Generation of True Random Numbers using Entropy Sources Present within Portable Computers

Conference Paper · July 2021

DOI: 10.1109/CONECCT52877.2021.9622734

6 authors, including:

Hari Om

2 PUBLICATIONS **3** CITATIONS

SEE PROFILE

Abhishek Banerji

People's Education Society

1 PUBLICATION **3** CITATIONS

SEE PROFILE

Sivaraman Eswaran

Curtin University Sarawak

47 PUBLICATIONS **250** CITATIONS

SEE PROFILE

Prasad Honnavalli

People's Education Society

88 PUBLICATIONS **259** CITATIONS

SEE PROFILE

Generation of True Random Numbers using Entropy Sources Present within Portable Computers

Rahul M Koushik
Dept. of Computer Science
PES University
Bangalore, India
rahulkaushik1999@gmail.com

Aravind Perichiappan
Dept. of Computer Science
PES University
Bangalore, India
aravindperi@gmail.com

Hari Om
Dept. of Computer Science
PES University
Bangalore, India
buzz.hari@gmail.com

Abhishek Banerji
Dept. of Computer Science
PES University
Bangalore, India
abhishekbanerji1999@gmail.com

Sivaraman Eswaran
Dept. of Computer Science
PES University
Bangalore, India
sivaraman.eswaran@gmail.com

Prasad Honnavalli
Dept. of Computer Science
PES University
Bangalore, India
prasad.honnavalli@gmail.com

2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT) | 978-1-6654-2849-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/CONECCT52877.2021.9622734

***Abstract*—Random numbers have wide-ranging applications in various domains such as cryptography, randomization of initial weights in machine learning and AI-simulation, Monte Carlo computation, industrial testing, computer games, gambling. The generation of random numbers is only possible from a sourceof entropy. A true random number generator (TRNG) uses a physical source of entropy to generate random numbers. The randomness of a TRNG can be scientifically characterized, and measured. A drawback of TRNGs is that they usually need an external hardware device containing the physical source of entropy. This necessity can be eliminated by attempting to use sources that are already part of the device's environment. This work attempts to identify such sources and analyze their entropy levels. The identified sources of entropy are then used to build a model that can be used to generate truly random numbers.**

***Keywords*—Randomness, Random number generation, Entropy source, True Random Number Generator**

I. INTRODUCTION

The generation of random numbers is vital to many computation processes such as cryptographical applications and procedural generation. A random number generator is an algorithm that outputs a sequence of numbers that exhibit the property of randomness. The output of a good random number generator should be such that it cannot be determined by an external attacker.

Random number generators can be classified into two broad categories – Pseudo Random Number Generators (PRNG) and True Random Number Generators (TRNG). TRNGs are random number generators that use natural sources such as visible lightor atmospheric noise to harvest entropy. From this harvesting, an entropy pool is created. This entropy pool contains bitsof entropy that are extracted from the values provided by the natural entropy source. Whenever a random number has to be generated, bits are depleted from the entropy pool to produce the number and the pool is refilled by harvesting data from the entropy source again. The process of harvesting entropy from natural sources can be slow since only a few bits of entropyare harvested at a time. This makes the generation of random numbers at bulk using TRNGs a time-consuming process and can block applications that need random numbers.

On the other hand, PRNGs use algorithms to compute long sequences of numbers that seem to be random, based on an initial number called the seed value. Thus, PRNGs do not make use of a entropy source (other than for the seed value) sotheir output is deterministic. As there is no source of entropy limiting the rate of production of random numbers, PRNGs are faster and allow for the bulk generation of random numbers. As few critical applications require true random numbers, PRNGs are not suitable in those cases. Hence, a TRNG is necessary to achieve true randomness.

The quality of a TRNG depends primarily on the source of entropy. But, the exact amount of entropy present in asource cannot be measured accurately. Therefore, there is a need to estimate the amount of entropy that can be extracted from a given source. There exist a number of estimators for this purpose, the most commonly used being the min-entropy estimate which is recommended by the National Institute of Standards and Technology (NIST) [5]. Based on this estimate, good entropy sources can be identified.

Existing TRNGs use internal hardware [1][4][10] and some even need special external hardware [2] in order to harvest entropy. Moreover, the physical source of entropy required for a particular TRNG may not be present in the environment of all devices. This work is an attempt to design a TRNG that makes use of internal sensors or readings that are already present within a device as the source of entropy.

