Use OpenResty XRay to Optimize Open Source OpenResty and its Applications

Yichun Zhang

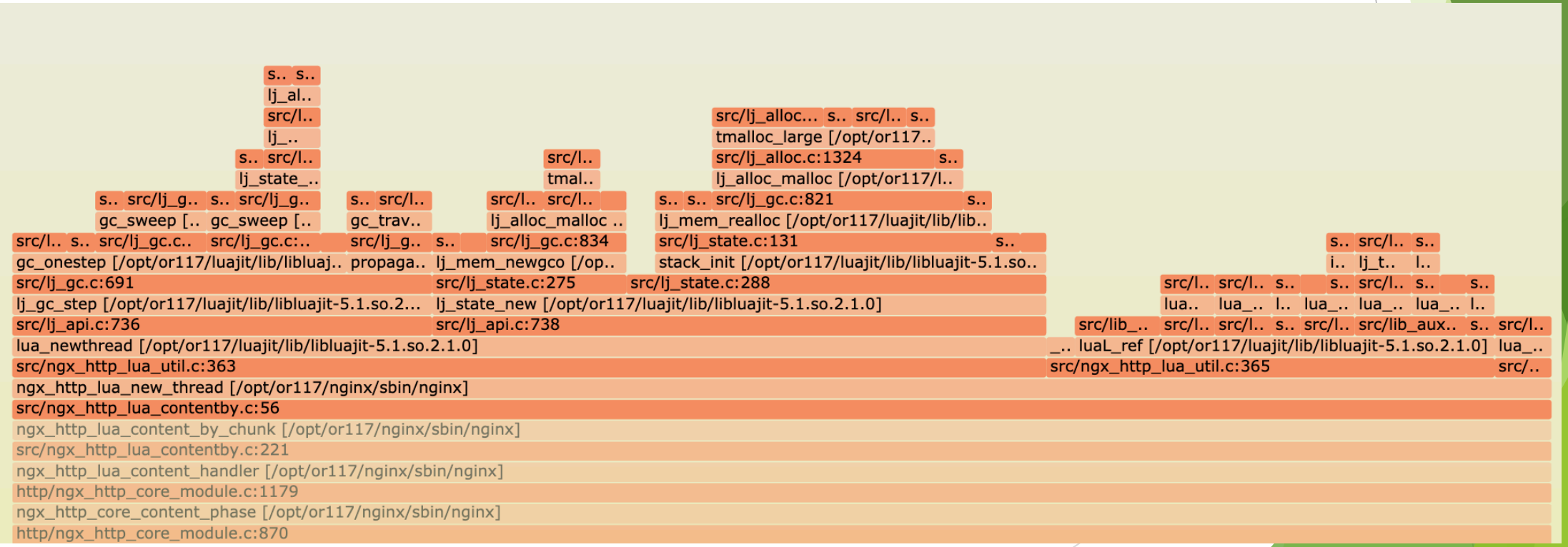
Founder & CEO of OpenResty Inc.

OpenResty 1.19.3.1

- 升级 Nginx 核心至 1.19.3.
- 升级 LuaJIT，从 Mike Pall 维护的的上游同步
- 优化：新增轻量级线程池来复用 Lua 轻线程
- 优化：给 LuaJIT 新增 lua_getexdata2() 和 lua_setexdata2() API，将单个请求内协程查找元信息时间复杂度从 O(n) 降低到 O(1)
- 优化：在 tcpsock/udpsock:send(LUA_TABLE) 等 API 中避免 Lua number 到 Lua string 的转化，减少创建不必要的 GC 对象
- 改进：ssl_*_by_lua* 阶段设置的 ngx.ctx，将被后续阶段所继承
- 新增：ngx.ssl 模块的 very_cert() API 允许动态设置客户端证书，开启客户端证书验证
- 新增：exit_worker_by_lua* 指令，可以更方便的在 worker 退出的阶段执行任务
- 新增：ngx.balancer 模块的 recreate_request() API，从而可以在 balancer 阶段改写上游 请求的信息，并且通过这个 API 使之生效
- 新增：ngx.req.socket(true?) 创建的 tcpsock 对象，也支持 receiveany() 方法
- 新增：ngx.ssl 模块的 server_port() API 可以获取 server_port
- non-SSE-4.2 CPUs
- http2 ngx.flush(true)

