Implementação de uma biblioteca cliente HTTP/2 multiplataforma em Lua

Murillo Rodrigues de Paula
Bruno Oliveira Silvestre

Roteiro

- Introdução
- HTTP/2
- Lua
- Implementação
- Testes
- Considerações Finais
- Trabalhos Futuros

Introdução

- Páginas Web possuem tamanho cada vez maior.
- Abordagem do HTTP/1.1:
  - Abrir uma conexão TCP por requisição.
  - Efeito colateral: cabeçalhos redundantes.
- Nova abordagem: HTTP/2.
  - Resolve problemas do HTTP/1.1.
  - Mais funcionalidades.
  - Resultado: melhor desempenho às aplicações.

Biblioteca Cliente HTTP/2

- Permite a escrita de clientes HTTP/2 genéricos.
- Exemplo: Python, Node.js, Go, Ruby, C/C++, etc.
- Para Lua, existe apenas uma.
  - Não é multiplataforma.
- Objetivo: implementar, em Lua, uma biblioteca cliente HTTP/2 que é multiplataforma.

HTTP/2

SPDY ("SPeeDY")

- Protocolo experimental criado pela Google.
  - Parte da iniciativa *Let's Make the Web Faster*.
- Objetivos: -50% de tempo para carregar páginas Web e manter compatibilidade com o HTTP/1.1.

SPDY ("SPeeDY")

- Resultados:
  - +60% sobre uma conexão TCP.
  - +55% sobre uma conexão TLS.
- Foi implementado rapidamente pela indústria:
  - Navegadores: Chrome, Firefox, IE, Opera, etc.
  - Sites: Google, Facebook, Twitter, Yahoo, etc.
  - Servidores: Apache e Nginx.

SPDY ("SPeeDY")

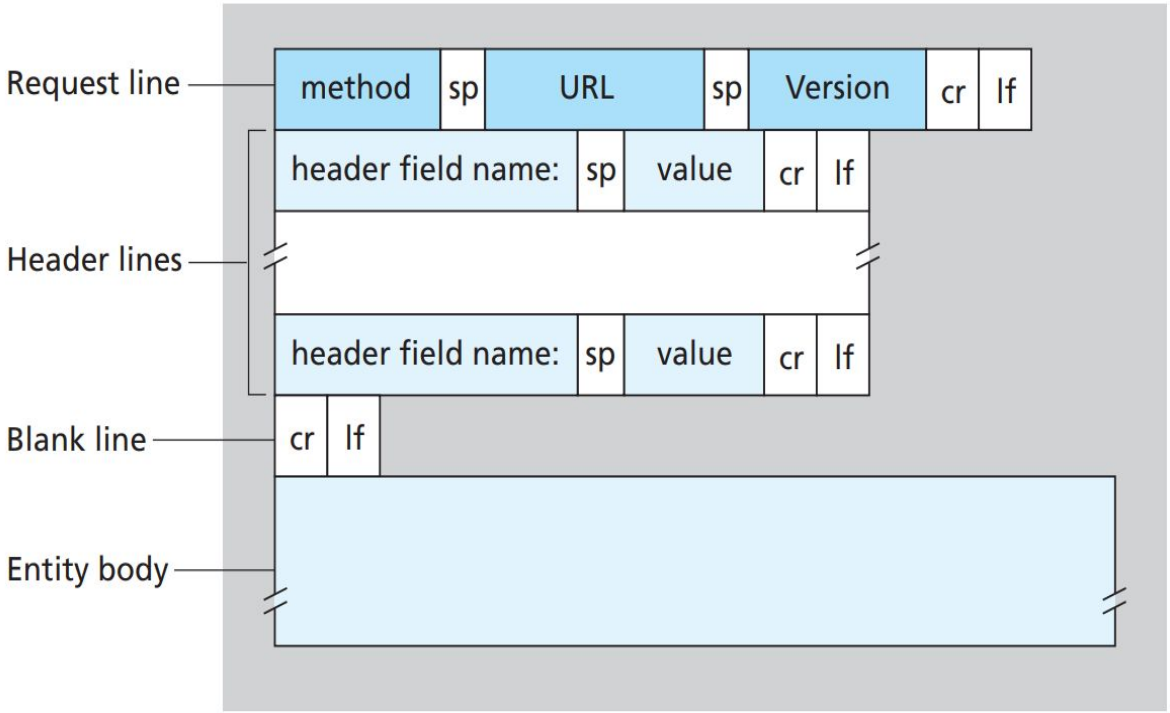
- A IETF padronizou o HTTP/2 a partir do SPDY.
- Decisões de projeto do SPDY:
  - **Uma única conexão TCP para requisições.**
  - **Compressão de cabeçalhos.**
  - Interoperabilidade com o HTTP/1.1.
  - Servidores fazem "requisições".
  - Prioridades de requisições.

