Rubyzg

RUBY HTTP CLIENTS

comparison by

zoran.majstorovic@promdm.com

PROMDM

Professional **M**obile **D**evice **M**anagement

RUBY HTTP CLIENTS

- What they are?
- Why do we need them?
- Which one to choose for your next ruby project?

RUBY HTTP CLIENT

- can be used in ruby code

RUBY HTTP CLIENT

- can be used in ruby code
- to perform HTTP requests

RUBY HTTP CLIENT

Hypertext Transfer Protocol -- HTTP/1.1

RFC 2616

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.

...

RUBY HTTP CLIENT

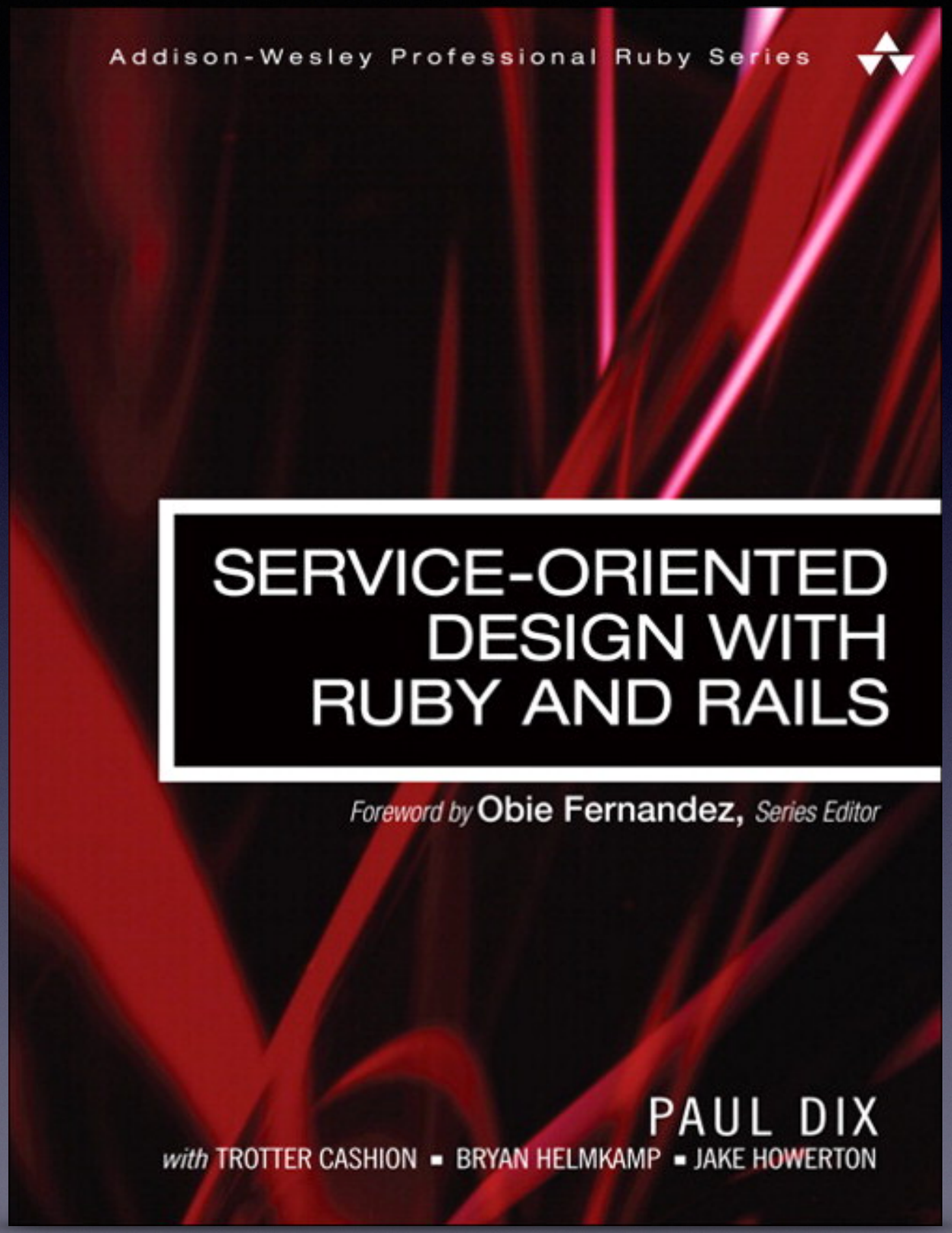
RFC 2616

1.3 Terminology

client
   A program that establishes connections for the purpose of sending
   requests.

user agent
   The client which initiates a request. These are often browsers,
   editors, spiders (web-traversing robots), or other end user tools.

A Service-Oriented Approach to System Design



RUBY HTTP CLIENT

pure **HTTP**/1.1 RFC 2616

```ruby
require 'net/http'
puts Net::HTTP.get(
```

github.com/ruby/ruby/blob/trunk/ext/socket/
socket.c
~2k LoC by **Yukihiro Matsumoto**

https://gist.github.co

```ruby
require 'socket'

host = "localhost"
port = 2345

