HPC Now!

HPCNow!

CTS-2023-0055-TestQRNG

Christian Bustelo, Elisabeth Ortega

[15/09/2023]

XUNTA DE GALICIA

Unión Europea

Unha maneira de facer Europa.

Fondos Europeos

CESGA GALICIA SUPERCOMPUTING CENTER

Despregamento dunha infraestructura baseada en tecnoloxías cuánticas da información que permita impulsar a I+D+i en Galicia.

Apoiar a transición cara a una economía dixital.

Operación financiada pola Unión Europea, a través do FONDO EUROPEO DE DESENVOLVEMENTO REXIONAL (FEDER) como parte da resposta da Unión á pandemia da COVID-19

DATA	AUTHOR	CAMBIOS	VERSION
14/09/2023	E. Ortega	Texto inicial	v0.1
27/09/2023	E. Ortega	Revisión y formato	v1.0

Táboa de contidos

Lista de figuras

Lista de táboas

1. Introducción y alcance del proyecto

En este informe se detallan los resultados y entregabes del proyecto CTS-2023-0055-TestQRNG. Dicho proyecto tiene como objetivo proveer herramientas para el estudio de la estabilidad del generador cuántico de números aleatorios instalado en CESGA.

Las tareas realizadas durante este proyecto son las siguientes:

- Búsqueda de sistemas de chequeo de generadores de números aleatorios.
- Diseñar y montar una solución para ejecutar tests y medir métricas.
- Diseñar y programar un sistema que ejecute los tests y métricas reportados anteriormente, recopile los resultados y los guarde.
- Importar los datos anteriores y graficarlos.

En los siguientes apartados se resumirán las tareas realizadas para concluir cada uno de los puntos anteriores.

1.1. Sistemas de chequeo de generadores de números aleatorios

Existe un gran número de sistemas de chequeo de generadores de números aleatorios y pseudoaleatorios. La referencia más conocida y en la que nos hemos basado para aplicar los tests es Special Publication 800-22. Rev. 1a de NIST (Bassham et al. 2010). Esta publicación incluye una batería de 15 tests en los que se realizan cálculos estadísticos que evaluarán si la secuencia obtenida es realmente aleatoria. Todos ellos aceptan como valor umbral un p-valor mayor o igual de 0.01, lo que asegura que la secuencia analizada se puede considerar aleatoria con una confidencia del 99.9%.

Sin embargo, dicha batería de tests está bajo revisión desde enero del 2022. El equipo detrás de esta publicación está trabajando en proporcionar una nueva herramienta de software, entre otras mejoras (NIST 2022).

1.2. Diseño y montaje de una solución para ejecutar tests y medir métricas

1.2.1. Ejecución de tests

Una vez determinados los tests a ejecutar, se buscaron proyectos de código abierto para posteriormente adaptarlos al sistema instalado en CESGA. Existen diferentes implementaciones de la batería de tests comentada anteriormente en repositorios de código abierto, y se prefirió por usar una versión adaptada en Python del repositorio original (InsaneMonster 2023). En dicha versión se incluyen las siguientes mejoras:

- Tratamiento de los datos de entrada: se puede partir de los datos del dispositivo QRNG (opción por defecto), tomarlos de un archivo del sistema o generar números aleatorios con la función random incluída en Python.