Enquadramento de Mensagens

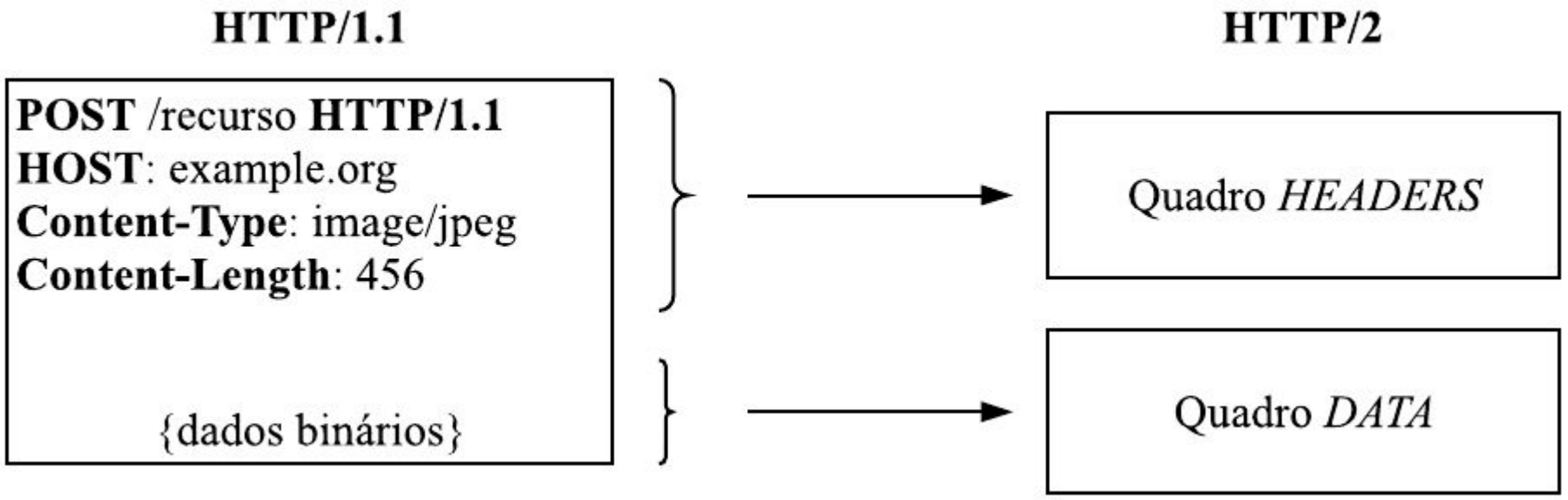
HTTP/1.1

POST /recurso **HTTP/1.1**
HOST: example.org
Content-Type: image/jpeg
Content-Length: 456

{dados binários}

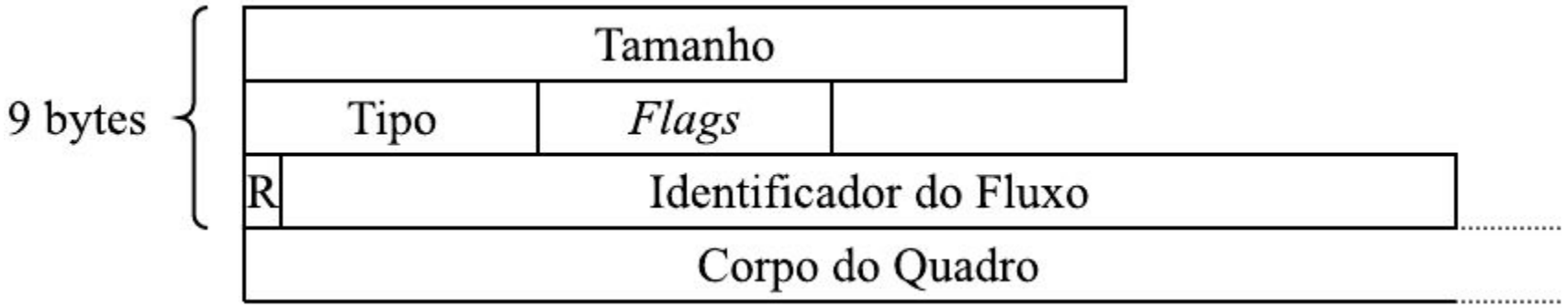


Enquadramento de Mensagens



Enquadramento de Mensagens

- Quadro: a unidade de comunicação do HTTP/2.

