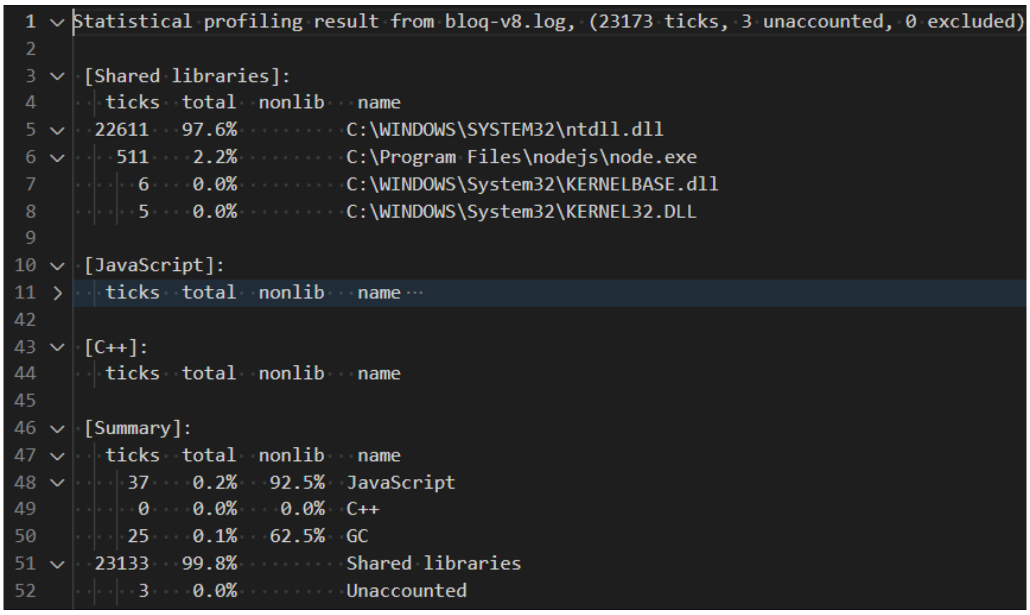
ANÁLISIS COMPLETO DE PERFORMANCE

A) Vamos a trabajar sobre la ruta '/info', en modo fork, agregando ó extrayendo un console.log de la información colectada antes de devolverla al cliente. Además desactivaremos el child_process de la ruta '/randoms'.

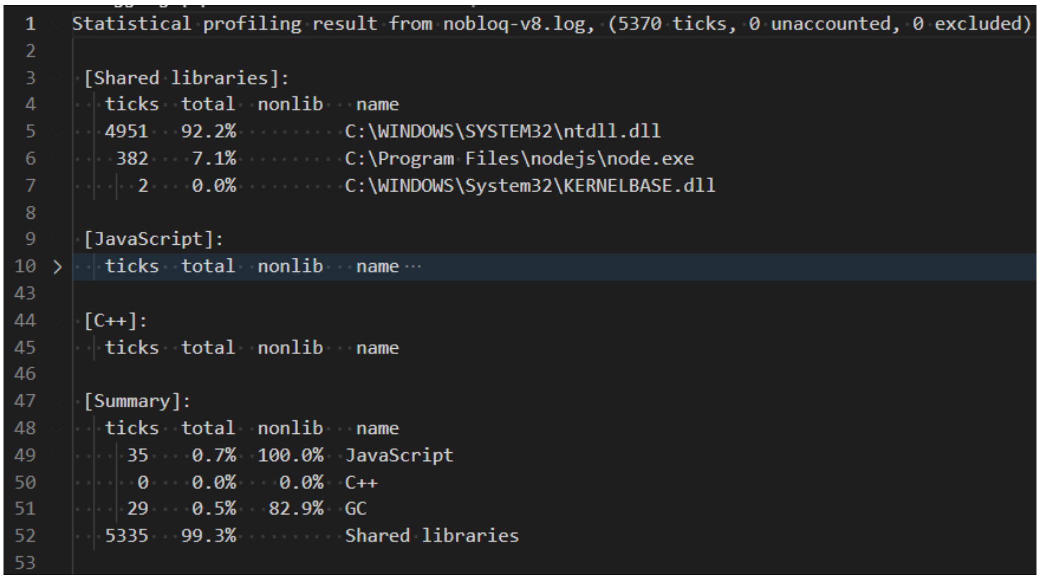
Para ambas condiciones (con o sin console.log) en la ruta '/info' OBTENER:

a. El perfilamiento del servidor, realizando el test con --prof de node.js. Analizar los resultados obtenidos luego de procesarlos con --prof-process.
Utilizaremos como test de carga Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una.