- Diferentes maneras de codificar los datos de entrada: con 8 bits, tal y como estaba codificado en el repositorio, o 32 bits tal y como se reciben los datos del dispositivo QRNG. En el caso de elegir codificación a 8 bits, se ha considerado generarlos de dos formas: mapeando los valores de 32 bits en valores de 8 bits, o solo escogiendo los 8 bits finales de los 32 bits provistos, ya que son los menos significativos en dispositivos generadores de números aleatorios basados en fotónica, y por tanto, los bits con más aleatoriedad (Chen et al. 2019).
- Ejecución de los tests de dos formas diferentes: el script puede ejecutarse de manera que muestre el resultado final por pantalla o exponiendo los resultados en una base de datos de ElasticSearch (Elastic 2023).

```
❯ /bin/python3 /home/eortega/coding/random/NistRng_32bits/qrng_test.py
NOT ELIGIBLE by block size- Binary Matrix Rank(8 vs 38)
NOT ELIGIBLE by bit size- Overlapping Template Matching(8192 vs 1,028,016)
NOT ELIGIBLE by bit size- Maurers Universal(8192 vs 387840)
NOT ELIGIBLE by bit size- Linear Complexity(8192 vs 1000000)
Running  monobit
Running  frequency_within_block
Running  runs
Running  longest_run_ones_in_a_block
Running  dft
Running  non_overlapping_template_matching
Running  serial
Running  approximate_entropy
Running  cumulative sums
Running  random_excursion
Running  random_excursion_variant
Test results:
- PASSED - score: 0.659  - Monobit - elapsed time: 0 ms. Details: 0.658531366498405
- PASSED - score: 0.969  - Frequency Within Block - elapsed time: 0 ms. Details: 0.9687335495122532
- PASSED - score: 0.875  - Runs - elapsed time: 1 ms. Details: 0.8753699494044527
- PASSED - score: 0.094  - Longest Run Ones In A Block - elapsed time: 6 ms. Details: 0.0941032500883576
- PASSED - score: 0.855  - Discrete Fourier Transform - elapsed time: 1 ms. Details: 0.8551910615589
- PASSED - score: 1.0    - Non Overlapping Template Matching - elapsed time: 12 ms. Details: 0.9999556429171125
- PASSED - score: 0.386  - Serial - elapsed time: 160 ms. Details: [0.09459929 0.67742847]
- PASSED - score: 0.025  - Approximate Entropy - elapsed time: 261 ms. Details: 0.024622929108098077
- PASSED - score: 0.779  - Cumulative Sums - elapsed time: 6 ms. Details: [0.89202296 0.66502149]
- FAILED - score: 0.087  - Random Excursion - elapsed time: 26 ms. Details: [3.81709648e-35 2.58682777e-22 5.57118450e-12 3.03350348e-01
 3.94164630e-01 6.17463868e-17 6.91764451e-39 1.56273902e-82]
- PASSED - score: 0.376  - Random Excursion Variant - elapsed time: 0 ms. Details: [0.78172092 0.84315554 0.90569434 0.88229812 0.6361417  0.33661455
 0.20862689 0.19588084 0.12722834 0.42409446 0.12242553 0.09483041
 0.04808779 0.05654593 0.11508236 0.11762264 0.33945223 0.52457588]
```

Figura 1: Resultado de la ejecución de los tests estadísticos.

El resultado de la ejecución (Figura 1) muestra la siguiente información:

- Lista de tests que no son elegibles debido al tamaño de la muestra, junto con el tamaño requerido.
- El nombre del test que está ejecutándose en ese momento. Útil para pruebas.
- Los resultados del test: incluyen información sobre si el test ha pasado (PASSED) o fallado (FAILED) segun el score calculado (el p-valor obtenido), el nombre del test, su tiempo de ejecución y detalles sobre el score (p-valor) obtenido. En algunos casos, es posible que el

resultado del p-valor reportado esté por encima del umbral (0,01) pero el test se muestre como fallido. Eso es debido a que en ese test en concreto se calculan varios p-valores y al menos uno de ellos es menor que el valor de significancia escogido.

1.2.2. Cálculo de la entropía

En el caso del cálculo de la entropía, no se pudo ejecutar el código oficial y se optó por codificarlo desde cero con Python siguiendo la información del artículo original del NIST (Turan et al. 2018). En este caso se ha dividido la ejecución en dos archivos independientes, uno centrado en la estimación de la entropía mínima y otro con el resto de pruebas incluidas dentro de la publicación (comprobación de que las muestras sean independientes e idénticamente distribuidas, y test de reinicio.)

Las muestras de una fuente de ruido se consideran independientes e idénticamente distribuidas si cada una de ellas tiene la misma distribución de probabilidad que otra muestra, y si todas las muestras son mutuamente independientes. Para comprobarlo, se utilizarán los tests de permutación, donde se comparan valores estadísticos con valores inferidos de los datos de entrada, en vez de con distribuciones estándar.