II. LITERATURE STUDY

Several security protocols [12] and cryptographic algorithms use TRN. Carmen Camara et. al. proposed that by measuring the Galvanic Skin Response signal, the skin conductance value can be determined and then, in turn used to generate random numbers [2]. They followed the NIST-published SP800- 90B recommendation for the testing and estimation of the entropy. Kyungroul Lee et. al proposed a TRNG that generates random numbers from the visible spectrum [3]. Visible spectrum has information of more than 300 electromagnetic waves, and contains a lot of unpredictable noise. This noise was utilized to generate random numbers. With this work, it was proved that the entropy level can be improved by conditioning using the XOR operation on the data from the entropy source, as compared to the entropy present in the original unprocessed data. Marcin Pawlowski et. al tried to find a viable source for high entropy random number generation in the Internet of Things (IoT) ecosystem in [4]. On-board integrated sensors (temperature, humidity and two different light sensors) were analyzed as potential sources of entropy. This work proposed a solution based on the least-significant bits concatenation entropy harvesting method. The presented approach was tested in four different experiments and then a statistical fine-tuning technique was used for further optimization.

978-1-6654-2849-1/21/$31.00 ©2021 IEEE

III. ENTROPY HARVESTING

Traditional sources of entropy used by TRNGs include atmospheric noise, visible spectrum [3], and thermal noise [1]. It can be difficult to reliably collect data from these sources, especially when considering mobile devices. This greatly limits the use of TRNGs for random number generation. To overcome this, sources that are already part of these devices can be analyzed as potential sources of entropy for a TRNG. Thus, the TRNG will be able to operate in a mobile environment and produce secure random numbers at a sufficient rate.

The aim of this work was to build an efficient True Random Number Generator that uses a source that was already present in a device's environment as the source of entropy.The potential sources of entropy that were identified for this purpose are packet arrival times, packet round trip times, and data from sensors commonly found on mobile devices such as accelerometers, gyroscopes, and magnetometers. The entropy sources had to adhere to the SP 800-90B recommendation [5], published by NIST. This recommendation specifies the design principles and requirements for the entropy sources used by Random Bit Generators, and the tests for the validation of such entropy sources.

A. Design Details

Firstly, the potential sources of entropy were analyzed by performing entropy validation on these sources. For performing validation, 1,000,000 data samples were collected fromeach source and created a dataset. After the generation of the dataset, the next step is to extract the entropy bits and create a binary file that is a representation of the entropy source. This file would be used to perform validation of the entropy source.This validation process is shown in Fig 1.

For entropy estimation, NIST SP 800-90B recommended two tracks, depending on whether the entropy source is Independent and Identically Distributed (IID) or Non-independent and identically distributed (IID). For IID, Chi-Squared Tests and Permutation Tests were performed and for non-IID, the10 estimators recommended by the NIST SP 800-90B were used to estimate the entropy of a non-IID source.

B. Data Collection

Data collection from the entropy source plays a crucial role in generating random numbers. Data is gathered on variables of interest in a systematic manner. The data is then cleaned and processed to create the dataset which will be used to estimate the entropy available in the source.

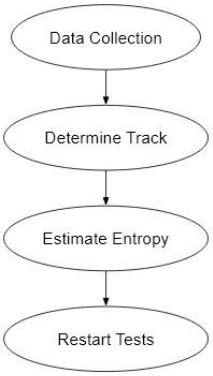

Fig. 1. Entropy source validation

1) Packet arrival time: A packet sniffer was built in C using the libpcap (library for network traffic capture). The sniffer works by using a packet capture handler on the network device that needs to be monitored. If no device is passed as a parameter, then, it sniffs on "any" device. For every packet captured, a callback function is invoked, which has data such as the packet itself, length of the packet, etc, and the most important data value in this case the timestamp (in microseconds) of when the packet was captured.

The packet sniffer program is allowed to run in the background during normal operation of the device and it keeps on collecting the data of the packets arrived on the device. This data is stored in a text file. From this file, the dataset of packet arrival timings is generated by extracting only the relevant information regarding packet arrival time and discarding all other information about the packets that were sniffed.

2) Packet round trip time: To get the round trip time of a packet, an HTTP packet would be sent to several servers. A socket would be created and a connection would be maintained with the servers. Then a basic HTTP GET request would be sent and receive a response as well. This gives the time taken by the packet to reach the server and subsequent response to reach the host in nanoseconds. These values are stored in atext file. From this file, the dataset of packet round trip time can be generated.