b. Resultados Prof



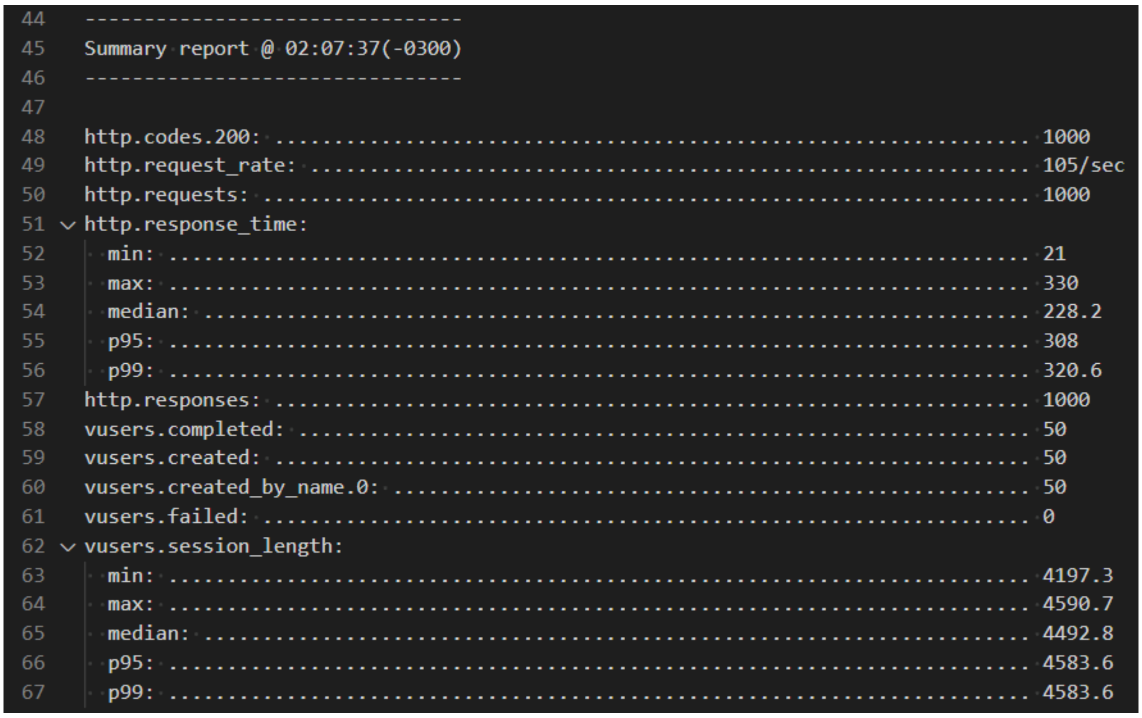
```
Statistical profiling result from bloq-v8.log, (23173 ticks, 3 unaccounted, 0 excluded)

 [Shared libraries]:
   ticks  total  nonlib   name
  22611   97.6%          C:\WINDOWS\SYSTEM32\ntdll.dll
    511    2.2%          C:\Program Files\nodejs\node.exe
      6    0.0%          C:\WINDOWS\System32\KERNELBASE.dll
      5    0.0%          C:\WINDOWS\System32\KERNEL32.DLL

 [JavaScript]:
   ticks  total  nonlib   name ···

 [C++]:
   ticks  total  nonlib   name

 [Summary]:
   ticks  total  nonlib   name
     37    0.2%   92.5%  JavaScript
      0    0.0%    0.0%  C++
     25    0.1%   62.5%  GC
  23133   99.8%          Shared libraries
      3    0.0%          Unaccounted
```



```
Statistical profiling result from nobloq-v8.log, (5370 ticks, 0 unaccounted, 0 excluded)

 [Shared libraries]:
   ticks  total  nonlib   name
   4951   92.2%          C:\WINDOWS\SYSTEM32\ntdll.dll
    382    7.1%          C:\Program Files\nodejs\node.exe
      2    0.0%          C:\WINDOWS\System32\KERNELBASE.dll

 [JavaScript]:
   ticks  total  nonlib   name ···

 [C++]:
   ticks  total  nonlib   name

 [Summary]:
   ticks  total  nonlib   name
     35    0.7%  100.0%  JavaScript
      0    0.0%    0.0%  C++
     29    0.5%   82.9%  GC
   5335   99.3%          Shared libraries