1.2.3. Entropía mínima

La publicación oficial del NIST comentada en el apartado anterior incluye un procedimiento para estimar la entropía de los números aleatorios generados denominado entropía mínima. En nuestro caso, damos por hecho que la fuente de ruido es independiente e idénticamente distribuida (IID), ya que la ejecución del test que lo comprueba no nos ha llevado a otra conclusión, así que usaremos el estimador del valor más común para obtener la entropía de la muestra. En el caso de que la fuente de ruido no sea IID, la estimación de la entropía se complica, usándose hasta diez estimadores no incluidos en los entregables.

Para realizar esta estimación, primero se identifica el número de veces que aparece el valor más repetido en la secuencia, dividido por la longitud de la misma, y aplicar ese resultado para encontrar su límite superior (Ecuación 1, siendo x_i el elemento i-ésimo de la secuencia, L la longitud de la misma, y $2{,}576$ la constante estadística $Z_{(1-0{,}005)}$).

$$\begin{aligned} \hat{p} &= max_i \frac{\#\{x_i \, in \, S\}}{L} \\ p_u &= min\left(1, \hat{p} + 2{,}576\sqrt{\frac{\hat{p}(1-\hat{p})}{L-1}}\right) \\ H_{est} &= -log_2(p_u) \end{aligned} \tag{1}$$

En el código proporcionado, se reporta el valor estimado de mínima entropía para tres secuencias de diferente longitud, para poder detectar anomalías cuando se requiere un gran número de valores al generador cuántico de números aleatorios.

1.2.4. Otros tests

Se codificaron dos tests más relacionados con la estimación de la entropía: el test de IID y el test de reinicio. Ambos están comentados en la publicación 800-90B (Turan et al. 2018).

En el apartado anterior se comentó que partíamos de la premisa de que la fuente de ruido es independiente e idénticamente distribuida. Existen unos tests específicos para realizar esta comprobación, y que se adjuntan como entregables, pero no se han incluido en la automatización al tener tiempos de ejecución muy largos.

Otro tipo de test que se adjunta pero no se aplica, por falta de conocimiento sobre su utilidad debido a la naturaleza del generador cuántico de números aleatorios, es el test de reinicio. La estimación de la entropía de una fuente de ruido, si se calcula a partir de una secuencia de salida larga, podría proporcionar una sobreestimación si la fuente de ruido genera secuencias correlacionadas después de los reinicios. En la documentación no se especifica qué tipo de reinicio aplicar, ya que depende del generador. En el caso del generador cuántico de números aleatorios, el reinicio podría ser pedir una nueva secuencia sin realizar la llamada de desconexión, pedir una nueva secuencia llamando a la función de desconexión o reiniciando el instrumento físicamente.

1.3. Diseño y ejecución de tests automáticos, y su posterior visualización

La información que puede extraerse de la ejecución de los tests mencionados anteriormente no nos proporciona conocimiento alguno sobre la calidad de los números aleatorios generados si solo realizamos cálculos puntuales. Es por ello que con la ejecución de este proyecto se proporciona una solución que permite ejecutar automáticamente las métricas y los tests, y conocer el resultado en tiempo real. Dicha solución incluye:

- Paneles de visualización en formato json compatibles con Grafana.
- Scripts en bash para el envío de métricas y otros datos a ElasticSearch.
- Adaptación de la ejecución de las métricas para que realice una llamada con el resultado al script mencionado en el punto anterior.
- Comandos cron para controlar la ejecución automática de dichos tests.

2. Resultados

2.1. Ejecución de tests

En este apartado comentaremos los resultados obtenidos al aplicar el procedimiento explicado en la sección 1.2.1. En la Tabla 1 se muestra el resultado de ejecución de los tests para dos tamaños de muestra diferentes: 1024 y 102816 bits.