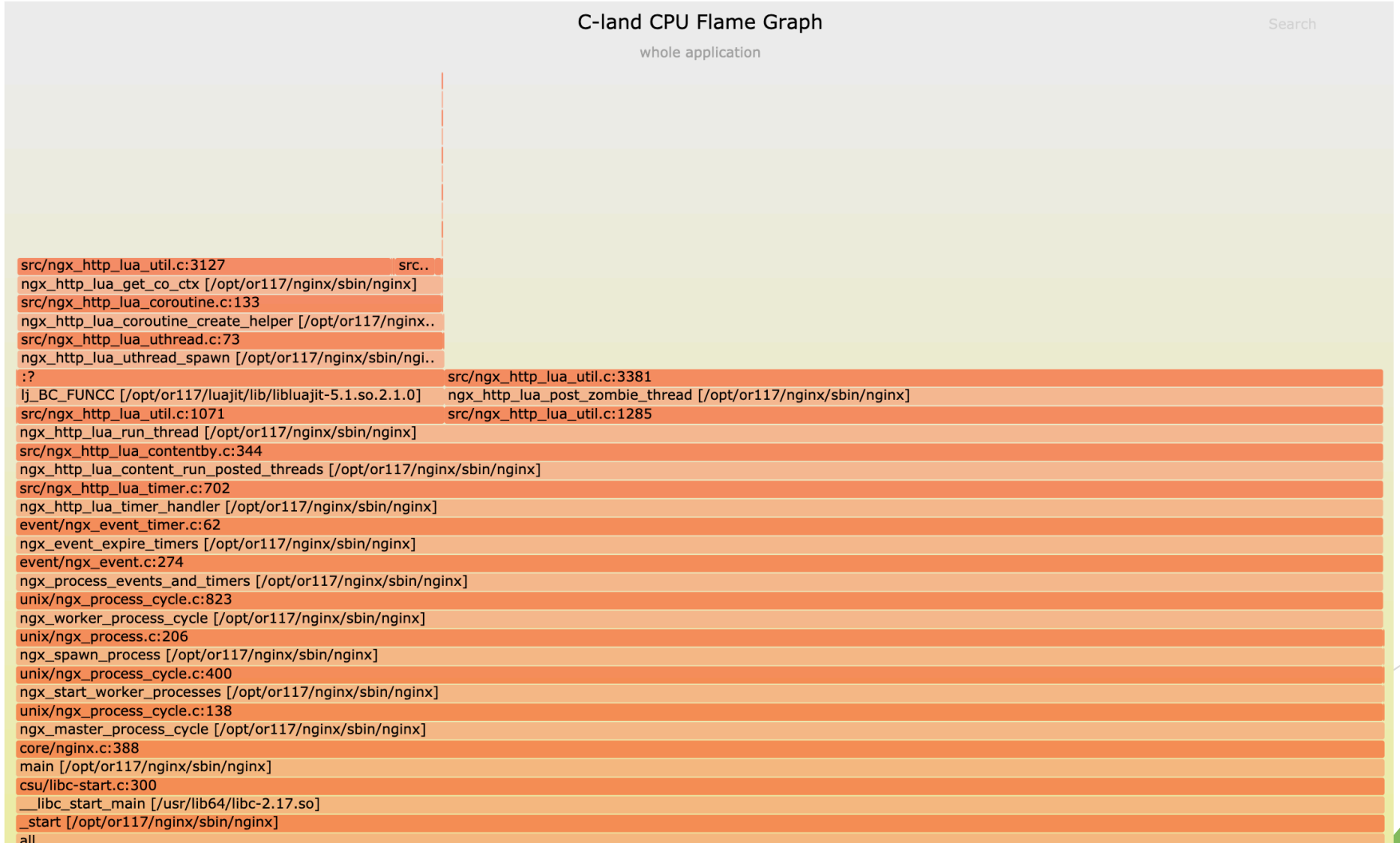
Lua light thread creation and release

- Ngx_http_lua_new_thread



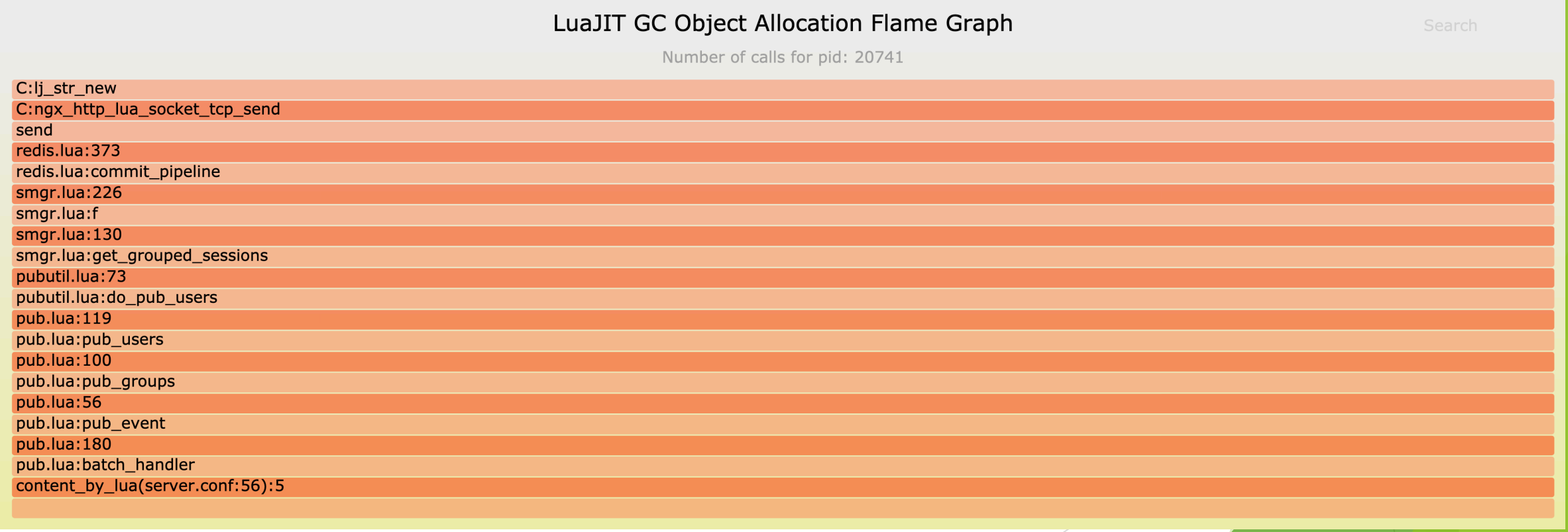
```
master_process on;
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    access_log off;
    keepalive_timeout 60s;

    server {
        listen 8080 reuseport;

        location / {
            content_by_lua_block {
                ngx.say("hello")
            }
        }
    }
}
```

