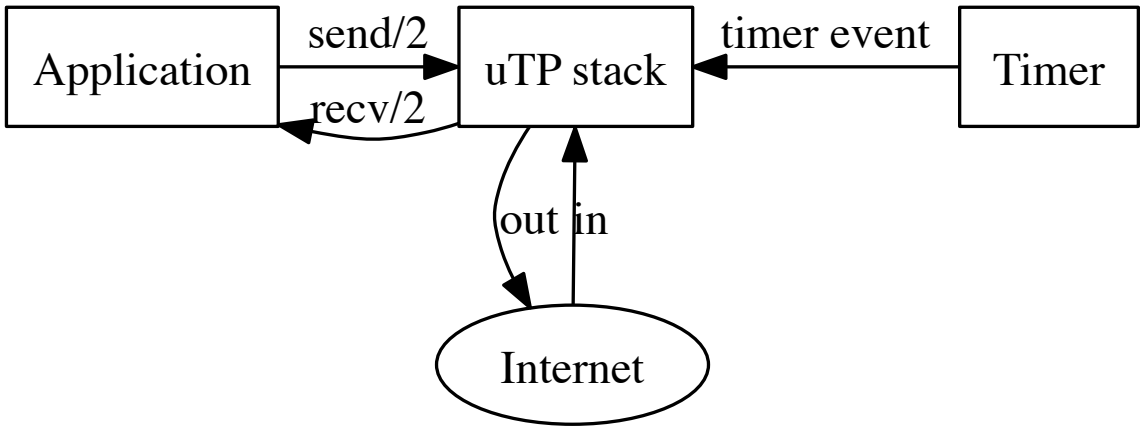
- μtp protocol prototype
- Used in BitTorrent clients
- This beast is, essentially, a TCP implementation + stuff
- Lets do that in Erlang!

WHY? TCP is in trouble

- The problem are buffers on the connection "path".
- *No* buffers is a problem, (See Scott L. Fritchie)
- `http://www.snookles.com/slf-blog/2012/01/05/tcp-incast-what-is-it/`
- *Too large* buffers is a problem (See Jim Gettys)
- `http://gettys.wordpress.com/category/bufferbloat/`