Los resultados obtenidos eran negativos en bastantes tests, lo que significa que las muestras analizadas serían poco aleatorias. Para descartar un error de codificación, se compararon los resultados con aquellos reportados en la publicación del NIST y con una aplicación, también

Resultado (p-valor) de los tests ejecutados en muestras de diferente tamaño		
Nombre del test	**1024 bits**	**1028016 bits**
Monobit	0.4920	0.3540
Frequency within a block	0.6640	0.5340
Runs	0.3730	0.9610
Longest run of ones in a block	0.2250	0.0861
Binary Matrix Rank	ND	0.8840
Discrete Fourier Transform	0.6460	0.000
Non-Overlapping matching	0.9910	0.3470
Overlapping template matching	ND	0.000
Maurers Universal	ND	0.000
Linear complexity	ND	0.000
Serial	0.5830	0.000
Approximate entropy	0.3340	0.000
Cumulative sums	0.7170	0.000
Random excursion	0.035	0.6300
Random excursion variant	0.337	0.000

Cuadro 1: Resultado de la ejecución de los diferentes tests considerando una muestra pequeña (1024 bits) y una muestra mayor (1028016 bits, que es el mínimo aceptado por algunos tests). ND refiere a aquellas pruebas con un número insuficiente de muestras para ser ejecutado. En rojo marcamos aquellos valores que indican que el test ha fallado, y por tanto, la muestra no es suficientemente aleatoria. Los valores marcados como erróneos con un valor superior a 0.0100 son aquellos que reportan un único valor pero que consideran un abanico de soluciones, siendo al menos una de ellas inferior al valor umbral.

de código abierto, disponible en línea[1] (Tabla 2). Otra discrepancia encontrada se encuentra al incrementar el tamaño de muestra, ya que algunos tests que reportaban valores aceptables para secuencias aleatorias utilizando secuencias cortas (1024 bits), fallan al usarse secuencias mayores, como es el caso de los test Serial, Approximate entropy y Cumulative sums.

De los quince tests, solo siete de ellos reportan resultados que coinciden entre los valores reportados en la publicación y los obtenidos con la ejecución del código. Cuatro de los siete tests se reportarán en el panel de visualización (Sección 3.6): Monobit, Runs, Approximate entropy y Cumulative sums. Las excepciones son, por un lado Random excursion y Random excursion variant ya que éstos no retornan un valor concreto al ejecutarse, sino una lista de p-valores los cuales todos deben ser mayores a la significancia indicada (0.01 para el 99.9% de certeza), y por otro lado Binary Matrix Rank, ya que precisa de muestras grandes. El resultado del test Cumulative sums se reporta con la media entre el valor superior e inferior.

Otra prueba que se realizó fue comparar los resultados al ejecutar una muestra obtenida con el generador cuántico de números aleatorios y otra muestra obtenida con la función built-in random que contiene Python. Para los siete tests que reportan resultados coherentes con la literatura (tal y como se ha comentado en el párrafo anterior), excluyendo Binary Matrix Rank por el tamaño de muestra que precisa, no se encuentran diferencias significativas (Tabla 3) ya

[1] `https://mzsoltmolnar.github.io/random-bitstream-tester/`

Resultado (p-valor) de los tests en tres orígenes diferentes			
Nombre del test	**Publicación NIST**	**Código Python**	**Web app**
Monobit	0.1096	0.1100	0.1995
Frequency within a block	0.7064	0.4930	1.0000
Runs	0.5008	0.5010	0.5008
Longest run of ones in a block	0.1806	0.0070	0.1806
Binary Matrix Rank	0.5320	0.5320	0.5321
Discrete Fourier Transform	0.1689	0.6460	ND
Non-Overlapping matching	NT	NT	NT
Overlapping template matching	0.11043	0.0000	0.1104
Maurers Universal	NT	NT	NT
Linear complexity	2.7000	0.0000	0.8454
Serial	0.844 - 0.562	0.000	0.8440 - 0.562
Approximate entropy	0.2353	0.2353	0.2350
Cumulative sums	0.2191 - 0.1149	0.167 (mean)	0.21 - 0.11
Random excursion	Non-random	Non-random	Failed
Random excursion variant	Random	Random	Random

Cuadro 2: Comparativa del resultado de la ejecución de los tests según la publicación del NIST, el código en Python adaptado y los mismos tests implementados en una aplicación en línea. Las celdas coloreadas muestran aquellos tests cuyo resultado es equiparable tanto en la publicación como en el código ejecutado. ND: no disponible por la plataforma, NT: no testeado por la naturaleza de los parámetros de entrada usados en la publicación.

que la ejecución reiterada de estos tests devuelve resultados sensiblemente diferentes en cada caso, tal y como se espera de una muestra no reproducible. Este comportamiento se podrá observar con más claridad con los paneles de visualización.