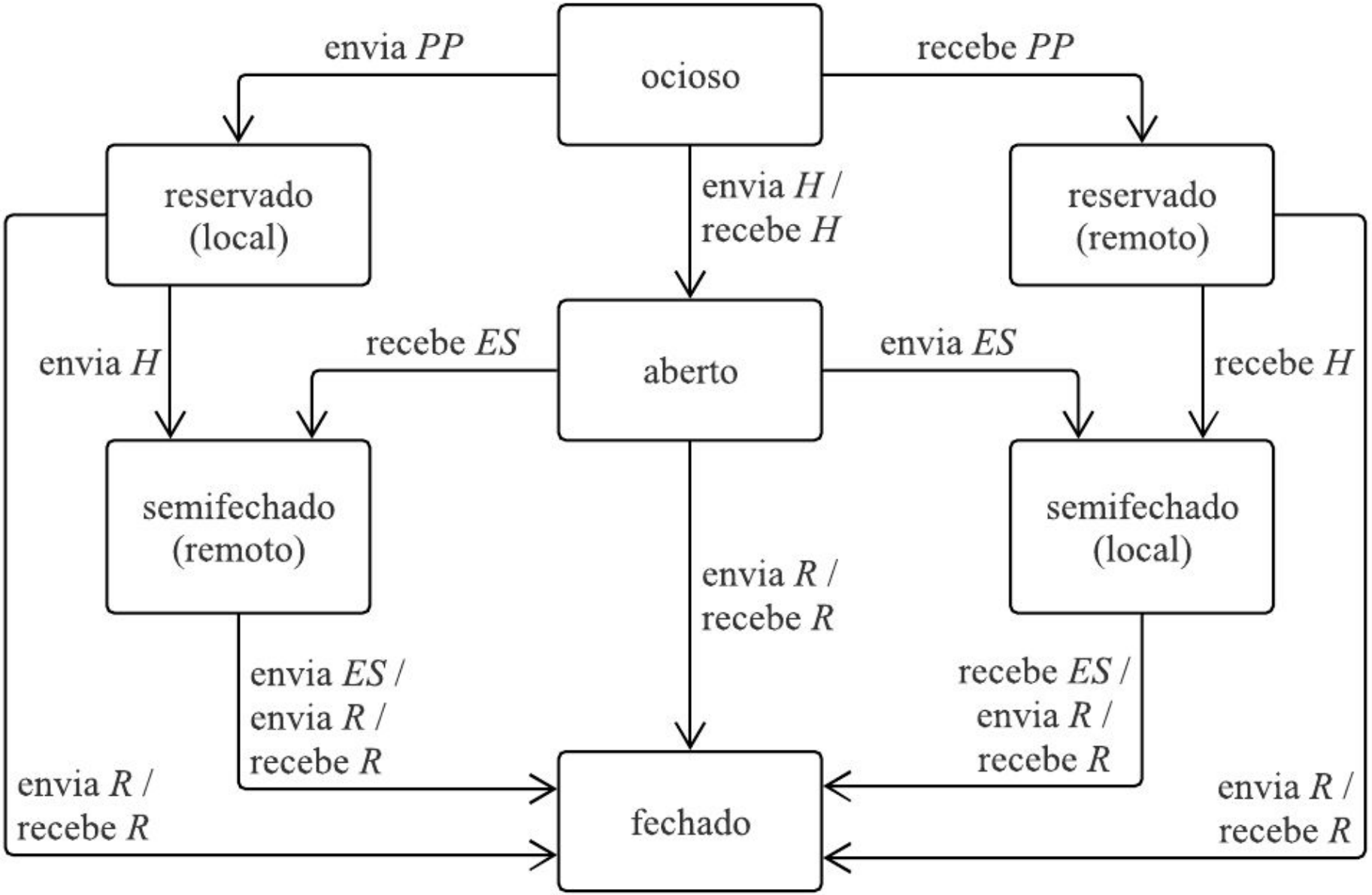


Enquadramento de Mensagens

- Quadro *DATA*:

Fluxo de Mensagens

- Fluxo: sequência de quadros na conexão TCP.
- No contexto de clientes e servidores:
  - Independente.
  - Bidirecional.
  - Multiplexação de fluxos.
  - Requisição/resposta é associada a um fluxo.
- Como conhecemos a estrutura de um quadro:
  - O próximo quadro do fluxo é conhecido.
  - O corpo do quadro é facilmente identificado.


ocioso
envia PP
recebe PP
reservado (local)
reservado (remoto)
envia H / recebe H
envia H
recebe ES
aberto
envia ES
recebe H
semifechado (remoto)
semifechado (local)
envia R / recebe R
envia ES / envia R / recebe R
recebe ES / envia R / recebe R
envia R / recebe R
envia R / recebe R
fechado

Controle de Fluxo

- Efeito colateral da multiplexação de fluxos:
  - Um fluxo pode bloquear outros fluxos.
- Controle de fluxo garante que isso não ocorra.
  - Implementado por *WINDOW_UPDATE*.
  - Baseado em créditos.
  - Qualquer algoritmo pode ser utilizado.