3) Sensor data: An Android app was built for the purpose of collecting data from sensors found in mobile phones. The app uses the Android sensor framework to detect which sensors are present on the device. Based on this, it registers itself with the available sensors. Whenever a sensor reports new data, the app stores the reported readings in a log. Thus, the app is able to collect all sensor data when it is runningand store them in the log. Based on the readings, it can be decided which sensors are viable sources of entropy. The app also provides an option to export the sensor readings in the form of a CSV file, where each sensor's data is reported separately. This is done by transferring the contents of the log into a CSV format file. The CSV file can be cleaned and processed to generate the dataset containing sensor data. Data was collected from the Accelerometer, Gyroscope, magnetometer, Light sensor, Proximity sensor, Ambient Temperature, Atmospheric Pressure, and Relative Humidity sensors during continuous usage of the phone to generate the dataset.

IV. DATA CLEANING

Before the data extracted from the entropy source can be evaluated, the data should be prepared for any further analysis by modifying or removing data that is incomplete, irrelevant, redundant, incorrect, or improperly formatted. Here, only sensor data had to be cleaned.

A. Sensor data

The Android application was built using Android Studio and uses the Android library to extract information about the user's device. This application first extracts information about all the sensors and continuously logs values from the sensors. Here, some sensors may be absent on a few devices. The values of these sensors were discarded. Some of the sensors lacked variability in terms of sensor readings. For the light sensor, the maximum value range was 30,000 numbers and

was mostly a gradual slope making the entropy value very low as observed in Fig 2.

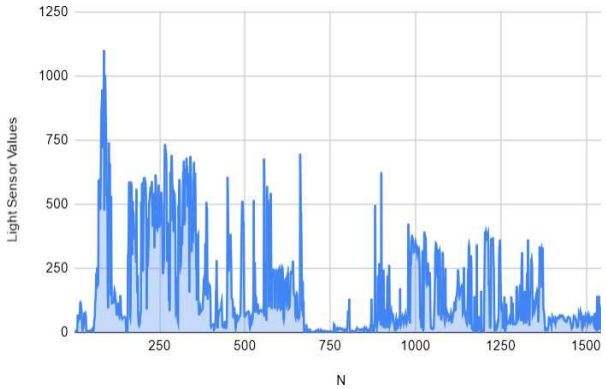

Fig. 2. Light sensor values

However, there were rare cases where it reached the cap value. In certain cases, even with no change in values, the sensor readings were logged. All these values were also removed from the dataset that was generated. Some of the values were incorrectly logged and lacked precision. For example, - 0.0002593994 would have been logged as -2.59E-04. This was re-represented and cleaned thoroughly. As seen in Fig 3, the proximity sensor readings did not represent the actual values but were rather just displaying values belonging to that value class, 0 or 5.0035, representing near and far.

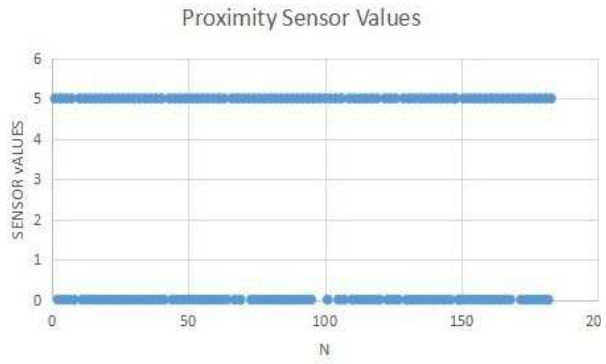

Fig. 3. Proximity sensor values

It can also be observed that the values for light sensors when plotted represented hills with gradual slopes. The rate at which the values for light and proximity sensors were recorded was significantly slower as it only noted the values only when there was a noticeable change in the environment in regards to those values. Hence, these two sensors are disregarded as sources of entropy. Further, the ambient temperature, relative humidity, and atmospheric pressure sensors were found to be absent on most Android devices, including the devices that were used for data collection. Based on this analysis, the accelerometer, gyroscope, and magnetometer were identified as the sensors from which the dataset would be generated.

V. ENTROPY ANALYSIS

On obtaining the datasets of the sensor data, packet arrival timings, and the round trip times, the data was thoroughly cleaned. Each dataset had at least 1,000,000 samples.