2.2. Estimación de la entropía

En la sección 1.2.3 se ha comentado el procedimiento seguido para construir el estimador. Dicho estimador se ha aplicado a muestras de diferente tamaño (un, dos y tres millones de bits), ya que la fórmula usada para el cálculo depende de la longitud de la cadena (Ecuación 1), y de dos orígenes diferentes: el dispositivo QRNG y la biblioteca random de Python. En la Figura 2 se muestran gráficos para ambos orígenes para los tres tamaños de muestra.

Si comparamos las muestras de diferente tamaño, podemos observar que la diferencia entre la entropía mínima de las cadenas de 1 millón y 2 millones de muestras es mayor que entre las muestras entre 2 millones y 3 millones para las muestras de ambos orígenes. Por otro lado,

Resultado (p-valor) de los tests ejecutados en muestras de diferente origen		
Nombre del test	**Muestra de QRNG**	**Muestra de bib. random**
Monobit	0.492	0.466
Runs	0.373	0.291
Approximate entropy	0.334	0.012
Cumulative sums	0.717	0.129
Random excursion	Non-random	Non-random
Random excursion variant	Non-random	Non-random

Cuadro 3: Comparación entre el p-valor de los tests seleccionados para muestras de 1024 bits de diferente origen (QRNG y biblioteca random de Python)

si comparamos el origen de la muestra, los resultados son muy parecidos entre ellos, siendo sensiblemente menores en la mitad de los casos de tamaño de muestra de 1 millón, pero no se puede asegurar si es un artefacto.

En la Sección 3.6 se muestra el panel de visualización de estos valores. El algoritmo programado recogerá muestras de datos de los tres tamaños diferentes mostrados en el párrafo anterior.

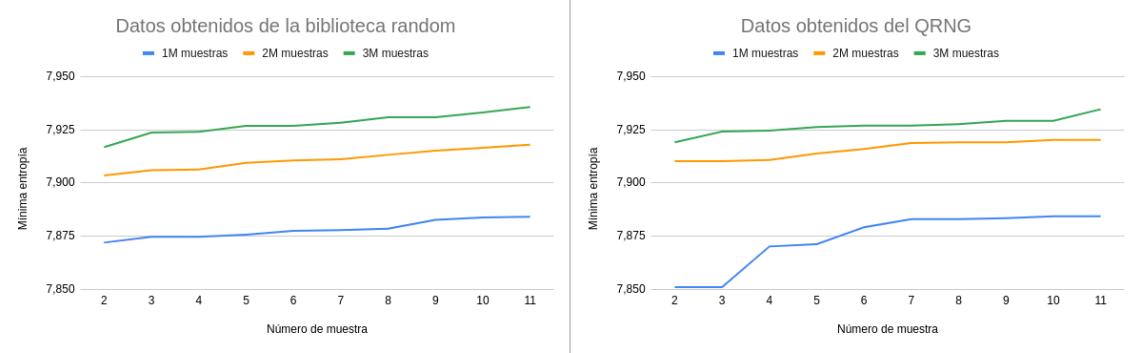

Figura 2: Valores de entropía mínimo para varias muestras de un, dos y tres millones de bits, ordenados de menor a mayor.

3. Entregables e instrucciones

A continuación se listan los entregables relacionados con este proyecto, y sus instrucciones de uso. Todos ellos, a excepción del comando cron que se mostrará en la Sección 3.4, se encuentran comprimidos en un archivo llamado "entregables-CTS-2023-055-TestRNG.zip" adjunto a este documento.

- Batería de tests NIST SP 800-22.

- Estimador de entropía NIST SP 800-90b.
- Tests extra relacionados con el cálculo de la entropía.
- Scripts en bash para la ejecución automátca de los tests y el estimador.
- Panel de visualización en formato JSON para su posterior importación en Grafana.

