

MODO INSPECT

Autocannon consola: 100 conexiones en un tiempo de 20 segundos

```
PS C:\Users\nico_\OneDrive\Documentos\GitHub\CoderHouse\14 - Decimo cuarto desafio> node src/autocannon/benchmark.js
Running 20s test @ http://localhost:8080/info
100 connections
```

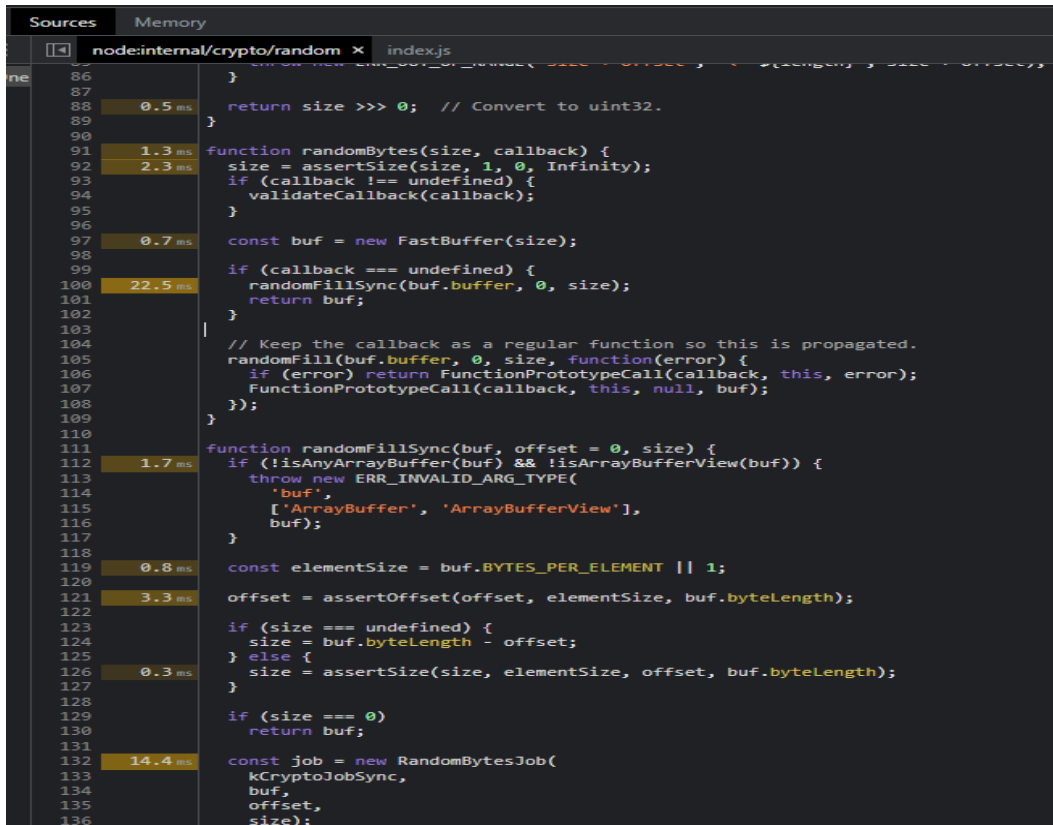
Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	193 ms	213 ms	267 ms	282 ms	215.15 ms	19.93 ms	386 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	402	402	460	501	461.75	29.16	402
Bytes/Sec	179 kB	179 kB	204 kB	222 kB	205 kB	12.9 kB	178 kB

función run

CPU PROFILES		Source	Time	Percentage	Destination
Profile 1	Save		98173 ms	51.89 %	11900.3 ms 62.90 %
		consoleCall			
		writeUtf8String	16422 ms	8.68 %	1642.2 ms 8.68 %
		writev	365.1 ms	1.93 %	365.1 ms 1.93 %
		(program)	359.4 ms	1.90 %	359.4 ms 1.90 %
		(garbage collector)	244.1 ms	1.29 %	244.1 ms 1.29 %
		hash	169.7 ms	0.90 %	372.1 ms 1.97 %
		next	165.0 ms	0.87 %	73186.0 ms 915.42 %
		getColorDepth	146.7 ms	0.78 %	146.7 ms 0.78 %
		nextTick	144.3 ms	0.76 %	236.1 ms 1.25 %
		Hash	131.9 ms	0.70 %	131.9 ms 0.70 %
		session	126.6 ms	0.67 %	16247.0 ms 85.88 %
		store.generate	117.7 ms	0.62 %	445.1 ms 2.35 %
		parse	114.2 ms	0.60 %	127.2 ms 0.67 %
		send	113.4 ms	0.60 %	2318.7 ms 12.26 %
		end	104.6 ms	0.55 %	1323.4 ms 7.00 %
		writeHead	101.0 ms	0.53 %	173.7 ms 0.92 %
		handle	98.3 ms	0.52 %	42230.2 ms 751.79 %
		initialize	87.8 ms	0.46 %	16426.7 ms 86.83 %
		asString	86.3 ms	0.46 %	101.8 ms 0.54 %
		json	79.8 ms	0.42 %	2620.0 ms 13.85 %
		memoryUsage	75.8 ms	0.40 %	75.8 ms 0.40 %
		(anonymous)	74.6 ms	0.39 %	15018.1 ms 79.38 %
		run	73.4 ms	0.39 %	73.4 ms 0.39 %
		randomFillSync	73.4 ms	0.39 %	73.4 ms 0.39 %
		stringify	71.9 ms	0.38 %	71.9 ms 0.38 %
		asyncTaskScheduled	68.6 ms	0.36 %	68.6 ms 0.36 %
		expressInit	68.2 ms	0.36 %	492.0 ms 2.60 %
		Hash	67.0 ms	0.35 %	199.2 ms 1.05 %
		resOnFinish	66.3 ms	0.35 %	295.6 ms 1.56 %
		writevGeneric	64.7 ms	0.34 %	506.2 ms 2.68 %