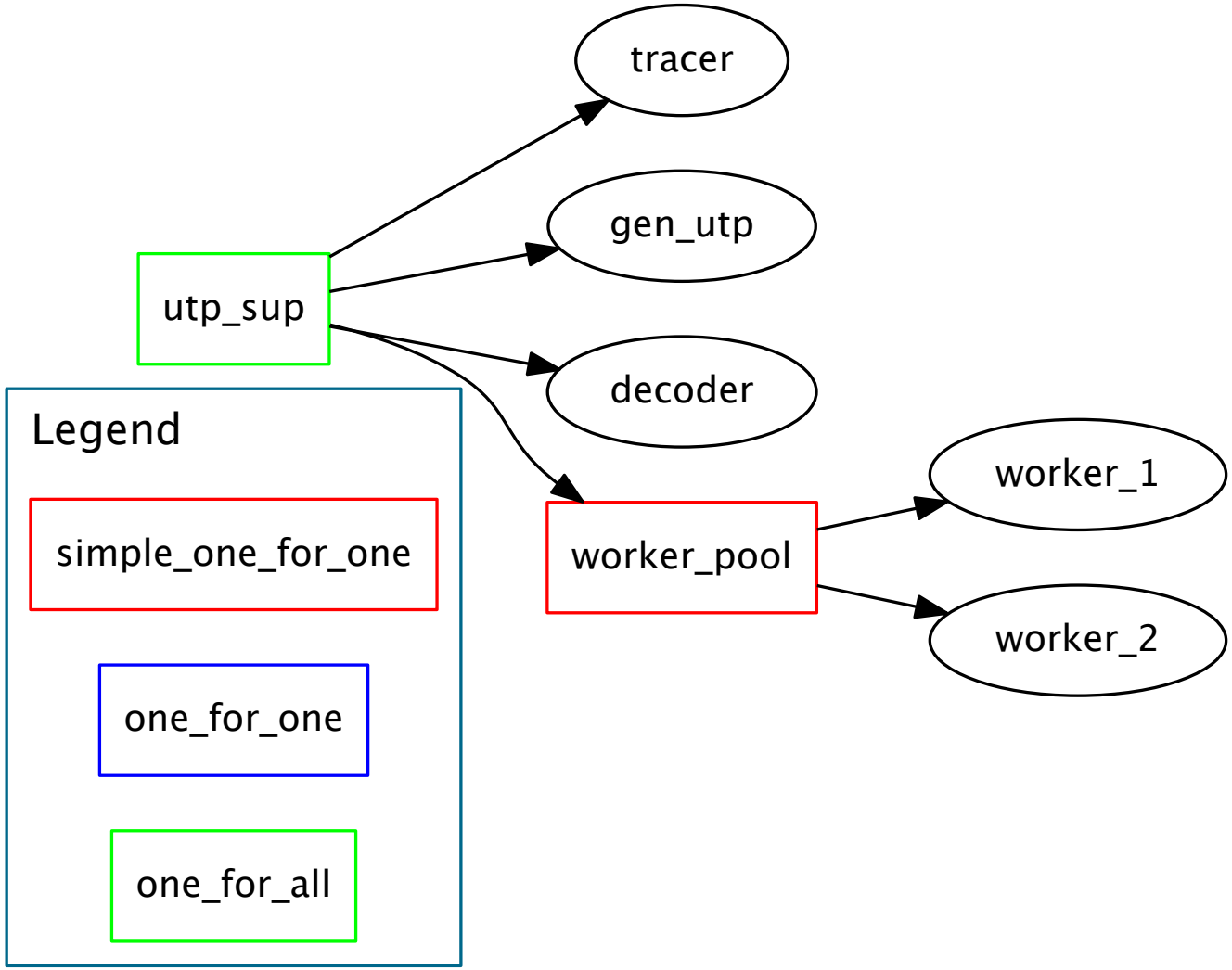
- TCP uses *packet loss* to detect congestion
- "bufferbloat" or "dark buffers" messes with the packet loss
- Idea of μtp: measure the *latency* of the line and use that for congestion control.