3.1. Código de la batería de tests NIST SP 800-22

El código para ejecutar la batería de tests se encuentra en la carpeta ''automation_tests'' e incluye el archivo ''qrng_test.py'' y la carpeta ''nistrng''. El primero incluye la codificación principal y es el que debe ejecutarse, el cual llamará a las funciones incluidas en la carpeta ''nistrng''. Este script se puede ejecutar de dos formas diferentes: inyectando el resultado en ElasticSearch (python qrng_test.py) o simplemente mostrándolo por pantalla añadiendo la opción ''-v'' (verbose) a la línea de comandos. La ejecución de este script se puede modificar para que use secuencias de diferente origen y longitud. Esas modificaciones son:

- El código para llamar al QRNG está en la línea 50. Ahí puede cambiarse la cantidad de bits a pedir.
- En la línea 116 podemos definir si queremos usar el generador de números aleatorios de Python, seteando la variable ''run_classic'' a True.
- En las líneas 131 y 132 se especificar si queremos usar un archivo externo que incluye la secuencia (formado por números en formato decimal escritos cada uno de ellos en una línea diferente), o usar el instrumento QRNG. Se añadió para evitar bloquear el instrumento al realizar las pruebas.
- En la linea 140 podemos dar opciones de cómo codificar la secuencia. El QRNG retorna números de hasta 32 bits, así que podemos usarlos todos (opción por defecto) o bien especificar si queremos mapear dichos números de hasta 32 bits dentro del rango de los 8 bits, o bien si queremos usar los ocho últimos bits de la cadena de 32 original, ya que son los menos significativos, y por tanto los que presentan mayor aleatoriedad (Chen et al. 2019).
- En las líneas 146-148 tenemos un pequeño código para cortar la secuencia, en caso de ser necesario.
- En la línea 177 se especifica el script en bash que inyectará los datos de salida en ElasticSearch. Se debe cambiar el path completo al script, apuntando donde sea necesario.

3.2. Código de los estimadores de entropía según NIST SP 800-90b

El código necesario para ejecutar el estimador de entropía mínima se encuentra en la carpeta ''automation_entropia''. Ahí encontraremos el scripot ''calculo_entropia.py''. Este código se ejecuta de forma similar al anterior, con el comando ''python calculo_entropia.py'' para la inyección de datos en ElasticSearch o usando el parámetro ''-v'' (verbose) para mostrar el resultado por pantalla. Este script ejecuta tres pasos para la recogida de muestras a diferentes

tamaños (uno, dos y tres millones de bits), y después transforma el resultado a un número dentro del rango de los ocho bits usando los bits menos significativos de la cadena original. Los parámetros que puede cambiar el usuario son:

- El origen de los datos y su tamaño en las líneas 145-147: en el estado actual del script se indican los bits a pedir al QRNG, pero puede usarse 1, 2 o 3 (tal y como indica el comentario en el código) para usar datos generados por la biblioteca random de Python. Si eso es lo que se desea, también deberán descomentarse la sección "Mocked" que va desde las líneas 45 a la 58, y comentar el resto de la función para no pasar por el QRNG.
- En la línea 170 se encuentra el path al script de inyección de datos a ElasticSearch. Éste deberá modificarse una vez movido dicho script al servidor.

3.3. Otros tests relacionados con el cálculo de la entropía

En la carpeta "other_algorithms" se encuentran los tests de IID y de reinicio que se codificaron pero no se usaron a posteriori. Se adjuntan por si son de utilidad. Estos tests se ejecutan a partir del script "calAndTestQRNG.py" de la misma forma que el test anterior, pero no inyectan datos a ElasticSearch, el parámetro "-v" solo incluye un resumen de la entropía calculada. También incluye otro parámetro de entrada llamado "--no-test" que evitará que se ejecute el test de IID y de reinicio, obteniendo únicamente la entropía, para acelerar el proceso. En otro script incluído en la carpeta, "permutationTest.py" incluye algunas funciones requeridas por el primero. La ejecución de este script es bastante duradera, pudiendo tardar hasta varias horas, y su funcionamiento se puede controlar modificando las siguientes líneas:

- En la línea 49 podemos controlar el número de bits a utilizar.
- En el bloque entre las líneas 58 y 61 podemos usar un archivo de entrada. En nuestro caso se usó para no bloquear el QRNG.