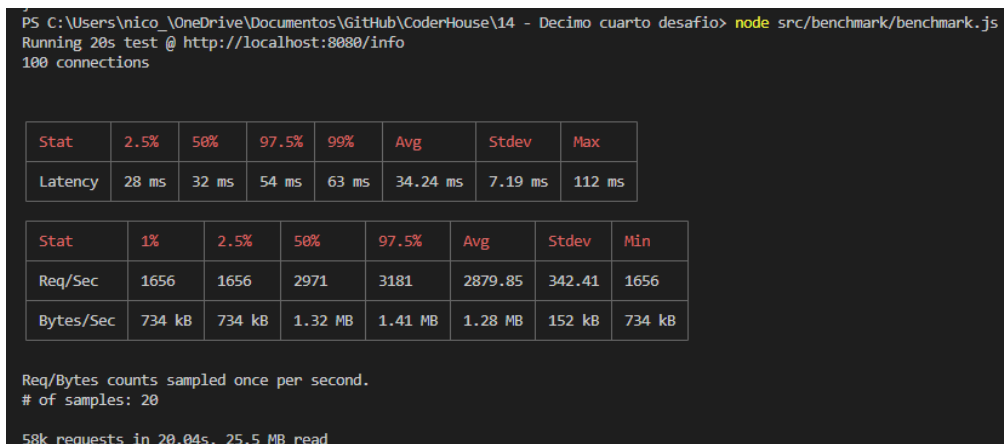
Profile (ruta <http://localhost:8080/info>):



```
86 }
87
88 0.5 ms return size >>> 0; // Convert to uint32.
89 }
90
91 1.3 ms function randomBytes(size, callback) {
92 2.3 ms   size = assertSize(size, 1, 0, Infinity);
93   if (callback !== undefined) {
94     validateCallback(callback);
95   }
96
97 0.7 ms   const buf = new FastBuffer(size);
98
99   if (callback === undefined) {
100 22.5 ms     randomFillSync(buf.buffer, 0, size);
101     return buf;
102   }
103
104   // Keep the callback as a regular function so this is propagated.
105   randomFill(buf.buffer, 0, size, function(error) {
106     if (error) return FunctionPrototypeCall(callback, this, error);
107     FunctionPrototypeCall(callback, this, null, buf);
108   });
109 }
110
111 function randomFillSync(buf, offset = 0, size) {
112 1.7 ms   if (!isArrayBuffer(buf) && !isArrayBufferView(buf)) {
113     throw new ERR_INVALID_ARG_TYPE(
114       'buf',
115       ['ArrayBuffer', 'ArrayBufferView'],
116       buf);
117   }
118
119 0.8 ms   const elementSize = buf.BYTES_PER_ELEMENT || 1;
120
121 3.3 ms   offset = assertOffset(offset, elementSize, buf.byteLength);
122
123   if (size === undefined) {
124     size = buf.byteLength - offset;
125   } else {
126 0.3 ms     size = assertSize(size, elementSize, offset, buf.byteLength);
127   }
128
129   if (size === 0)
130     return buf;
131
132 14.4 ms   const job = new RandomBytesJob(
133     kCryptoJobSync,
134     buf,
135     offset,
136     size);
```

OX

Autocannon consola: 100 conexiones en un tiempo de 20 segundos



```
PS C:\Users\nico\OneDrive\Documentos\GitHub\CoderHouse\14 - Decimo cuarto desafio> node src/benchmark/benchmark.js
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	28 ms	32 ms	54 ms	63 ms	34.24 ms	7.19 ms	112 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1656	1656	2971	3181	2879.85	342.41	1656
Bytes/Sec	734 kB	734 kB	1.32 MB	1.41 MB	1.28 MB	152 kB	734 kB

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

58k requests in 20.04s, 25.5 MB read

Flame Graph (ruta <http://localhost:8080/info>):



Conclusión:

En el modo inspect dentro de la función “run” podemos observar un método (`randomFillSync`). Dentro de este vemos que el mayor tiempo tarda en ejecutar una línea de código es de 22.5 ms y justamente porque es una función sincrónica.

Con 0x: se puede ver en la gráfica que hay picos porque tenemos código que se ejecuta rápidamente (código asíncronico) y por lo tanto la longitud es pequeña.

Por lo tanto, podemos decir que es mejor trabajar de forma asíncronica porque reduce los tiempos de ejecución del código.