```

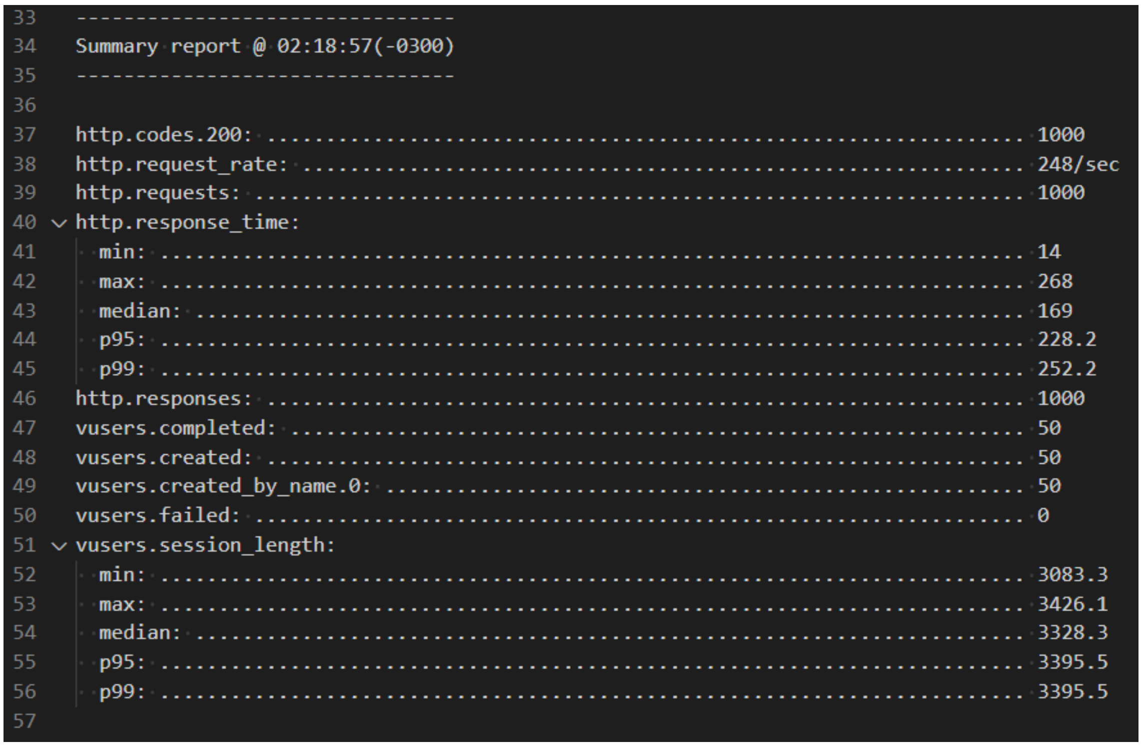
c. Resultado Artillery

Con console.log()



```
--------------------------------
Summary report @ 02:07:37(-0300)
--------------------------------

http.codes.200: ............................................................ 1000
http.request_rate: ......................................................... 105/sec
http.requests: ............................................................. 1000
http.response_time:
  min: ..................................................................... 21
  max: ..................................................................... 330
  median: .................................................................. 228.2
  p95: ..................................................................... 308
  p99: ..................................................................... 320.6
http.responses: ............................................................ 1000
vusers.completed: .......................................................... 50
vusers.created: ............................................................ 50
vusers.created_by_name.0: .................................................. 50
vusers.failed: ............................................................. 0
vusers.session_length:
  min: ..................................................................... 4197.3
  max: ..................................................................... 4590.7
  median: .................................................................. 4492.8
  p95: ..................................................................... 4583.6
  p99: ..................................................................... 4583.6
```