3.4. Comando *crontab* para la ejecución de las pruebas

Los scripts comentados anteriormente pueden ejecutarse automáticamente mediante crontab. Para ello, ejecutamos "crontab -e" en el servidor donde se quiera ejecutar los scripts, y se añaden ambos. En el ejemplo inferior indicamos que el test de métricas se ejecute cada día a las 8 y a las 20h, y el de entropía mínima a las 7 y a las 19h:

- * 8,20 * * * python /path/to/qrng_test.py >/path/to/output.txt
- * 7,19 * * * python /path/to/calculo_entropia.py >/path/to/output.txt

3.5. *Scripts* para la inyección de datos en *Elasticsearch*

Los scripts para la inyección de datos en ElasticSearch se encuentran en las siguientes localizaciones:

- Carpeta "automation_tests", archivo "send_elastic_data_testqrng.sh", para la inyección de las métricas. Se inyecta el resultado de las quince métricas calculadas, aunque luego se usen cuatro en el panel de visualización, tal y como se comentó en la Sección 2.1.

- Carpeta ''automation_entropia'', archivo ''send_elastic_data_entropy.sh'', para la inyección de los resultados de la estimación de mínima entropía.

Ambos scripts deben modificarse con la URL del servidor de ElasticSearch (CURL_URL dentro del script). No deben crearse los índices usados manualmente, se crean automáticamente al realizar la inyección de datos.

3.6. Panel de visualización

Por último, el panel de visualización se encuentra en formato ''json'' dentro de la carpeta ''dashboards'', con el nombre ''DashboardCesgaQRNG.json''. Antes de importarlo deberán crearse los datasources correspondientes en Grafana: uno para el índice ''testqrng'' y otro para el índice ''entropy'', ya que serán requeridos por Grafana al importar el dashboard. El resultado de las métricas automatizadas puede verse en la Figura 3.

4. Mejoras y evolución de la solución

La temática estudiada y el hardware usado durante la ejecución de este proyecto ha sido muy motivador para el personal implicado en su ejecución, y quedan algunos temas en el tintero debido al tiempo de duración del encargo:

- Adaptación de los tests originales del NIST: no hemos podido usar el código proporcionado por NIST (escrito en C) por la forma en la que está programada su ejecución (pidiendo los datos por consola) y por tener fallos de segmentación al hacer alguna de sus llamadas.
- Refinado de los tests de métricas encontrados en repositorios de código abierto: confiábamos en la calidad del código abierto usado para el cálculo de las métricas pero los parámetros de entrada de cada uno de los tests (tamaño de bloque, etc.) no se han refinado para conseguir un resultado exitoso en todas las ejecuciones. Se precisaría un estudio más exhaustivo de cada uno de los tests para refinar sus parámetros de entrada.
- Incremento del número de tests y métricas: durante la ejecución de este proyecto se siguió una metodología Agile enfocada en el principio MVP (o Producto Mínimo Viable) y su posterior iteración, con el objetivo de cumplir con todos los requisitos listados en la definición del proyecto. Por ese motivo, se optó por cumplir con los objetivos antes que a dedicar la mayoría del tiempo asignado en la primera parte del proyecto.
- Aplicación de tests IID y de reinicio. En este proyecto se han descartado debido al gran tiempo de ejecución que precisan, pero se podría trabajar en su paralelización para reducir el tiempo de ejecución.