Lua thread metadata lookup (co ctx) ngx_http_lua_get_co_ctx

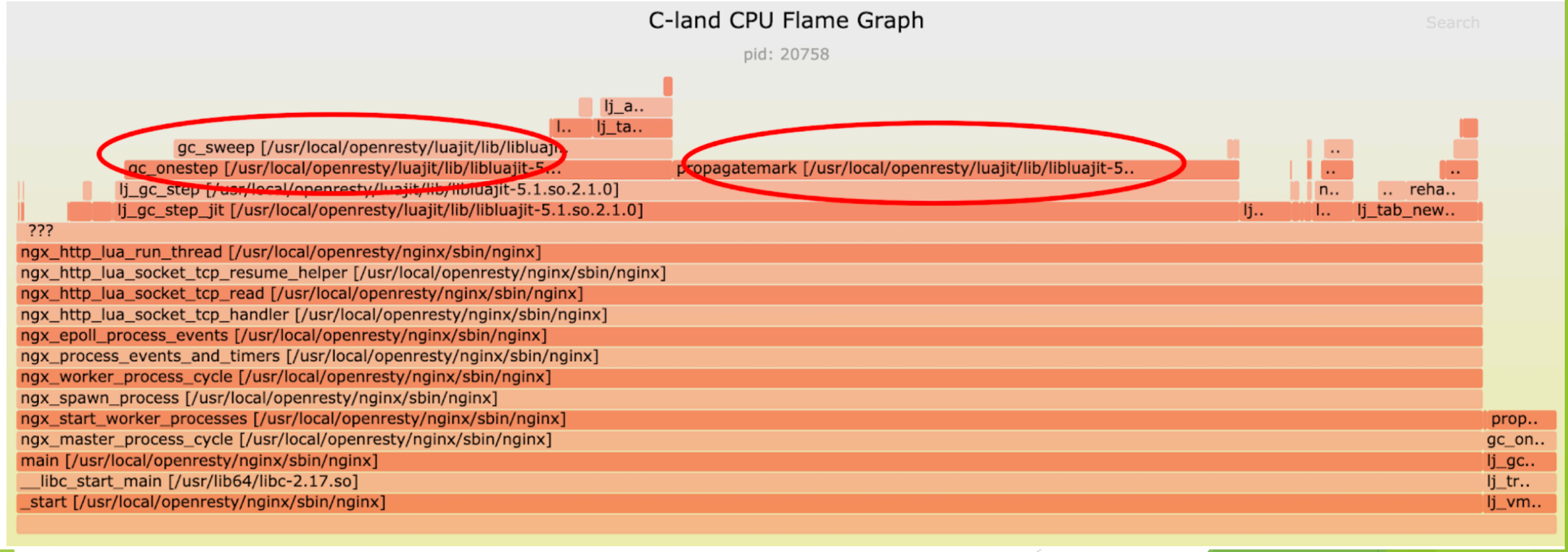


```
14      init_worker_by_lua_block {
15          ngx.timer.at(0, function ()
16              local n = 100000
17
18              local function f()
19              end
20
21              local thrs = require "table.new"(n, 0)
22              for i = 1, n do
23                  thrs[i] = ngx.thread.spawn(f)
24              end
25
26              for i = 1, n do
27                  assert(ngx.thread.wait(thrs[i]))
28              end
29
30              local function g()
31                  local exiting = ngx.worker.exiting
32                  while not exiting() do
33                      ngx.sleep(0)
34                  end
35              end
36
37              ngx.thread.spawn(g)
38          end)
39      }
40
41      server {
42          listen 8081 reuseport;
43
44          location / {
45              return 200 ok;
46          }
47      }
```

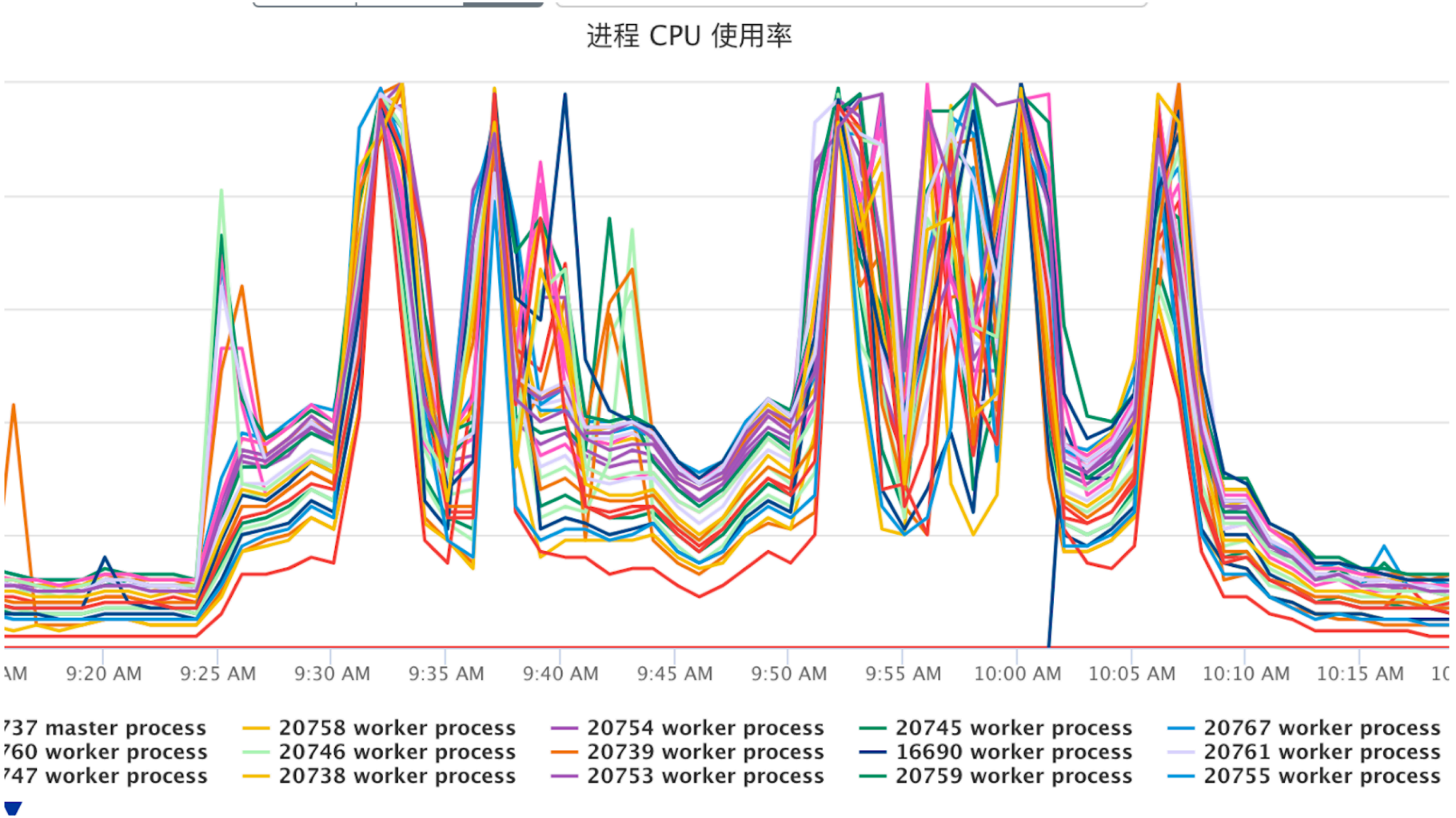
Temporary Lua string creations in cosocket:send() as seen on customer's machines