A. Entropy Harvesting

From the sample dataset, it was noted that most of the fluctuations in the readings were among the least significant bits. So, the last 8 bits of each sample value were extracted in order to create the sample entropy pool in the form of a binary file. Entropy analysis would then be performed on this binary file.

B. Entropy Source Validation

Each of the three entropy sources was validated in accordance with NIST Special Publication 800-90b by performing min-entropy tests on the sample datasets. According to the recommendation, at least 1,000,000 data samples need to be generated for each of the sources. The first step was to determine whether the sample represented an Independent and Identically Distributed random source (IID) or Non-Independent and Identically Distributed random source (Non-IID). Each sample in an IID source has the same probability distribution as every other sample from that source and all samples are mutually independent. Samples of IID data and Non-IID data are shown in Fig 4.

To test for IID behaviour, permutation testing and chi-square tests were performed as mentioned before. If a source fails either one of these tests, it would be considered as a Non-IID source.

TABLE I. IID TESTING RESULTS

	Chi-Square Tests Testing	**Permutation**
Packet Arrival Time	Failed	Failed
Round Trip Time	Passed	Passed
Sensor Data	Failed	Failed

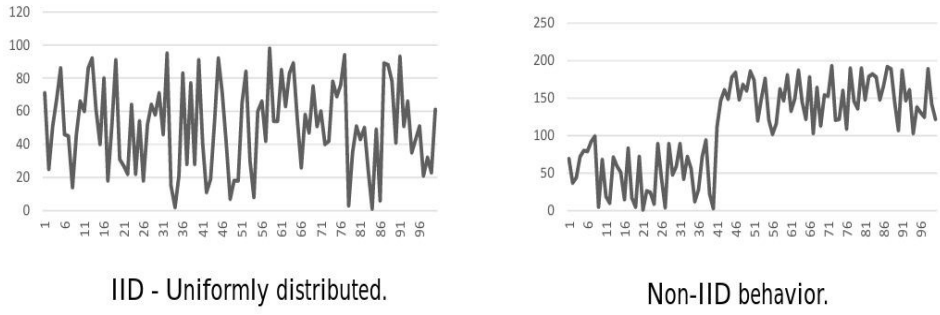

Fig. 4. IID and Non-IID behavior

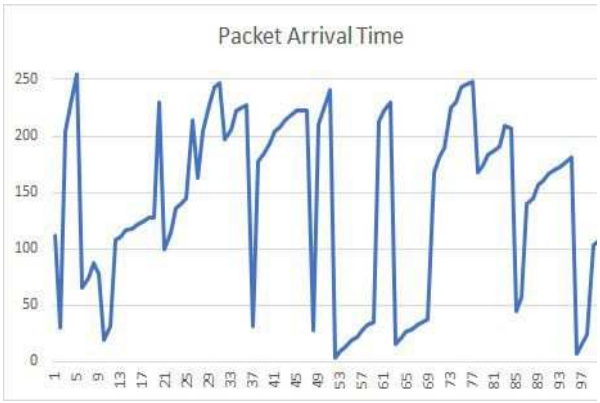

Fig. 5. Non-IID behavior of packet arrival time

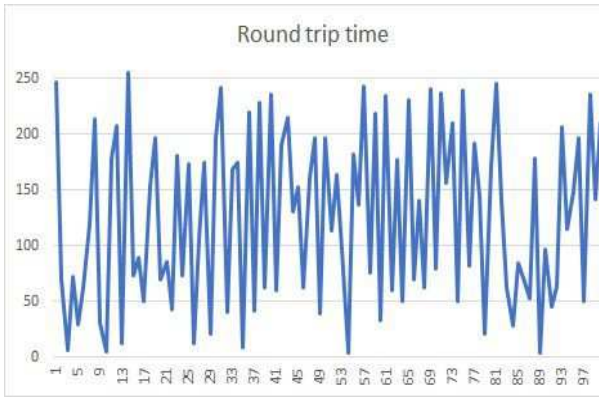

Fig. 6. IID behavior of packet round-trip time

Based on the IID or non-IID track, different tests were performed to determine the min-entropy.