socket = TCPSocket.new host, port

msg = "GET / HTTP/1.1\r\n" +
      "Accept: */*\r\n" +
      "User-Agent: Ruby/1.0\r\n" +
      "Host: #{host}:#{port}\r\n" +
      "\r\n"

socket.write(msg)

while line = socket.gets
  puts line
end

socket.close unless socket.closed?
```

```
    activesupport (4.2.6)
      i18n (~> 0.7)
      json (~> 1.7, >= 1.7.7)
      minitest (~> 5.1)
      thread_safe (~> 0.3, >= 0.3.4)
      tzinfo (~> 1.1)
    acts-as-taggable-on (3.5.0)
      activerecord (>= 3.2, < 5)
    addressable (2.4.0)
    ansi (1.5.0)
    apple_dep_client (2.1.0)
      oauth (~> 0.4.7)
      plist (~> 3.1.0)
      typhoeus (~> 0.7)
    arel (6.0.3)
    autoprefixer-rails (6.3.6.1)
      execjs
      ...
      multi_json (1.12.1)
    multi_xml (0.5.5)
    multipart-post (2.0.0)
    net-ldap (0.14.0)
    nokogiri (1.6.7.2)
      mini_portile2 (~> 2.0.0.rc2)
    oauth (0.4.7)
    oauth2 (1.1.0)
      faraday (>= 0.8, < 0.10)
      jwt (~> 1.0, < 1.5.2)
      multi_json (~> 1.3)
      multi_xml (~> 0.5)
      rack (>= 1.2, < 3)
    omniauth (1.3.1)
      hashie (>= 1.2, < 4)
      rack (>= 1.0, < 3)
    omniauth-oauth2 (1.3.1)
```