Too many temp GC objects lead to large LuaJIT GC overhaed

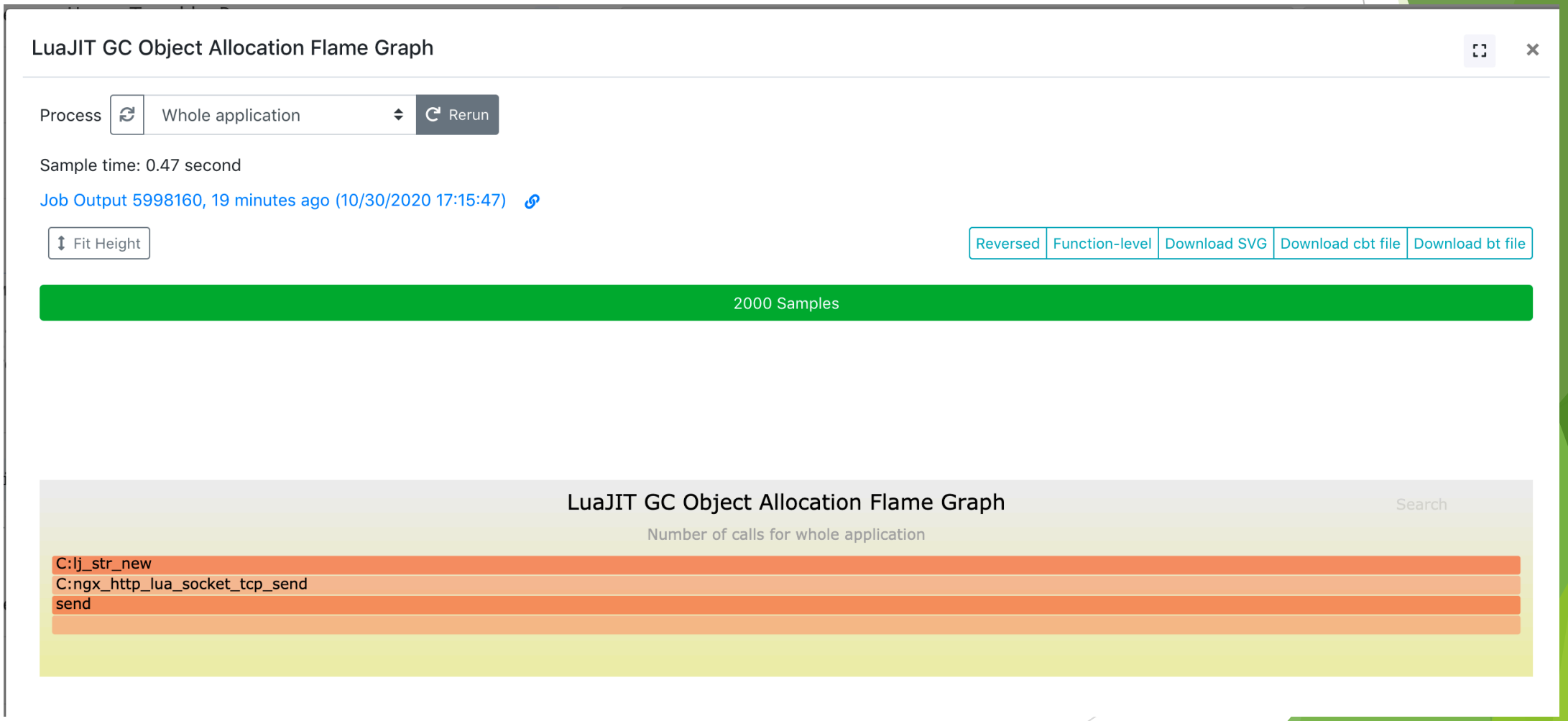


Leading to intermittent CPU spikes

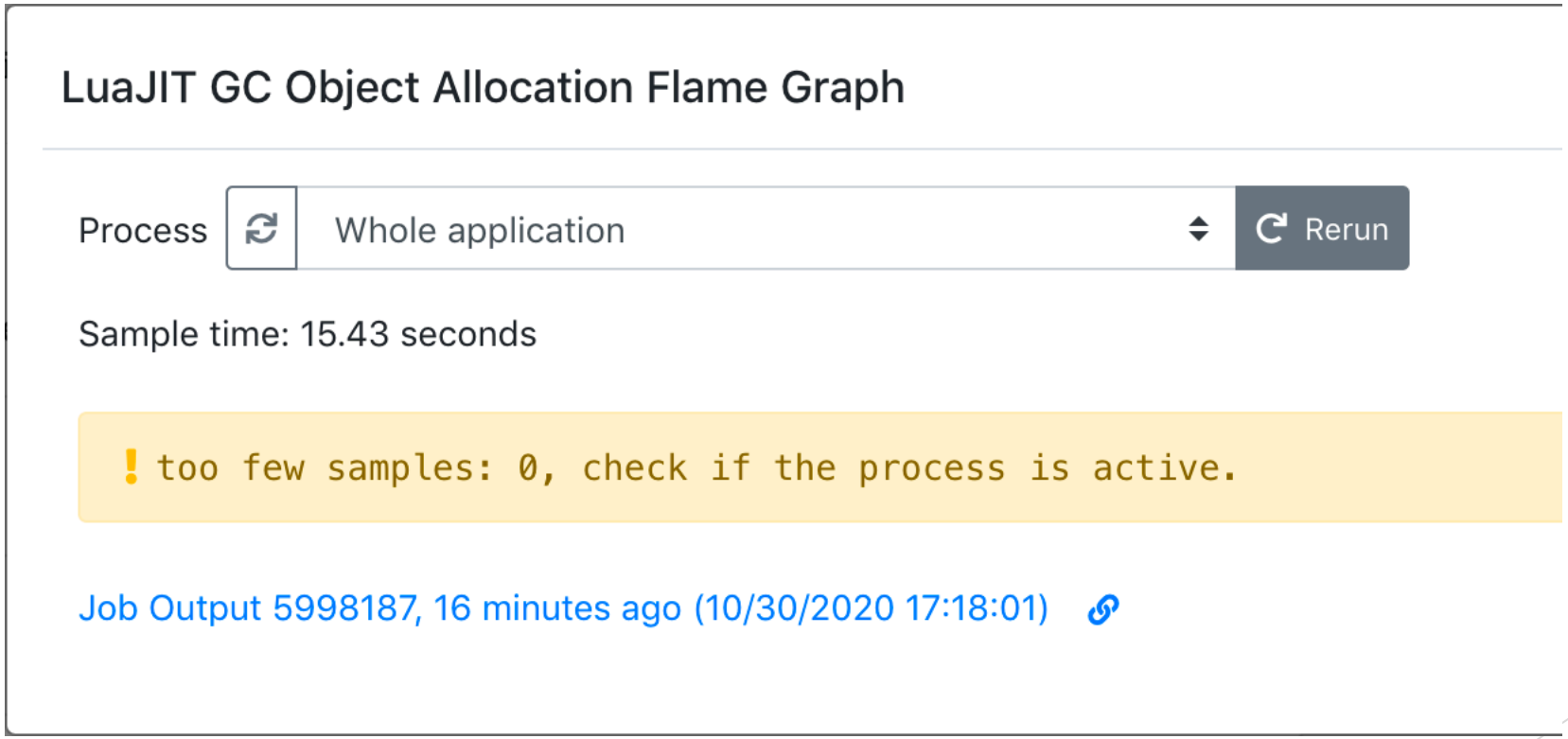


```
    init_worker_by_lua_block {
        --[[
        ngx.timer.at(0, function ()
            while not ngx.worker.exiting() do
                local sock = ngx.socket.tcp()
                assert(sock:connect("127.0.0.1", 5000))
                for i = 10000, 20000 do
                    sock:send(i)
                end
                assert(sock:setkeepalive(10))
            end
        end)
        ]]
    }

    server {
        listen 8082 reuseport;

        location / {
            return 200 ok;
        }
    }
```