HPACK

- HPACK é um compressor de cabeçalhos.
- Principais elementos:
  - Campos de cabeçalhos.
  - Blocos de cabeçalhos.
  - Tabela estática.
  - Tabela dinâmica.
  - Codificador.
  - Decodificador.

Priorização de Requisições

- No contexto de multiplexação de fluxos:
  - Fluxos podem ser priorizados pelos clientes.
  - Com isso, requisições podem ser priorizadas.
- Método de alocação de recursos.
  - Proporcional em relação a outras requisições.
- Implementado por *HEADERS* ou *PRIORITY*.
- Baseado em dependências e pesos.

Push de Requisição/Resposta

- Uma página Web consiste de vários recursos.
- HTTP/1.1: cliente faz uma requisição por recurso.
- HTTP/2: servidor pode fazer "requisições":
  - *Push* ("forçar") várias respostas.
- Implementado dividindo-o em duas fases:
  1. Fluxo é reservado com *PUSH_PROMISE*.
  2. Servidor força respostas nesse fluxo.

Lua

Características de Lua

- Linguagem de *scripting* multiparadigma.
- Descrição de dados.
- Mundialmente usada:
  - Linguagem de *scripting* mais usada em jogos.
  - Uma das linguagens de *scripting* mais rápidas.
  - Sistemas embarcados.
  - Várias outras aplicações industriais.

Características de Lua

- Algumas semelhanças com outras linguagens:
  - Estruturas de dados dinâmicas.
  - *Garbage collector*.
- Diferenças:
  - Simplicidade.
  - Extensibilidade.
  - Eficiência.
  - Portabilidade.

```
function fact (n)
  if n == 0 then
    return 1
  else
    return n * fact(n-1)
  end
end
```

Co-rotinas

- Funcionam como *multithreading* cooperativa.
- Valores de primeira classe.
- Principais funções:
  - `coroutine.create`
  - `coroutine.yield`
  - `coroutine.resume`
- Muitos `yield` e `resume` são difíceis de lidar.
- Uma co-rotina pode travar outras co-rotinas.