RUBY HTTP CLIENTS

found in `Gemfile.lock`

The Universe of

RUBY HTTP CLIENTS

Net::HTTP

Rest-Client

HTTParty

EM-HTTP-Request

Faraday

HTTP Client

HTTP.rb

Curb

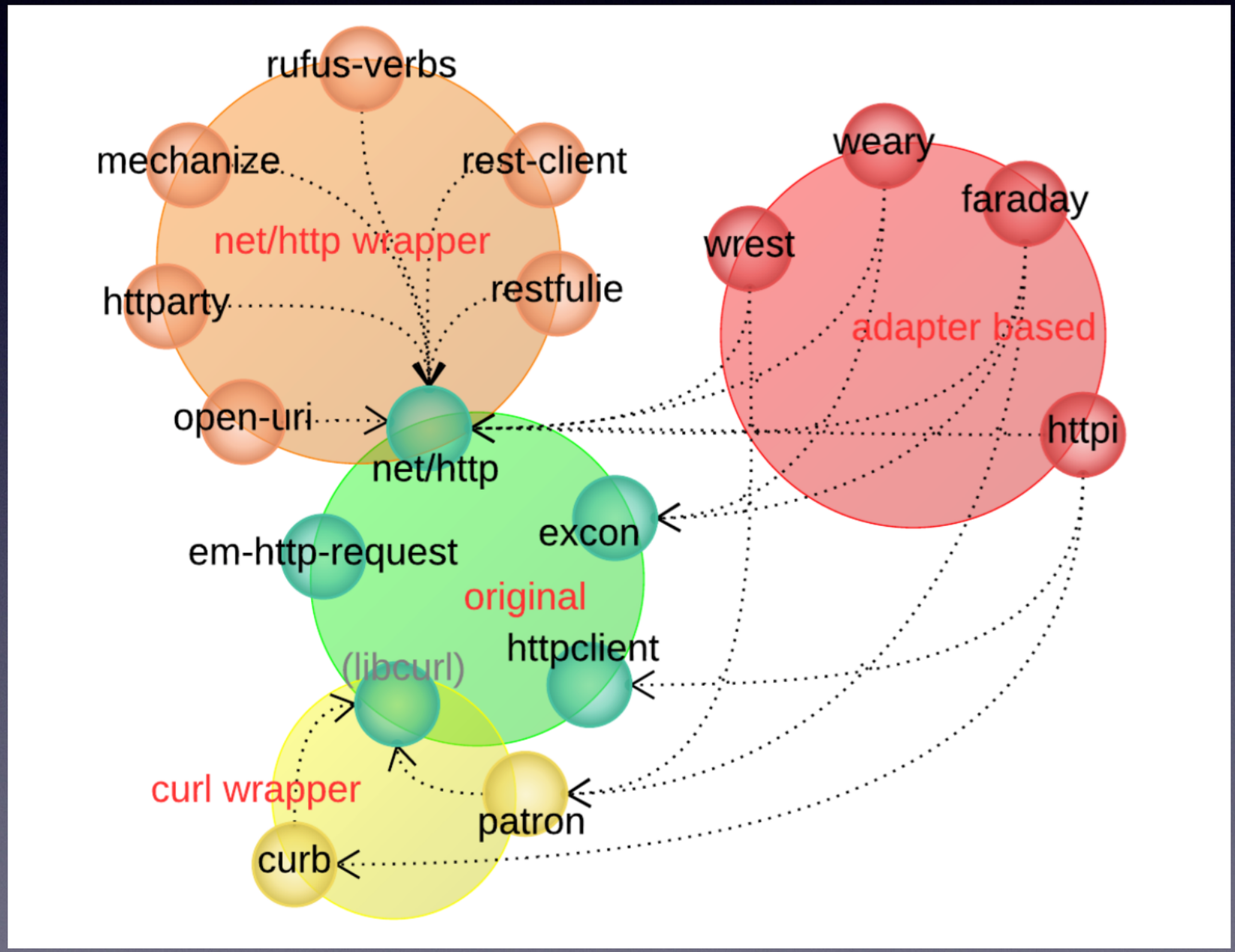
Typhoeus

Patron

Excon

RUBY HTTP CLIENTS

about
4 years ago
by Hiroshi
Nakamura



http://bit.ly/RubyHTTPClients2012

RUBY HTTP CLIENTS

taxonomy based on library dependecy

- **Net::HTTP**, **HTTPClient** based on TCPSocket (Ruby C library)
- **HTTParty**, **HTTP.rb**, **Rest-Client** based on Net::HTTP
- **Typhoeus**, **Curb**, **Patron** based on libcurl (C library)
- **EM-HTTP-Request**, **Excon** based on *eventmachine* (C++ library)
- **Faraday** "the wrapper gem" with adapters for: Net::HTTP (default), HTTPClient, Typhoeus, Patron, EM-HTTP-Request, Excon