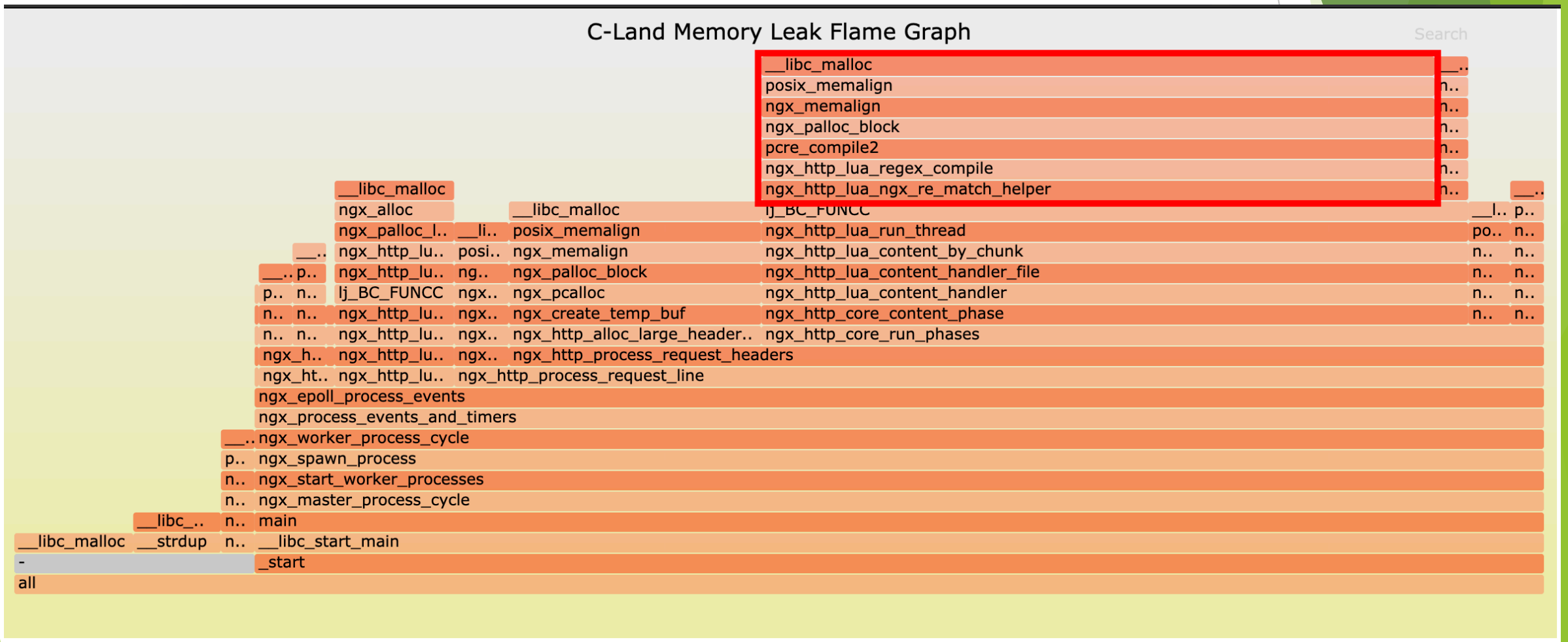
Before the sock:send() optimization



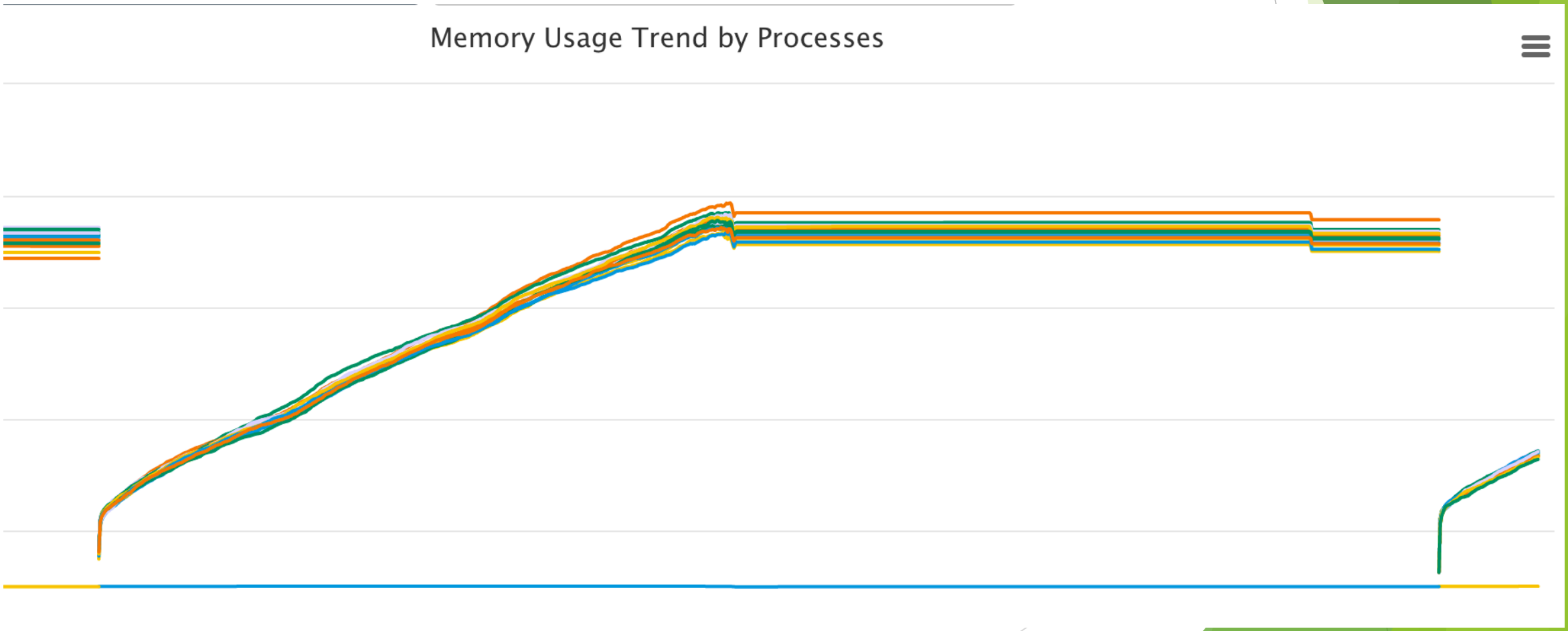