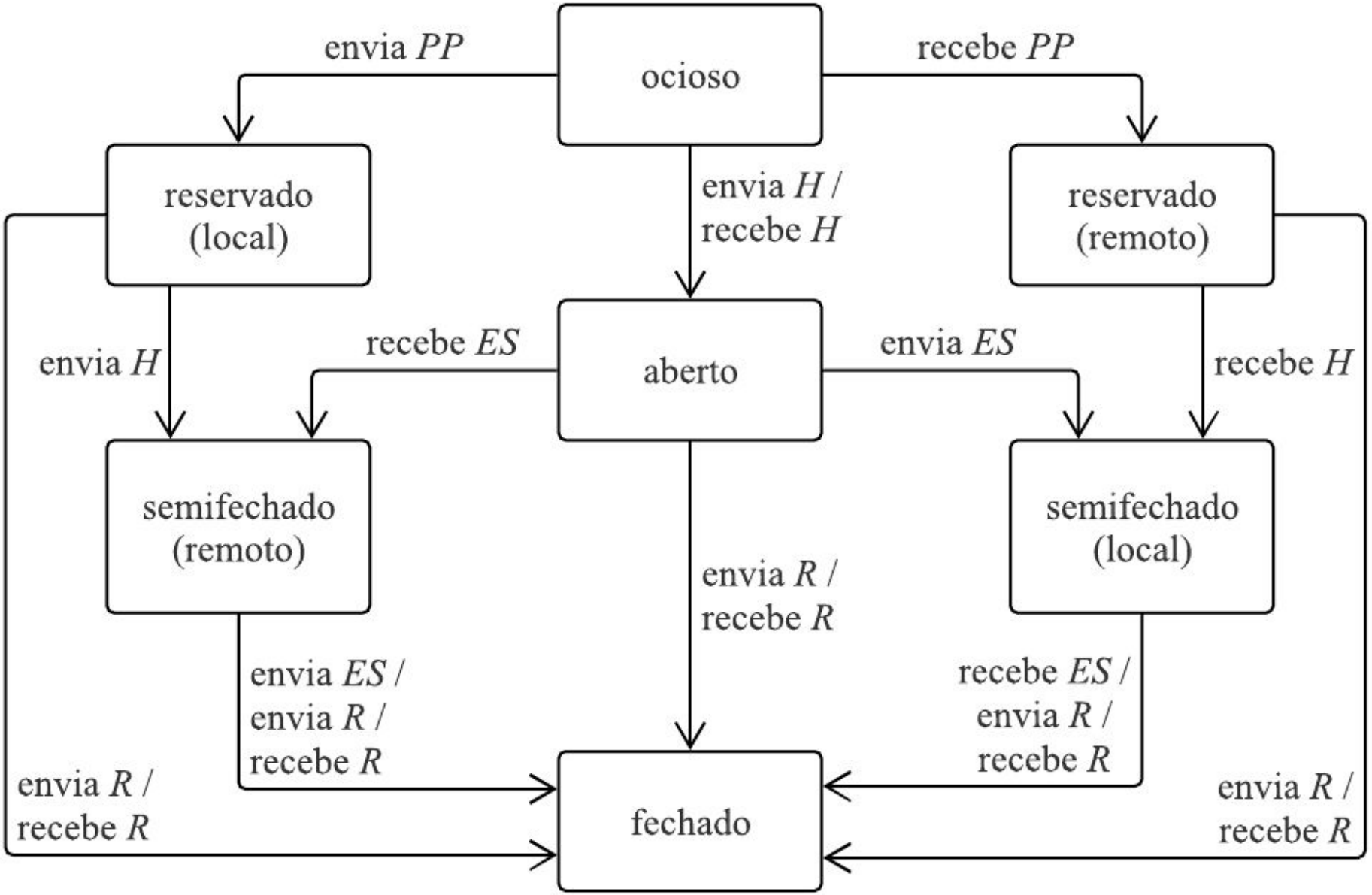
Biblioteca Copas

- Escalonador de co-rotinas.
- Seu foco é na criação de clientes e servidores.
- Funções assíncronas:
  - `copas.send` e `copas.receive`
- Funções para escalonar co-rotinas:
  - `copas.sleep`, `copas.wakeup`, `copas.addthread` e `copas.loop`