RUBY HTTP CLIENTS taxonomy based on library dependecy

HTTParty

Rest-Client

Net::HTTP

HTTP.rb

HTTP Client

TCPSocket
Ruby C lib

Faraday

EM-HTTP-Request

eventmachine C++ library

Excon

Typhoeus

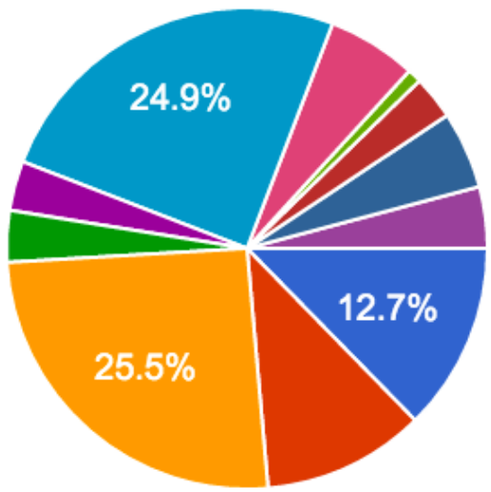
Curb

libcurl C library

Patron

Avdi's Survey about **RUBY HTTP CLIENTS**

What is your go-to Ruby HTTP client library?



Built-in Net::HTTP	**95**	12.7%
Rest-Client	**82**	10.9%
HTTParty	**191**	25.5%
HTTP.rb	**26**	3.5%
Excon	**25**	3.3%
Faraday	**187**	24.9%
Typhoeus	**45**	6%
Patron	**7**	0.9%
Curb	**22**	2.9%
HTTPClient	**39**	5.2%
Other	**31**	4.1%

http://devblog.avdi.org/2015/10/16/results-of-ruby-http-client-library-survey

Avdi's Survey

HTTParty	simplicity and ease of use
Faraday	ability to modify its behavior with middlewares, swap backend libraries
Net::HTTP	built-in, no extra dependency, robustness and stability
Rest-Client	easy API, just works, can do file uploads
HTTPClient	thread-safe, uses keepalive, fast, supports http streaming
Excon	easy to use, customize, easy to handle errors, has feature to debug
HTTP.rb	thread safety (jRuby), a sane API for SSL (e.g. mutual auth)
Typhoeus	concurrent requests and multipart posts actually work
Curb	benchmarks showed that curb is by far the fastest one
Patron	nicer API than Curb, easy to set timeout, easy to log and do REST actions

http://devblog.avdi.org/2015/10/16/results-of-ruby-http-client-library-survey

RUBY HTTP CLIENTS

	Avdi's Survey	GithHub Stars *
HTTParty	25.5%	3.661
Faraday	24.9%	3.119
Net::HTTP	12.7%	-
Rest-Client	10.9%	3.385
Typhoeus	6%	2.798
HTTPClient	5.2%	517
HTTP.rb	3.5%	1.540
Excon	3.3%	725
Curb	2.9%	973
Patron	0.9%	472

* GithHub Stars count on 01 June 2016

RUBY HTTP CLIENTS

Code Samples for Top 5

HTTParty

Faraday

Net::HTTP

Rest-Client

Typhoeus

HTTParty

```ruby
class Google
  include HTTParty
  format :html
end

# google.com
pp Google.ge
```

```ruby
class HtmlParserIncluded < HTTParty::Parser
  def html
    Nokogiri::HTML(body)
  end
end

class Page
  include HTTParty
  parser HtmlParserIncluded
end

pp Page.get('http://www.google.com')
```

Faraday

```ruby
conn = Faraday.new(:url => 'http://sushi.com') do |faraday|
  faraday.request  :url_encoded             # form-encode POST params
  faraday.response :logger                  # log requests to STDOUT
  faraday.adapter  Faraday.default_adapter  # make requests with Net::HTTP
end
```

```ruby
## GET ##

response = conn.get '/nigiri/sake.json'      # GET http://sushi.com/nigiri/sake.json
response.body

conn.get '/nigiri', { :name => 'Maguro' }    # GET http://sushi.com/nigiri?name=Maguro

conn.get do |req|                            # GET http://sushi.com/search?page=2&limit=100
  req.url '/search', :page => 2
  req.params['limit'] = 100
end
```