After the sock:send() optimization



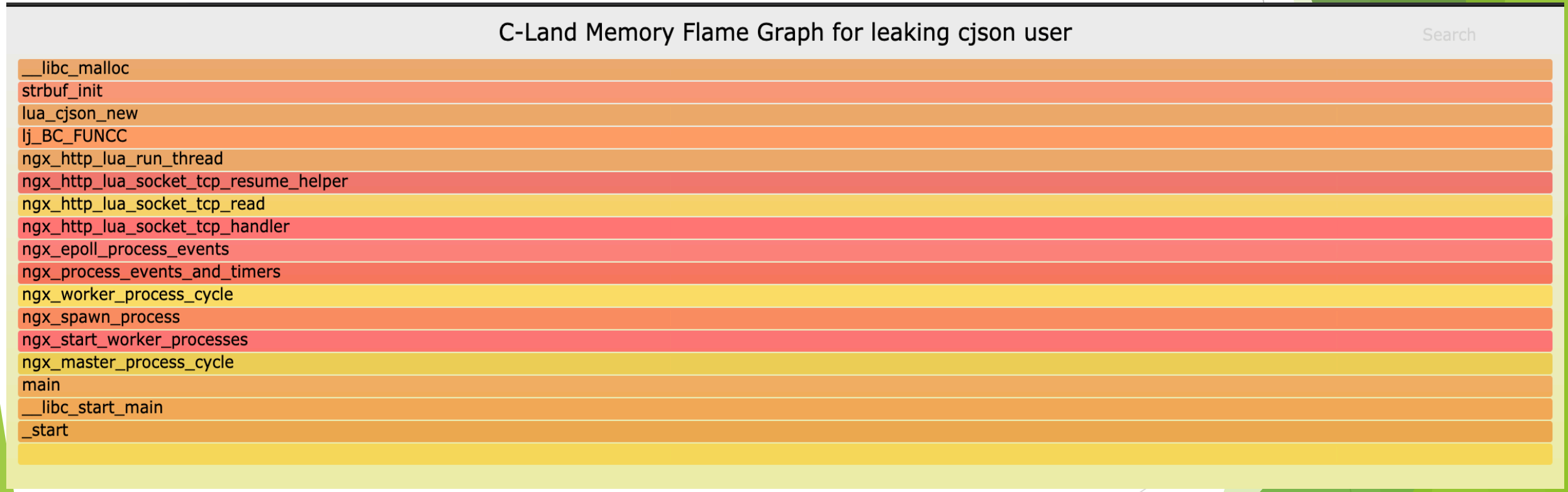
Memory leaks in ngx.re.match of older OpenResty (use lua-resty-core!)