Implementação

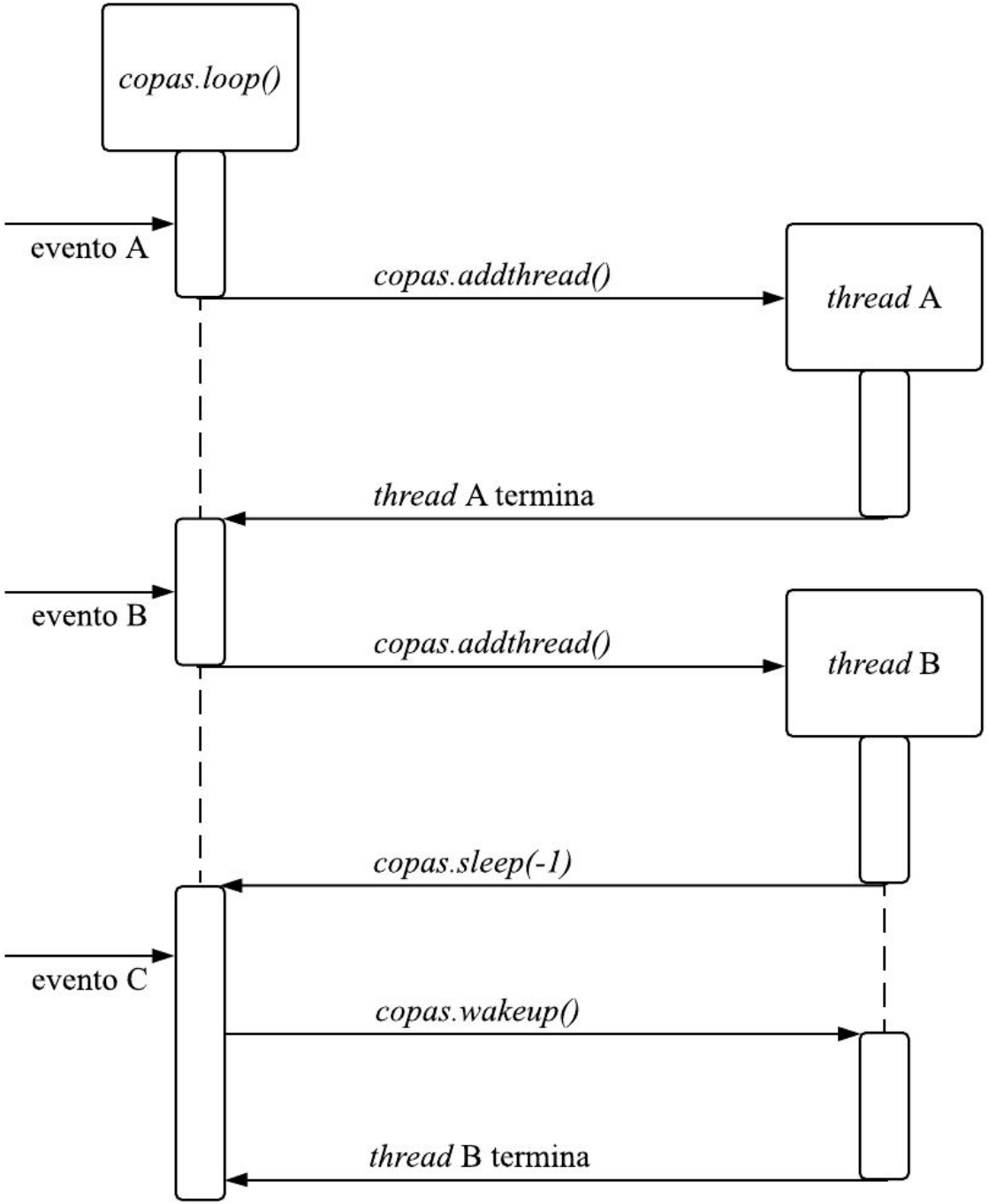
Modelo de Eventos

- Ciclo de vida de um fluxo HTTP/2:
  - Modelado como uma máquina de estados.
  - Programação orientada a eventos.


ocioso
envia PP
recebe PP
reservado (local)
reservado (remoto)
envia H / recebe H
aberto
recebe ES
envia ES
envia H
recebe H
semifechado (remoto)
semifechado (local)
envia R / recebe R
envia ES / envia R / recebe R
recebe ES / envia R / recebe R
envia R / recebe R
envia R / recebe R
fechado