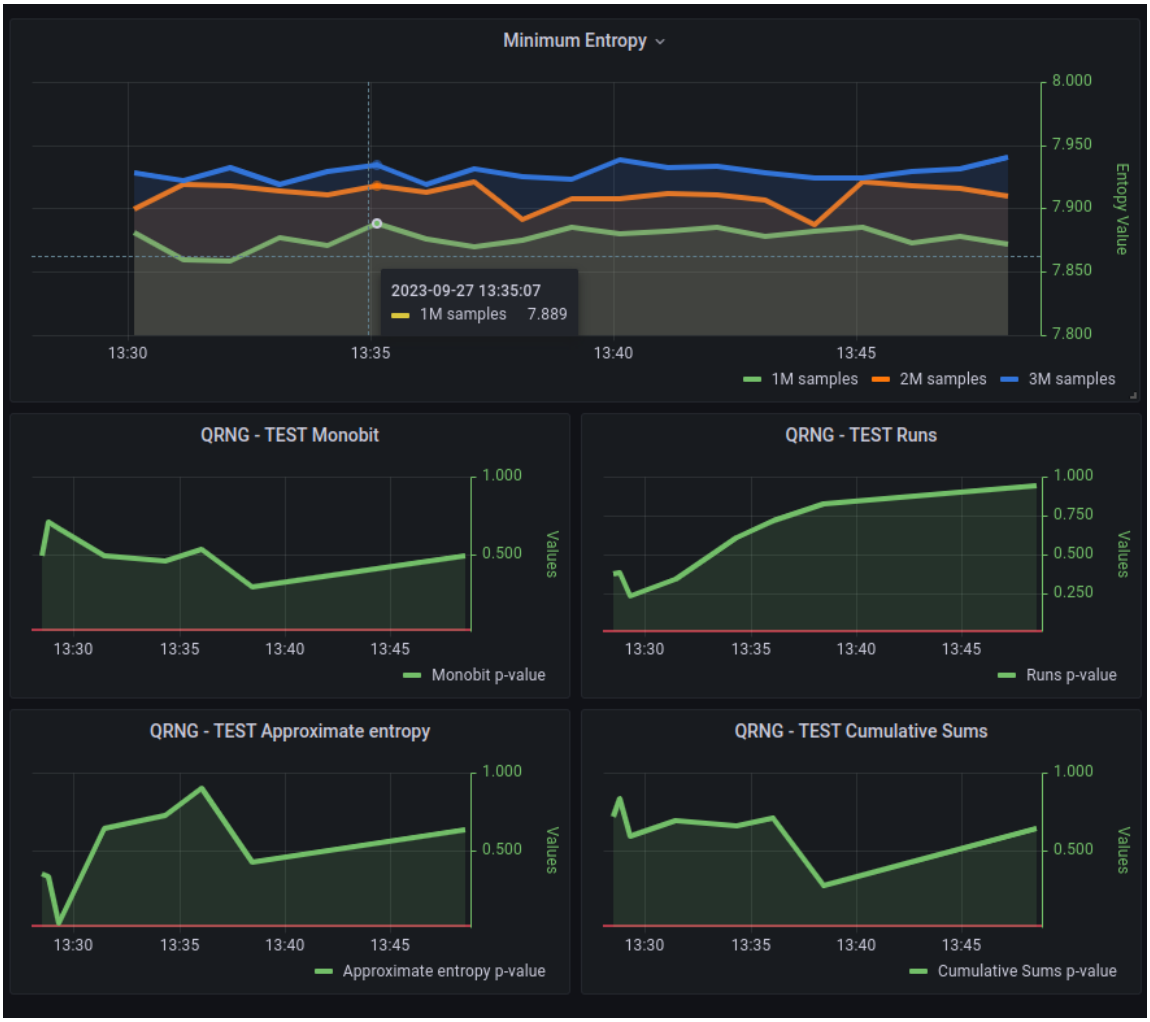

Figura 3: Ejemplo del panel de visualización de las métricas.

Referencias

Bassham, Lawrence et al. (2010-09-16 de 2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. en.

Chen, Ziyang et al. (2019). «The m-least significant bits operation for quantum random number generation». En: *Journal of Physics B: Atomic, Molecular and Optical Physics* 52.19, pág. 195501.

Elastic (2023). *Elasticsearch: motor de búsqueda*. URL: `https://www.elastic.co/es/elasticsearch`.

InsaneMonster (2023). *Insanemonster/NISTRNG: Random number generator NIST test suite framework for python 3.6 - sailab - university of siena*. URL: `https://github.com/InsaneMonster/NistRng`.

NIST (2022). *Decision to Revise NIST SP 800-22 Rev. 1a*. URL: `https://csrc.nist.gov/news/2022/decision-to-revise-nist-sp-800-22-rev-1a`.

Turan, Meltem Sönmez et al. (2018). «Recommendation for the entropy sources used for random bit generation». En: *NIST Special Publication* 800.90B.