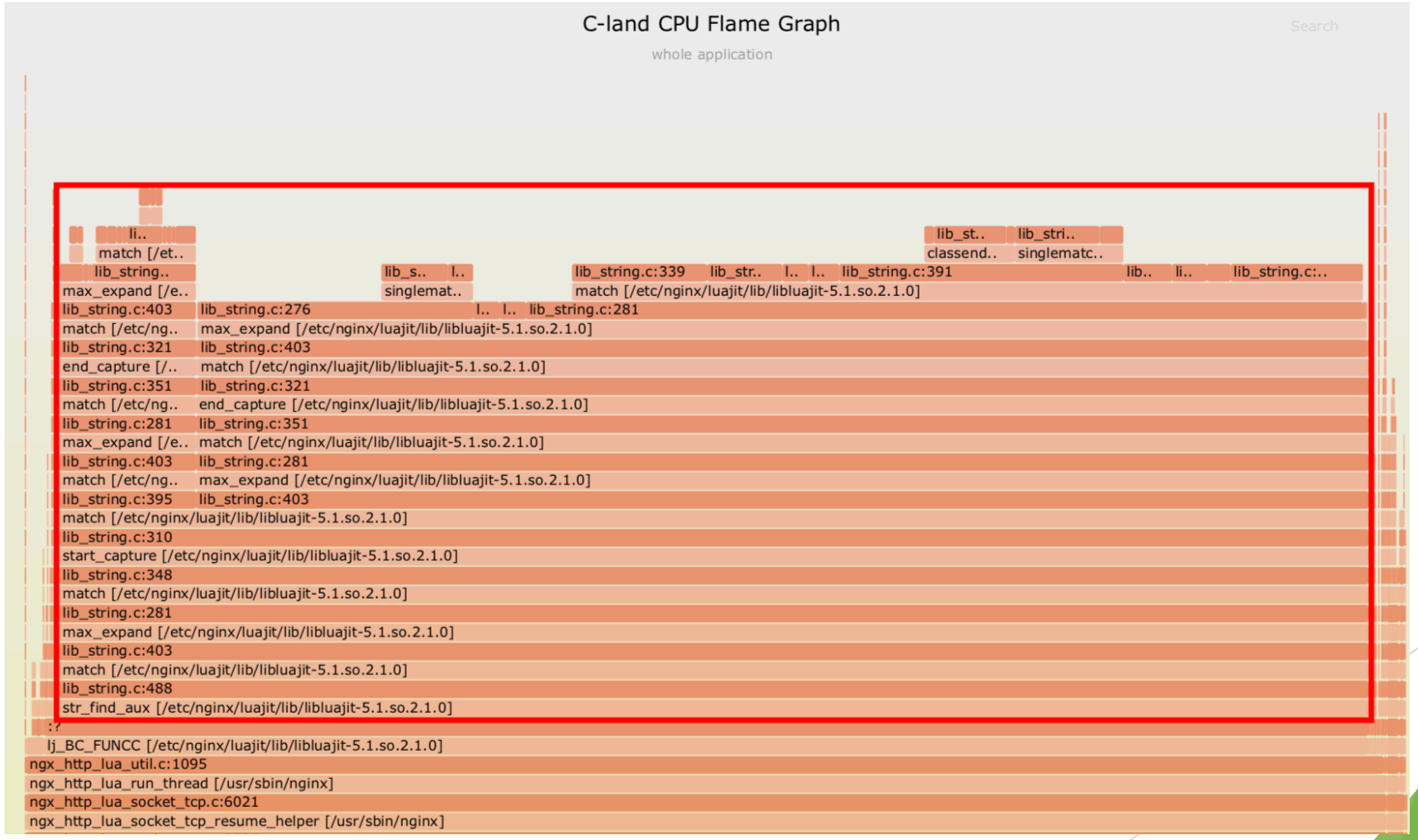
The customer's nginx processes were growing fast



Lua-cjson's cjson.new() is leaking memory! Never use it!