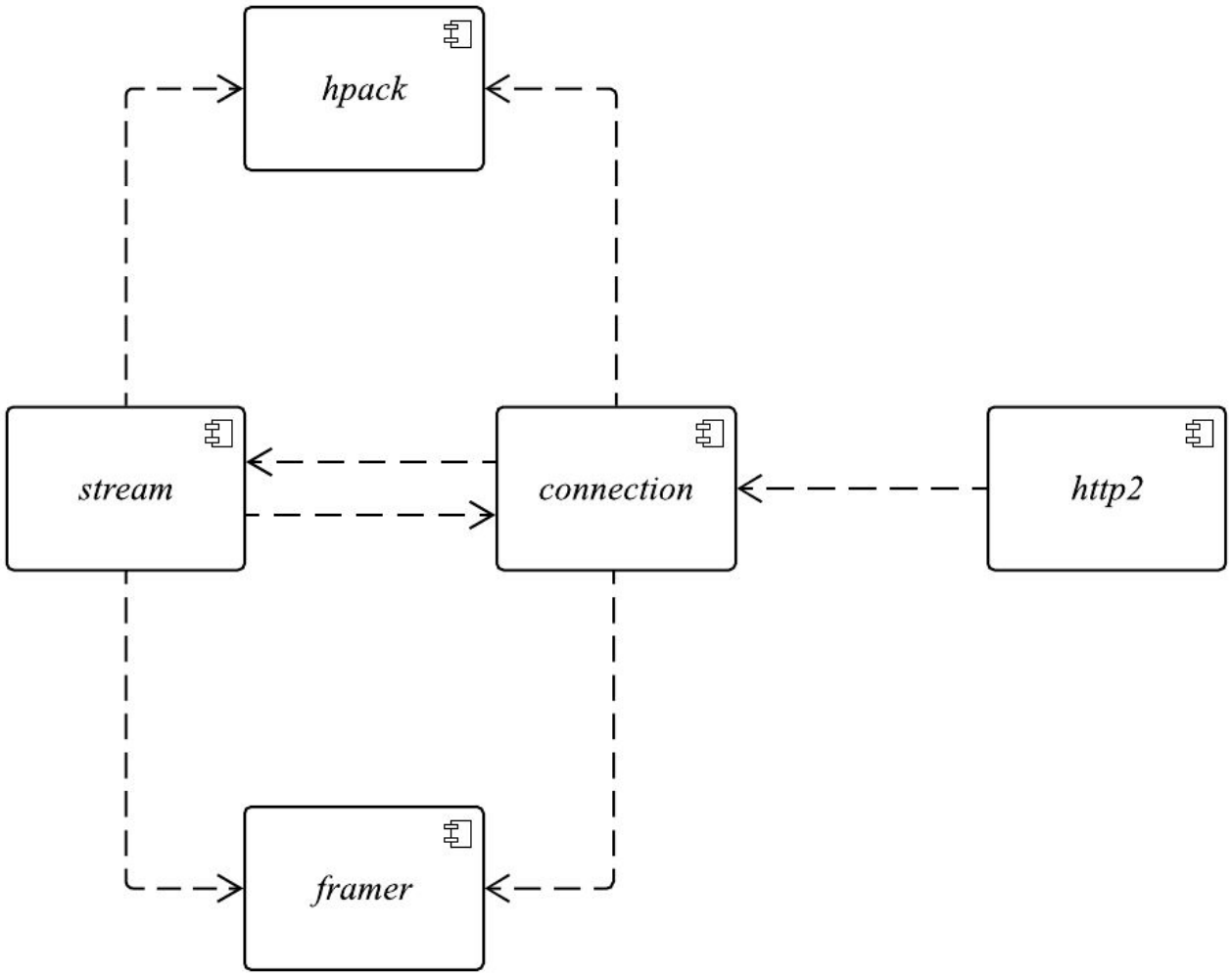
Modelo de Eventos

- Optamos por implementar eventos em Copas.
  - API orientada a eventos.
  - Ciclo de vida de um fluxos em eventos.


copas.loop()
evento A
copas.addthread()
thread A
thread A termina
evento B
copas.addthread()
thread B
copas.sleep(-1)
evento C
copas.wakeup()
thread B termina

Modelo de Concorrência

- Objetivo: implementar multiplexação de fluxos.
- "Escalonador de fluxos".
- Copas atende bem a esses requisitos:
  - *Multithreading* cooperativa de Lua.
  - Operações assíncronas de rede.


hpack
stream
connection
http2
framer

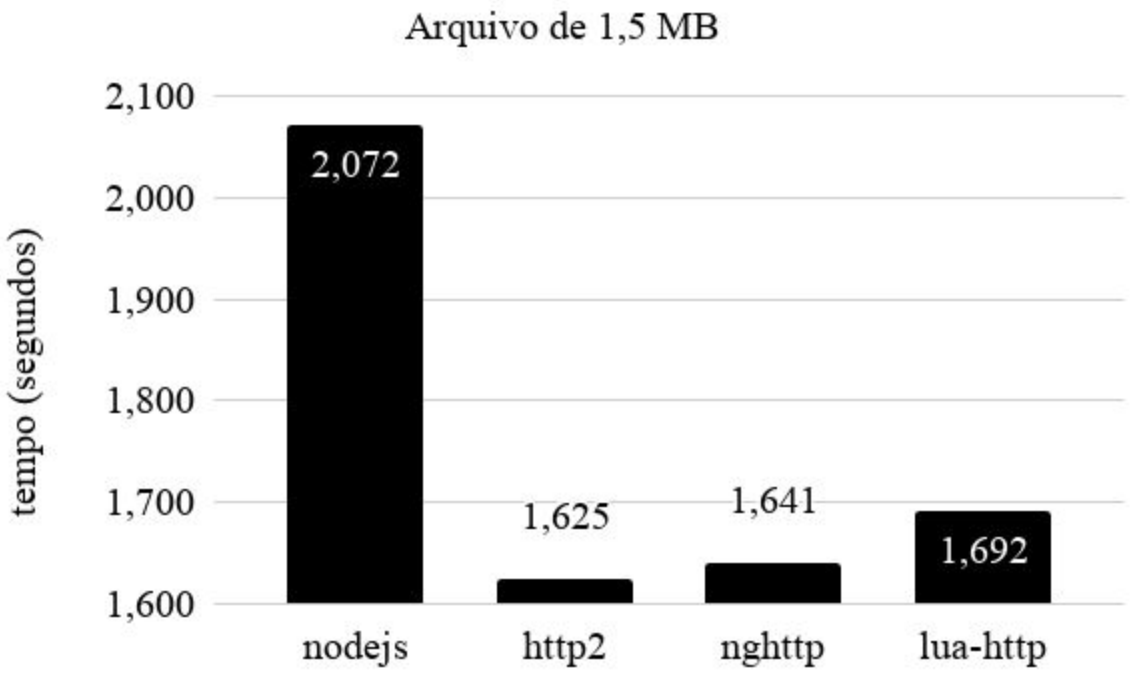
```
local http2 = require "http2"

url = "https://www.google.com.br/"

http2.on_connect(url, function(session)
  local req = session.request()

  req.on_response(function(headers)
        ...
  end)

  r.on_data(function(data)
        ...
  end)
end)
```