Faraday adapters

	Survey	GitHub Stars
HTTParty	25.5%	3.661
Faraday ◆	24.9%	3.119
Net::HTTP ●	12.7%	-
Rest-Client	10.9%	3.385
Typhoeus ●	**6%**	2.798
HTTPClient ●	5.2%	517
HTTP.rb	3.5%	1.540
Excon ●	3.3%	725
Curb	2.9%	973
Patron ●	0.9%	472
em-http-request ●	-	3.661

Net::HTTP

```ruby
Net::HTTP.get('example.com', '/index.html') # => String
```

```ruby
uri = URI('http://example.com/index.html?count=10')
Net::HTTP.get(uri) # => String
```

```ruby
uri = URI('http://www.example.com/search.cgi')
res = Net::HTTP.post_form(uri, 'q' => 'ruby', 'max' => '50')
puts res.body
```

Net::HTTP

```ruby
http = Net::HTTP.new(uri.host, uri.port)

request = Net::HTTP::Post.new(uri.request_uri)
request.set_form_data({"q" => "My query", "per_page" => "50"})

# Tweak headers, removing this will default to application/x-www-form-urlencoded
request["Content-Type"] = "application/json"

response = http.request(request)
```

https://github.com/augustl/net-http-cheat-sheet

Rest-Client

a simple DSL for accessing HTTP and REST resources

```
RestClient.get 'http://example.com/resource'

RestClient.get 'http://example.com/resource', {:params => {:id => 50, 'foo' => 'bar'}}

RestClient.get 'https://user:password@example.com/private/resource', {:accept => :json}

RestClient.post 'http://example.com/resource', :param1 => 'one', :nested => { :param2 => 'two' }

RestClient.post "http://example.com/resource", { 'x' => 1 }.to_json, :content_type => :json, :accept => :js

RestClient.delete 'http://example.com/resource'
```

```
RestClient.post( url,
  {
    :transfer => {
      :path => '/foo/bar',
      :owner => 'that_guy',
      :group => 'those_guys'
    },
     :upload => {
      :file => File.new(path, 'rb')
    }
  })
```

Typhoeus

A single request:

```
Typhoeus.get("www.example.com", followlocation: true)
```

Parallel requests:

```
hydra = Typhoeus::Hydra.new
10.times.map{ hydra.queue(Typhoeus::Request.new("www.example.com", followlocation: true)) }
hydra.run
```

Cookies

```
Typhoeus::Request.get("www.example.com", cookief
```

```
Typhoeus.post(
  "http://localhost:3000/posts",
  body: {
    title: "test post",
    content: "this is my test",
    file: File.open("thesis.txt","r")
  }
)
```

RUBY HTTP CLIENTS

Speed Tests

DISCLAIMER: Most benchmarks you find in READMEs are crap, including this one. These are out-of-date. If you care about performance, benchmark for yourself for your own use cases!