Bad regex in string.match can eat a lot of CPU time

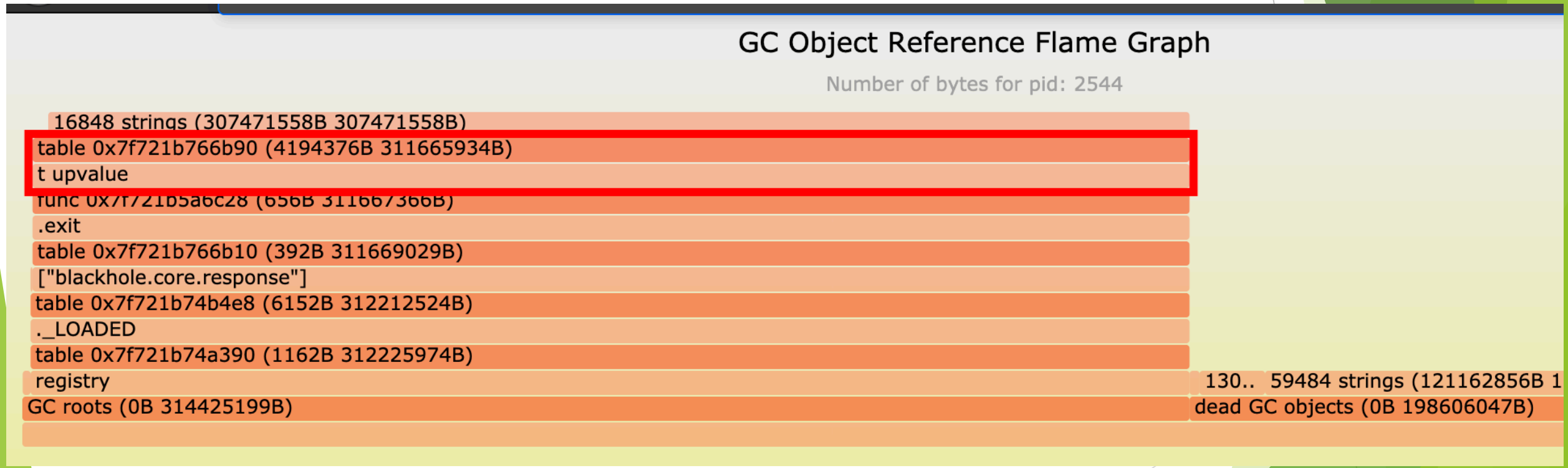


The slowest regex patterns

a single match took almost 1 second! Wow. Note the .*.* part.

```
match: 992786907 ns: pat: <myURI.*(https?://[%./%w%d-_#\?].*).*]]></myURI>, subj len: 5595
match: 986029780 ns: pat: <myURI.*(https?://[%./%w%d-_#\?].*).*]]>
</myURI>, subj len: 5583
...
match: 991585544 ns: pat: <myURI.*(https?://[%./%w%d-_#\?].*).*]]>
</myURI>, subj len: 5594
match: 990244260 ns: pat: <myURI.*(https?://[%./%w%d-_#\?].*).*]]>
</myURI>, subj len: 5582
```

Memory leak in the customer's Lua code



Write your own dynamic tracing analyzers in OpenResty XRay via a superset of Lua language

Ylang code | Ylua code *

Auto-saved a minute ago Restore

```
1 probe process.begin
2     print(dump(package.loaded.jit.version))
3     for k, v in pairs(package.loaded) do
4         print("module ", k)
5     end
6     exit()
7 end
8
```

Use a superset of the C language to write custom analyzers

Ylang code | Ylua code *

Auto-saved an hour ago Restore Vim mode ON

```
1 _probe ngx_http_finalize_request(ngx_http_request_t *r,
  ngx_int_t rc) {
2     printf("%d: uri: %.*s\n", (int) rc, (int)
  r->uri.len, r->uri.data);
3     _exit();
4 }
5
6 _probe _begin {
7     _warn("Start tracing...\n");
8 }
9
```

Lua-resty-jsonb for OpenResty XRay customers

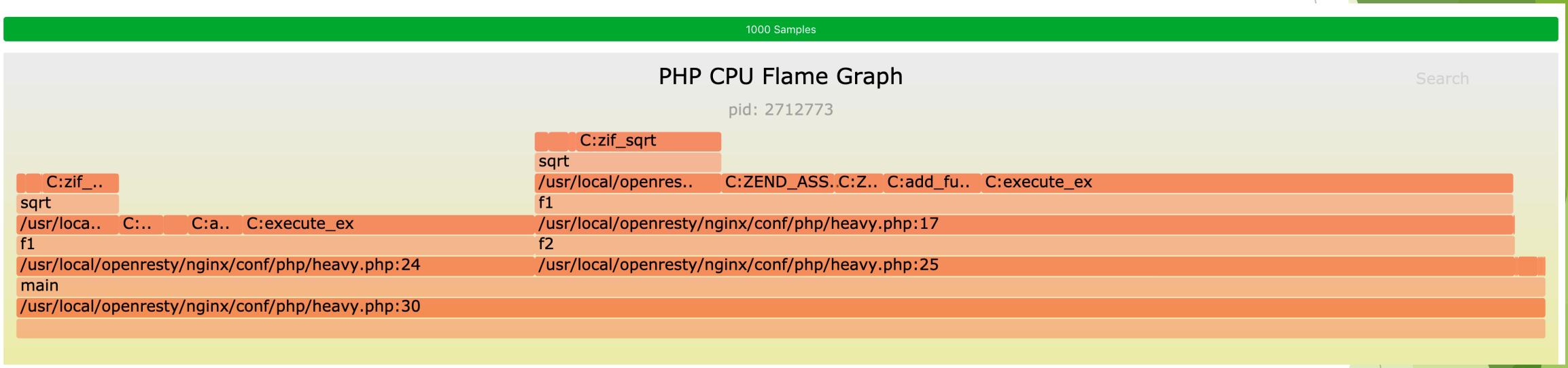
```lua
local libjsonb = require "resty.jsonb"

--[[
local f = assert(io.open("a.json", "r"))
local json = f:read("*a")
f:close();

local jsonb, root_val_ref, err = libjsonb.compile(json)

f = assert(io.open("a.jsonb", "wb"))
assert(f:write(jsonb))
f:close()
]]

local f = assert(io.open("a.jsonb", "rb"))
local jsonb = f:read("*a")
f:close()

local val

ngx.update_time()
local begin = ngx.now()

for i = 1, 100 do
    local val_ref, err = libjsonb.path(jsonb, libjsonb.root_val_ref, 'data', 7,
                                       "timestamp")
    if not val_ref then
        ngx.say("no .data[7].timestamp: ", err)
        return
    end

    val = assert(libjsonb.deref(jsonb, val_ref))
end

ngx.update_time()
local elapsed = ngx.now() - begin
ngx.say(string.format("elapsed: %.03f sec", elapsed))

print("value: ", val)
```

Lua-resty-jsonb is much much faster than lua-cjson

- $ resty cjson-test.lua
- elapsed: 0.133 sec
- value: 1600075810

- $ resty jsonb-test.lua
- elapsed: 0.001 sec
- value: 1600075810

PHP CPU Flame Graphs



More software stacks are coming… Python, Ruby, Perl, NodeJS, Go, Java, MySQL, PostgresSQL, Android…

Read more about OpenResty XRay on our blog site

- https://blog.openresty.com.cn/

Try out OpenResty XRay for free!

- Send a request via
- https://openresty.com.cn/cn/xray/request-demo/

We’e hiring !

- Please send your resume to talents@openresty.com
- Thank you!

Thank you!

- Join our OpenResty open source community!