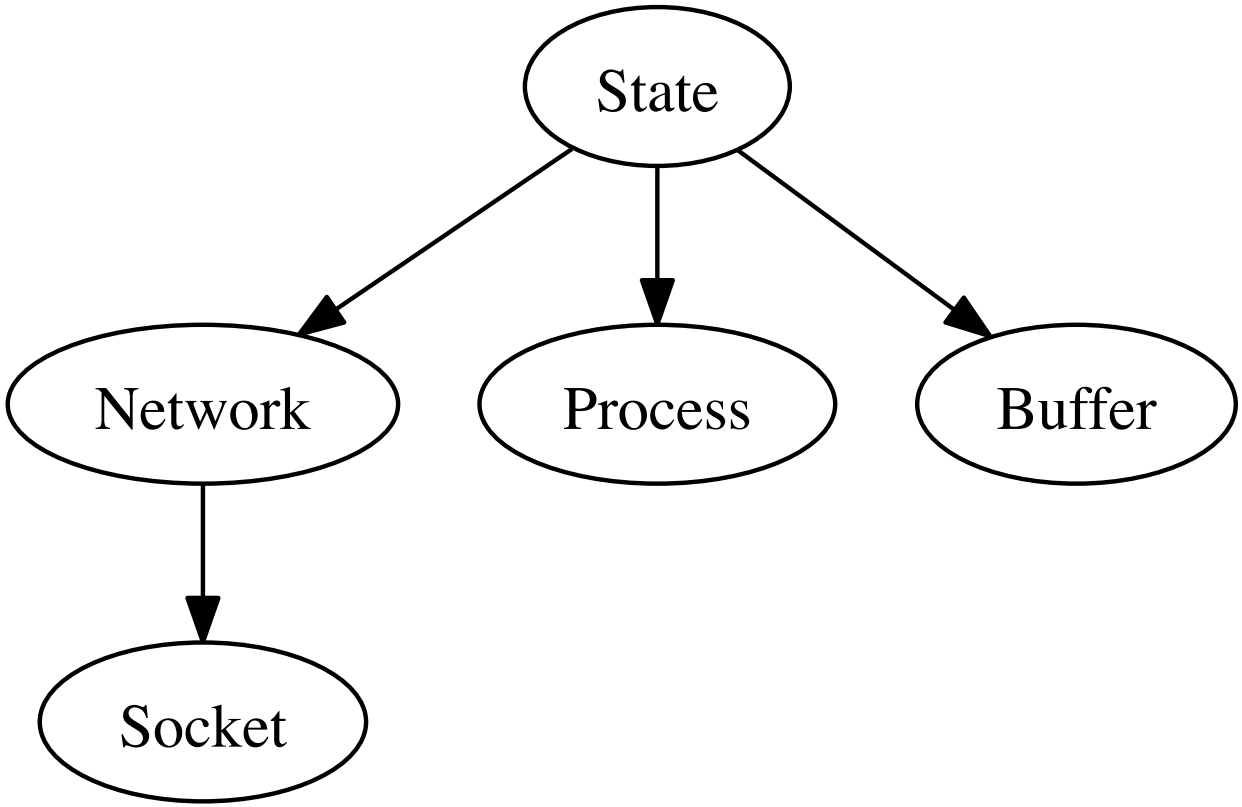
- Very little documentation ("It's like TCP .. but")
- C++ reference implementaion – reverse engineering starts
- Change *Control Flow* oriented code to *Data Flow* oriented code.
- Implement a TCP-like stack in Erlang
- Once we get the model right, this is awfully easy!


Application
send/2
recv/2
uTP stack
timer event
Timer
out
in
Internet

- Key insight no. 1: Find a good process split
- Key insight no. 2: Find a good data split

uTP Supervisor Tree:




State
Network
Process
Buffer
Socket

- Key insight no 3: Avoid Boolean Blindness
- Suppose we compute E to *true*
- We have no *evidence* why E is *true*.

- *true* carries no additional *meaning* so we, as programmers, must *know*. It is but *1* bit of data.
- *true* is no *proof*
- Be worried about booleans, prefer constructing more structured terms which *tell*
- Match on terms!

```
case length(List) == 0 of
  true -> ...;
  false -> ... H = hd(List) ... T = tl(List)
end,

case List of
  [] -> ...
  [H | T] -> ...
end
```

Worrying thought: Whenever you do a boolean you may be doing ex1 here!
Equality is the scourge of computing!

- Write an *analyzer* returning a term / data type providing evidence
- Write an *executor* case..end on the analysis
- Splits concerns
- Avoids you having to *recompute* the evidence of "why"
- Gives additional information

```
handle_receive_buffer(SeqNo, Payload,
                      PacketBuffer, State) ->
    case update_recv_buffer(SeqNo, Payload,
                            PacketBuffer, State) of
        duplicate -> {PacketBuffer, [{send_ack, true}]};
        {ok, #buffer{} = PB} ->
            {PB, consider_send_ack(PacketBuffer, PB)};
        {got_fin, #buffer{} = PB} ->
            {PB, [{got_fin, true},
                  {send_ack, true}]}
    end.
```

Current State

- Our μtp stuff works - about 80 percent implemented
- Tested with Linux NetEM locally and over the internet
- Can't talk with the reference implementation – yet.

- `https://github.com/jlouis/etorrent`
- Questions?