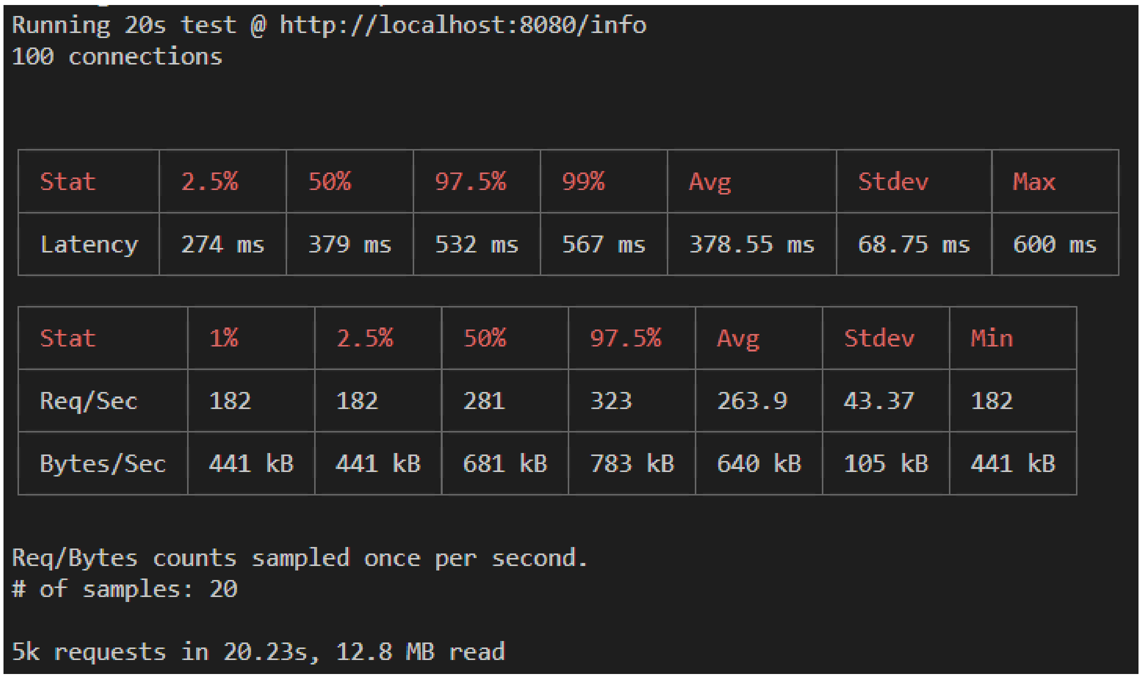
Sin console.log()



```
--------------------------------
Summary report @ 02:18:57(-0300)
--------------------------------

http.codes.200: ............................................................ 1000
http.request_rate: ......................................................... 248/sec
http.requests: ............................................................. 1000
http.response_time:
  min: ..................................................................... 14
  max: ..................................................................... 268
  median: .................................................................. 169
  p95: ..................................................................... 228.2
  p99: ..................................................................... 252.2
http.responses: ............................................................ 1000
vusers.completed: .......................................................... 50
vusers.created: ............................................................ 50
vusers.created_by_name.0: .................................................. 50
vusers.failed: ............................................................. 0
vusers.session_length:
  min: ..................................................................... 3083.3
  max: ..................................................................... 3426.1
  median: .................................................................. 3328.3
  p95: ..................................................................... 3395.5
  p99: ..................................................................... 3395.5
```

B) Luego utilizaremos Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos. Extraer un reporte con los resultados (puede ser un print screen de la consola).

a. Resumen Autocannon No Bloqueante



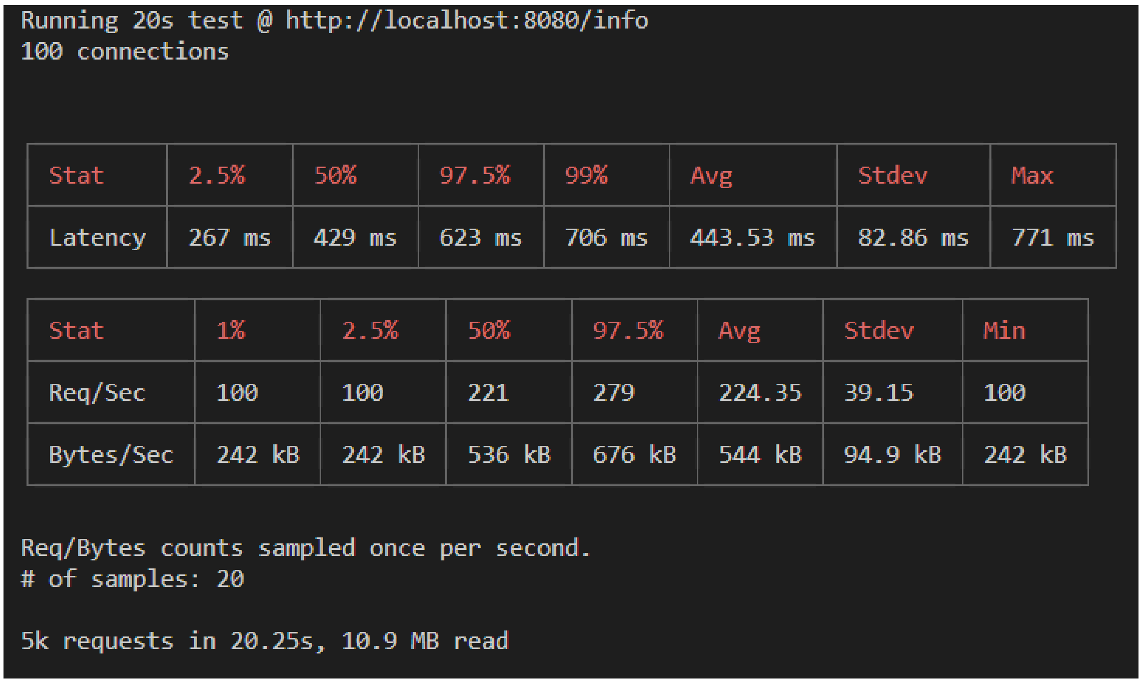
```
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	274 ms	379 ms	532 ms	567 ms	378.55 ms	68.75 ms	600 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	182	182	281	323	263.9	43.37	182
Bytes/Sec	441 kB	441 kB	681 kB	783 kB	640 kB	105 kB	441 kB