1) Packet arrival time: 1,000,000 samples of 256 distinct 8-bit-wide symbols, i.e., 8,000,000 binary samples were loaded into the binary file to perform entropy analysis. This sample was found to have a raw mean of 127.4182258 and a median of 127.000000. Based on the Most Common Value (MCV) estimate the entropy of the bitstring was found to be 0.998120 and the entropy of the sample dataset was found to be 7.869929. Min-entropy is the min (Entropy of sample dataset, 8 × Entropy of bitstring) = 7.869929. However, as it failed both the permutation testing and the chi-square test, it shows that the data from the entropy source is of non-IID track, therefore the MCV estimate can't be used here and further non-IID tests have to be performed to get the min-entropy estimate of the source. The non-IID behavior of this source is exhibited in Fig. 5.

TABLE II. ENTROPY ESTIMATION RESULTS FOR PACKET ARRIVAL TIME DATASET USING NON-IID TESTS

Estimation method	Bitstring Estimate
Most Common Value	0.998120 / 1 bit
Collision	1.000000 / 1 bit
Markov	0.999339 / 1 bit
Compression	1.000000 / 1 bit
t-Tuple	0.729735 / 1 bit
MultiMCW Prediction	0.987814 / 1 bit
Lag Prediction	0.888284 / 1 bit
MultiMMC Prediction	0.760044 / 1 bit
LZ78Y	0.998408 / 1 bit

It can be seen from Table II that the min-entropy for this data sample is 0.760044 per bit for each bitstring. A complementary LRS test which is a part of this suite was not run on this dataset as it is an alternative to the t-Tuple and is only necessary to run it when the t-Tuple test does not run efficiently.

2) Packet round-trip time: 1,007,203 samples of 256 distinct 8-bit-wide symbols, i.e, 8,057,624 binary samples were loaded into the binary file to perform entropy analysis. This sample was found to have a raw mean of 127.440730 and a median of 127.000000. Based on the Most Common Value estimate the entropy of the bitstring was found to be 0.998588 and the entropy of the sample dataset was found to be 7.880283. Min-entropy is the min(Entropy of sample dataset, 8 × Entropy of bitstring) = 7.880283. For the Chi-square independence test and for the Chi- square goodness of fit test, the p-values observed were 0.885775 and 0.179445 respectively. It also passed all the tests in the permutation testing suite. These tests include number of directional runs, average and maximum collision tests, periodicity and covariance tests at intervals in powers of 2, compression tests, among others. The IID behavior of the round trip time is shown in Fig. 6, where a set of samples from this data source are graphed.

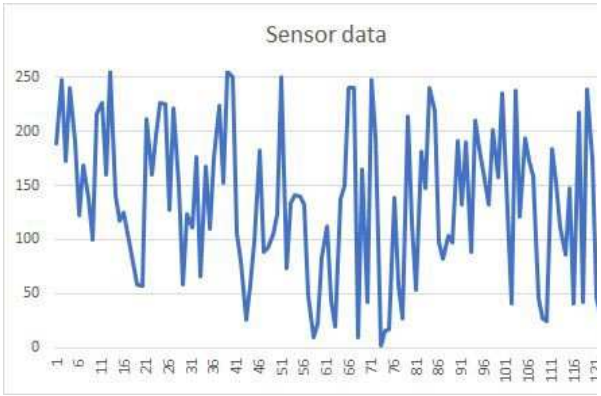

Fig. 7. Non-IID behavior of sensor data

3) Sensor data: 1,014,288 samples of 256 distinct 8-bit-wide symbols, i.e., 8,114,304 binary samples were loaded into the binary file, to perform entropy analysis. This sample was found to have a raw mean of 127.467826 and a median of 127.000000. Based on the Most Common Value estimate the entropy of the bitstring was found to be 0.986571 and the entropy of the sample dataset was found to be 7.658332. Min-entropy is the min (Entropy of sample dataset, 8 × Entropy of bitstring) = 7.658332. However, it failed the chi square test, it was a non-IID sample, therefore this min-entropy cannot be considered as the min-entropy of the sample data and further Non-IID tests have to be performed. A subset of the sample data was plotted on a graph as shown in Fig. 7 to exhibit the non-IID behaviour of the source. The Non-IID test results for entropy estimation of this source are given in Table III.