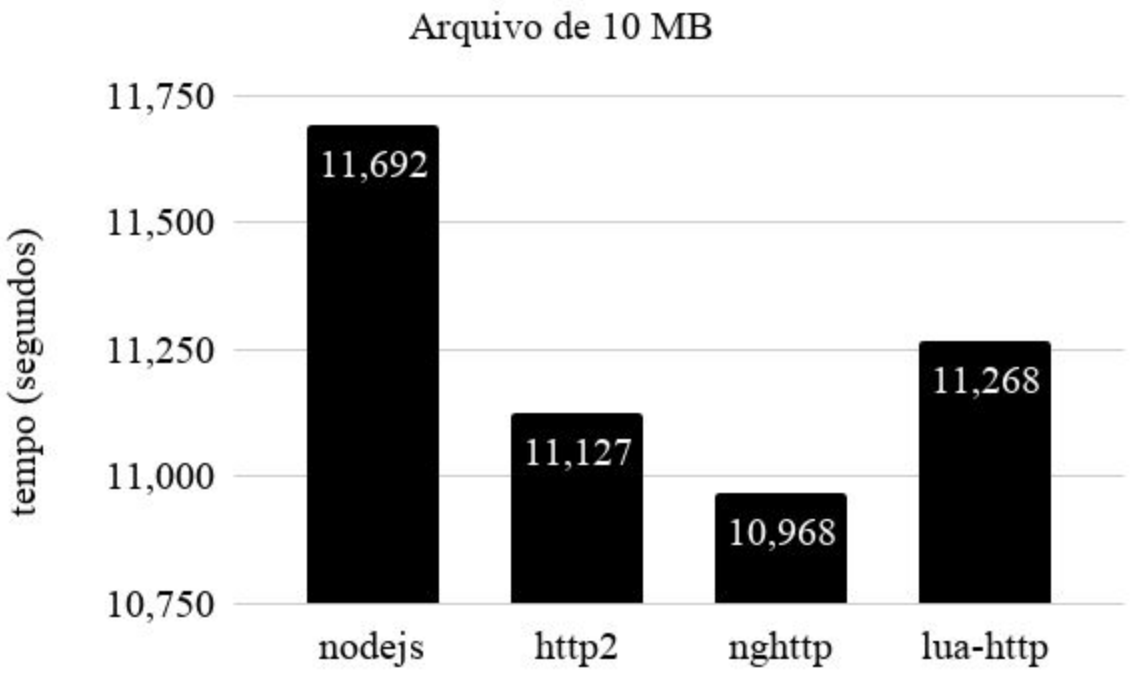
```
local function dispatcher()
  while true do
    s = conn:parse_stream()

    if s.state == "open" then
      copas.wakeup(on_response[s.id])
    end

    if stream.state == "closed" then
      copas.wakeup(on_data[s.id])
      break
    end
  end
end
```

Testes

Testes

- Os seguintes clientes HTTP/2 foram testados:
  - lua-http (em Lua).
  - nghttp (em C/C++).
  - nodejs (em Node.js).
  - http2 (na presente biblioteca).
- Apenas um servidor HTTP/2 foi utilizado: nghttpd.
- Fizemos 3 testes:
  - Dez execuções cada.
  - Dez requisições na mesma conexão TCP.


Arquivo de 1,5 MB
tempo (segundos)
2,100
2,000
1,900
1,800
1,700
1,600
2,072
1,625
1,641
1,692
nodejs
http2
nghttp
lua-http


Arquivo de 10 MB
tempo (segundos)
11,750
11,500
11,250
11,000
10,750
11,692
11,127
10,968
11,268
nodejs
http2
nghttp
lua-http


Arquivo de 100 MB
tempo (segundos)
96,000
94,000
92,000
90,000
88,000
89,158
88,679
88,670
94,396
nodejs
http2
nghttp
lua-http

Considerações Finais

Considerações Finais

- Programação orientada a eventos.
  - Implementação do ciclo de vida um fluxo.
  - API baseada em *callbacks*.
- Programação concorrente.
  - Multiplexação de fluxos (requisição/resposta).
- Simples API para clientes HTTP/2 genéricos.
- **Multiplataforma.**

Considerações Finais

- O que está sendo suportado:
  - Requisições GET e POST simples.
  - Várias requisições na mesma conexão TCP.
  - Principais quadros, com exceção de *PUSH_PROMISE*, *PING e PRIORITY*.
  - Estados de um fluxo: ocioso, aberto, semifechado (remoto) e fechado.

Trabalhos Futuros

Trabalhos Futuros

- Estender a biblioteca:
  - Priorização de fluxos.
  - *Push* de requisição/resposta.
- Gerenciamento de *cookies*.
- Cache.
- Tratamento de erros.
- Reavaliar Copas.

Implementação de uma biblioteca cliente HTTP/2 multiplataforma em Lua

Murillo Rodrigues de Paula
Bruno Oliveira Silvestre