```
Req/Bytes counts sampled once per second.
# of samples: 20

5k requests in 20.23s, 12.8 MB read
```

b. Resumen Autocannon Bloqueante



```
Running 20s test @ http://localhost:8080/info
100 connections
```

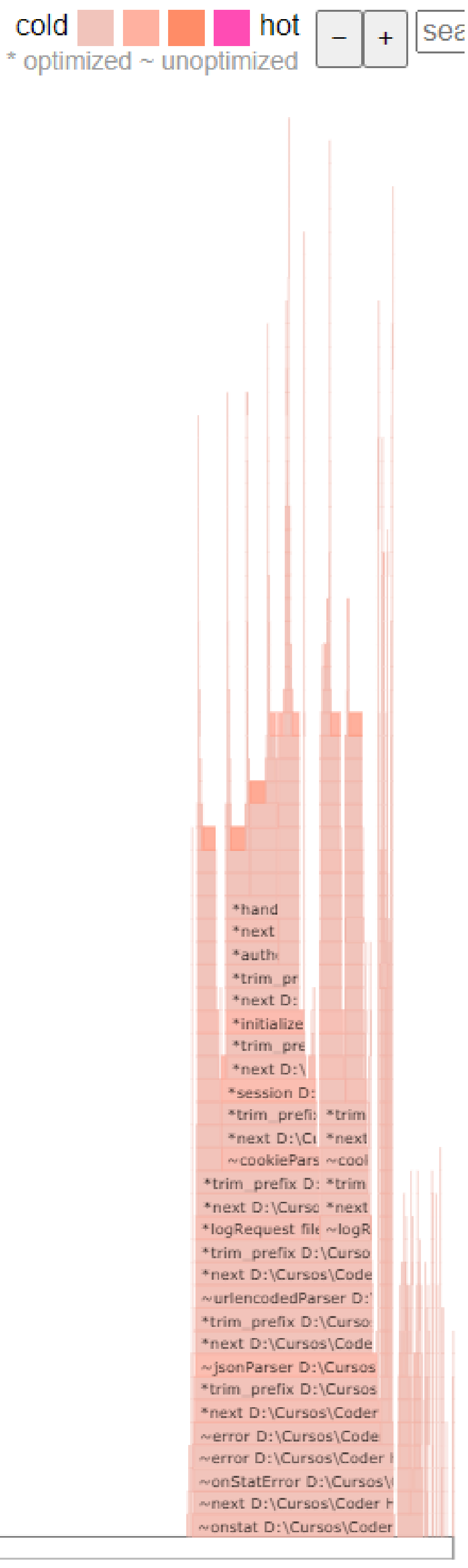
Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	267 ms	429 ms	623 ms	706 ms	443.53 ms	82.86 ms	771 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	100	100	221	279	224.35	39.15	100
Bytes/Sec	242 kB	242 kB	536 kB	676 kB	544 kB	94.9 kB	242 kB

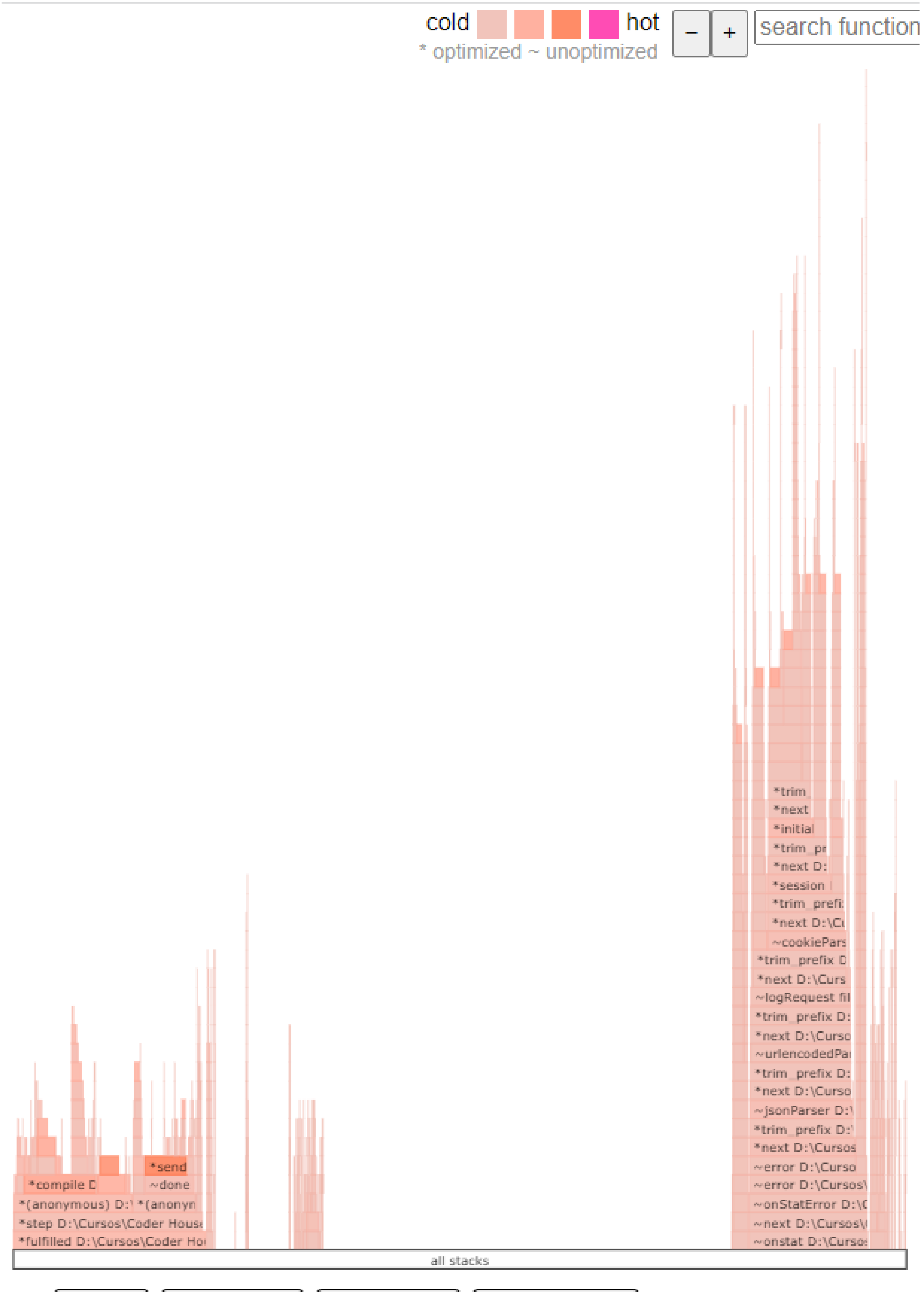
```
Req/Bytes counts sampled once per second.
# of samples: 20

5k requests in 20.25s, 10.9 MB read
```

C) Flame Graph 0x

a. Con console.log()



b. Sin console.log()


cold
hot
* optimized ~ unoptimized
search function
*trim
*next
*initial
*trim_pr
*next D:
*session
*trim_prefi
*next D:\Cu
~cookiePars
*trim_prefix D
*next D:\Curs
~logRequest fil
*trim_prefix D:
*next D:\Curso
~urlencodedPa
*trim_prefix D:
*next D:\Curso
~jsonParser D:\
*trim_prefix D:\
*next D:\Cursos
~error D:\Curso
~error D:\Cursos\
~onStatError D:\C
~next D:\Cursos\
~onstat D:\Cursos
*send
*compile D
~done
*(anonymous) D:\
*(anonym
*step D:\Cursos\Coder House
*fulfilled D:\Cursos\Coder Ho
all stacks