TABLE III. ENTROPY ESTIMATION RESULTS FOR SENSORS DATASET USING NON-IID TESTS

Estimation method	Bitstring Estimate
Most Common Value	0.986571 / 1 bit
Collision	0.946944 / 1 bit
Markov	0.988442 / 1 bit
Compression	0.843168 / 1 bit
t-Tuple	0.928582 / 1 bit
MultiMCW Prediction	0.992958 / 1 bit
Lag Prediction	0.810587 / 1 bit
MultiMMC Prediction	0.983829 / 1 bit
LZ78Y	0.986633 / 1 bit

After analysing the entropy levels of the three sources, sensor data and packet round-trip time data were selected as the entropy sources for the TRNGs. However, the packet arrival time was dropped as the min-entropy of this source was sub-optimal for further processing.

VI. TRNG CONSTRUCTION

After identifying viable entropy sources, they were used to construct two TRNGs according to the recommendations specified in SP 800-90C [6]. The XOR approach was selected to construct the TRNGs as shown in Fig 8.

As a part of this approach, it was necessary to identify and integrate an SP 800-90A approved PRNG and use its output XOR-ed with the TRNG's output to generate the random number. The advantage of such an approach is that in the event of failure of the entropy source, the TRNG will still output random numbers at the randomness level of the PRNG. Thus, the PRNG acts as a fallback and also enhances the randomness output by the TRNG. In the case of the round-trip time TRNG, a pseudo random number generator present in the UNIX gcrypt library was selected for this purpose. For the sensors-based TRNG, an SP 800-90A approved random number generator present in the Bouncy Castle crypto library was used.

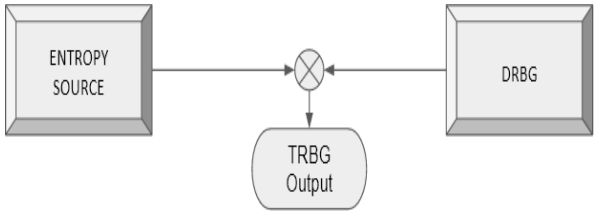

Fig. 8. XOR-TRNG construction

Continuous health tests were integrated in the TRNGs to monitor the health of the entropy sources during its operation to ensure all failures of the sources are detected. The health tests included a repetition count test and an adaptive proportion test. A brief generic pseudocode for both the TRNGs is given below:

1) Receive request for a random number of specified bitlength.
2) Request necessary amount of data from the entropy source (s).
3) Analyze entropy source health by running continuous health tests on the data samples.
4) Extract bits of entropy from the retrieved data.
5) Combine the bits of entropy extracted from multiple datasamples into one bitstring.
6) Request a NIST SP 800-90A approved DRBG to generate random bits of required length.
7) Perform a XOR operation on the random bits generated by the DRBG with the bitstring generated from the entropy source (s).
8) Convert the result of the XOR operation to a positive integer.
9) Return this integer as the generated random number and then wait for the next request.

VII. TRNG VALIDATION

For the validation of the randomness of the constructed TRNG, the Diehard tests, a battery of statistical tests developed by George Marsaglia, were used. These tests include matrix ranks test, birthday spacings test, bitstream test, parking lot test and random spheres test among others. The diehard tests were implemented in the dieharder application, developed by Robert G. Brown [7]. This application was used on the datasets to calculate the randomness strength of the TRNGs.

1) Packet round-trip time: For the round-trip time TRNG, a random sequence of over 36 million 32-bit numbers was generated in order to create the dataset required to run the diehard tests. The round-trip time TRNG passed all the tests in the diehard suite. The results of the tests are shown in table IV.

TABLE IV. DIEHARD TEST RESULTS FOR RTT TRNG

Test	p-value	Assessment
Birthdays spacings test	0.24781361	Passed
Overlapping Permutations	0.55055752	Passed
Ranks of 32x32 Matrices	0.99257668	Passed
Rank of 6x8 Matrices	0.08266680	Passed
Bitstream test	0.58486077	Passed
Overlapping pairs sparse occupancy test	0.00510408	Passed
Overlapping quadruples sparse occupancy test	0.01441708	Passed
DNA test	0.50687371	Passed
Count 1s test (stream)	0.38285950	Passed
Count 1s test (bytes)	0.34599065	Passed
Parking lot test	0.14465251	Passed
2d spheres test	0.41704303	Passed
3d sphere test	0.80421574	Passed
Squeeze test	0.37776156	Passed
Runs test	0.79189668	Passed
Craps test	0.72493546	Passed

2) Sensor data TRNG: For the sensor data TRNG, a random sequence of over 12 million 32-bit numbers was generated in order to create the dataset required to run the diehard tests. The sensors TRNG passed all the tests in the diehard suite. Although it passed the 2d spheres test, it was assessed to be weak. Even for a very strong RNG, it is very likely that 0 to 2 tests will give a weak assessment. By increasing the sample dataset, this result can be further improved. The results of the tests are shown in Table V.