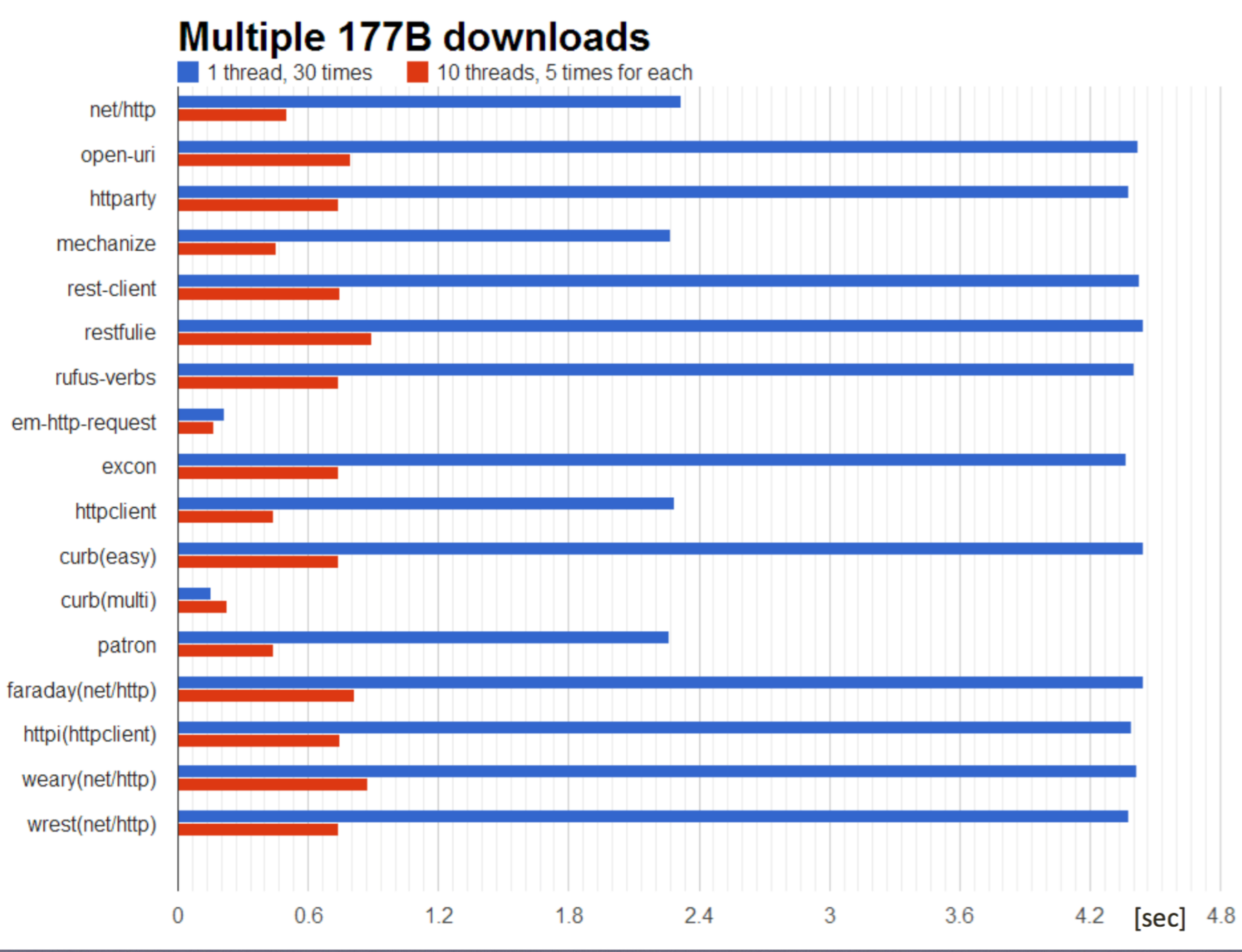
https://github.com/httprb/http/blob/master/README.md

HTTP client	Time	Implementation
curb (persistent)	2.519	libcurl wrapper
em-http-request	2.731	EM + http_parser.rb
Typhoeus	2.851	libcurl wrapper
StreamlyFFI (persistent)	2.853	libcurl wrapper
http.rb (persistent)	2.970	Ruby + http_parser.rb
http.rb	3.588	Ruby + http_parser.rb
HTTParty	3.931	Net::HTTP wrapper
Net::HTTP	3.959	Pure Ruby
Net::HTTP (persistent)	4.043	Pure Ruby
open-uri	4.479	Net::HTTP wrapper
Excon (persistent)	4.618	Pure Ruby
Excon	4.701	Pure Ruby
RestClient	26.838	Net::HTTP wrapper

RUBY HTTP CLIENTS

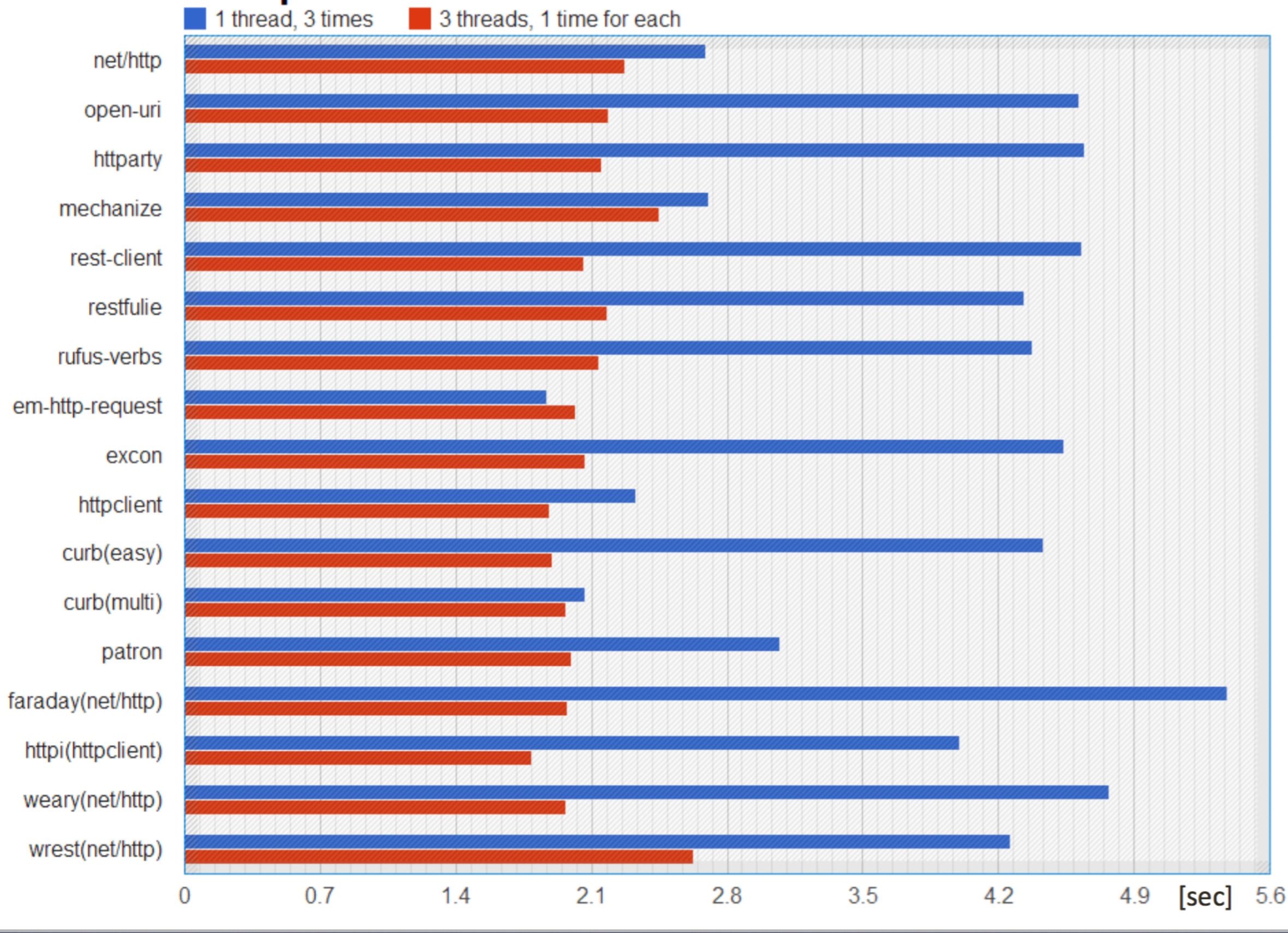
Speed Tests
by @tarcieri

https://github.com/httprb/http/blob/master/README.md



Speed Tests by @nahi

Multiple 24MB downloads



Speed Tests by @nahi

RUBY HTTP CLIENTS

that's all

thanks