TABLE V. DIEHARD TEST RESULTS FOR SENSORS TRNG

Test	p-value	Assessment
Birthdays spacings test	0.59240373	Passed
Overlapping Permutations	0.03219095	Passed
Ranks of 32x32 Matrices	0.84276204	Passed
Rank of 6x8 Matrices	0.82542551	Passed
Bitstream test	0.33635406	Passed
Overlapping pairs sparse occupancy test	0.06644921	Passed
Overlapping quadruples sparse occupancy test	0.09174928	Passed
DNA test	0.65076972	Passed
Count 1s test (stream)	0.41364038	Passed
Count 1s test (bytes)	0.21762042	Passed
Parking lot test	0.58615511	Passed
2d spheres test	0.99871810	Weak
3d sphere test	0.90351670	Passed
Squeeze test	0.06892150	Passed
Runs test	0.39494760	Passed
Craps test	0.40835817	Passed

VIII. CONCLUSION

Based on the analysis and results, it can be concluded that sources such as device sensors and packet round-trip time can be used as entropy sources to construct true random number generators that perform well for practical use cases. Through this research, the randomness of these entropy sources and the TRNGs constructed using them has been established. As an extension of this research, future work could be done to analyse the security of these TRNGs by testing their vulnerability against attacks. Further testing can alsobe performed, for example, by using the TRNG output inimage encryption applications and then evaluating the strength of the encryption using NPCR and UACI randomness tests [13]. Work can also be done on the TRNGs to build a GSL-compatible wrapper interfaces for them, so that they can be used by other applications.

REFERENCES

[1] B. Jun and P. Kocher, “The Intel random number generator”, Cryptography Research Inc., White paper, Apr. 1999, pp. 1-8.

[2] C. Camara, H. Mart´ın, P. Peris-Lopez and M. Aldalaien, “Design and analysis of a true random number generator based on GSR signals for body sensor networks”, Sensors, 19 (9): 2033, 2019.

[3] K. Lee, S. Lee, C. Seo and K. Yim, “TRNG (True Random Number Generator) method using visible spectrum for secure communication on 5G network”, IEEE Access, vol. 6, Jan. 2018, pp. 12838-12847.

[4] M. P. Pawlowski, A. Jara and M. Ogorzalek, “Harvesting entropy for random number generation for internet of things constrained devices using on- board sensors” Sensors, 15(10): 26838-26865, 2015.

[5] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish and M. Boyle, “Recommendation for the entropy sources used for random bit generation”, NIST Special Publication 800-90B, Jan. 2018.

[6] E. Barker and J. Kelsey, “Recommendation for random bit generator (RBG) constructions”, NIST Special Publication 800-90C, Apr. 2016.

[7] R. G. Brown, “Dieharder: A random number test suite”, https://webhome.phy.duke.edu/~rgb/General/dieharder.php, [Accessed Online].

[8] L. E. Bassham, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, S. D. Leigh, M. Levenson, M. Vangel, N. A. Heckert and D. L. Banks, “A statistical test suite for random and pseudorandom number generators for cryptographic applications”, NIST Special Publication 800-22 Rev 1a, Sept. 2010.

[9] E. B. Barker and J. Kelsey, “Recommendation for random number generation using deterministic random bit generators”, NIST Special Publication 800-90A Rev 1, Jun. 2015.

[10] E. Schreck and W. Ertel, “Disk drive generates high speed real random numbers”, Microsystem Technologies, vol. 11, 2005, pp. 616–622.

[11] C. Hennebert, H. Hossayni and C. Lauradoux, “Entropy harvesting from physical sensors”, Proceedings of the sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC’13), Apr. 2013, pp. 149-154.

[12] S. Eswaran, A. Srinivasan and P. Honnavalli, “A threshold-based, real-time analysis in early detection of endpoint anomalies using SIEM expertise”, Network Security, 2021 (4), Apr. 2021, pp. 7.16. https://doi.org/10.1016/S1353-4858(21)00039-8.

[13] Y. Wu, J. Noonan and S. Agaian, “NPCR and UACI randomness tests for image encryption”, Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications, Apr. 2011, pp. 31-38.