An Efficient Random Number Generator for Low-Power Sensor Networks

Deva Seetharam and Sokwoo Rhee
Millennial Net
201 Broadway, Cambridge, MA - 02139.
{dseetharam,sokwoo}@millennial.net

***Abstract*— We have designed an ultra-low power sensor networking platform called the i-Bean Network [1], [2]. The i-Bean Network requires a reliable RNG (Random Number Generator) for various purposes such as random backoffs, random transmission delays and random packet sequence numbers.**

We could not use the existing RNGs because they require special purpose hardware and/or involve complex computations. To conserve battery power by minimizing computations, we attempted to develop a simple and efficient RNG.

In this paper, we describe a simple RNG based on a free-running timer. Although this RNG was specifically designed for the i-Bean Network, we believe that this generator could be useful in other low-power embedded networks.

I. Introduction

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. - John Von Neumann (1951)

We have designed an ultra-low power sensor networking platform called the i-Bean Network [1], [2] to enable a wide variety of control and automation applications. The i-Bean Network is composed of numerous embedded computing devices that communicate with each other to haul data between sensors and host computers. These devices require a reliable RNG for the following purposes:

- Random Packet Sequence Numbers - The IEEE 802.15.4 standard [3] specifies that packet (beacon, data and command frames) sequence numbers must be random[1].
- Random Backoff - When packets collide in the channel, the transmitting nodes back off and wait for a random length of time before retransmitting. This random wait duration reduces the probability of two nodes retransmitting at the same time.
- Random Transmission Delays - If a node receives a broadcast packet, it waits a random length of time before rebroadcasting.

Since the RNG would be used as often as the packets are transmitted, an efficient generator is necessary to decrease power consumption and to increase battery life. We could not use the existing RNGs [4], [5], [6] either because they require special purpose hardware and/or complex computations.

Adding extra hardware would increase the hardware dimensions and that is not acceptable because the i-Bean devices are designed to be tiny to be unobtrusively embedded in their operating environment. We find even simple computations such as as division and modulo arithmetic operations to be expensive, since the microcontroller platform (Microchip PIC) doesn't have native support for these operations.

A software implementation of linear feedback shift register could have been used since it neither requires special hardware nor complex computations. However, the simple implementation is slow and the faster ones require parallelization [7].

Since we couldn't find a suitable RNG, we attempted to design a simple and efficient pseudo random number generator, despite Von Neumann's ominous warning.

II. Random Number Generator

We designed the generator with the following design constraints:

- The generator must be efficient and we define efficiency as follows:
  - It doesn't require any multiplication or division operations. It would be desirable if the generation could be achieved using just the logical operations.
  - The maximum number of steps required for generating a single random number in the sequence is less than ten.
  - The maximum code memory required is used is less than 50 bytes.
- The generator must produce an uniform distribution of random numbers between 0 - 255 inclusive.
- The generator must not use any microcontroller specific features that would prevents us from porting this code to other hardware platforms. However, we exploit the nature of application features. For example, since this generator is intended for communicating embedded systems, we use the checksum/CRC of the transmitted/received packets to re-key the generator.

Based on these constraints, we have developed a RNG based on a free running timer. A random number is generated by XORing the current value of the timer with a key. After generating a random value, ones complement of the timer value becomes the new key and the ones complement of the new random number becomes the next timer value. In addition, the key also gets updated with the 8-bit CRC values of transmitted and received packets. The code is given below.

[1]Since acknowledgement packets don't carry the sender ID of the original packet, a random sequence number serves the dual purpose of identifying the packet and the source node.

```
/* Initialize Key to the ID of the node */
unsigned char key = nodeID;

unsigned char random()
{
    unsigned char rv = 0;
    unsigned char tv = 0;

    tv = get_timer();

    rv = tv ^ key;
    key = ~tv;
    tv = ~rv;
    set_timer(tv);

    return rv;
}
```

Although RNGs based on free-running timers have been proposed before [8], we are not aware of a low-power RNG tailored for sensor networks.

III. Evaluation

We implemented this generator on a Microchip PIC 18F8720 platform running at a clock speed of 8 MHz. We ran this RNG with initial $key = 25$ and $tv = 0$ and changed the key once every 10 iterations to emulate[2] packet transmission or reception.

This implementation produced 10 million random bytes in approximately 200 minutes. That is, it could generate 50,000 random bytes per minute. A plot of the distribution of random numbers is presented in Figure 1.

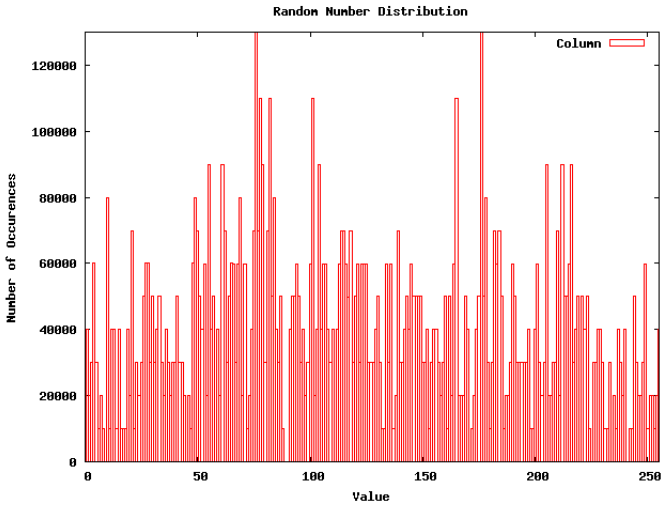

Fig. 1. Random Number Distribution.

[2]We believe that this emulation is reasonable because the RNG would not be invoked without any ongoing communications.

We used ENT [9] for testing this random number generator. ENT performs the following tests and produced the corresponding results:

Test	Result	Ideal Results
Entropy	7.713327 bits per byte. Optimum compression would reduce the size of this 10000000 byte file by 3 percent.	8 bits per byte
Chi-square Test	Chi square distribution is 3828655.45, and randomly would exceed this value 0.01 percent of the times.	Depends on the distribution and a percentage value of between 10% and 90% the sequence is truly random.
Arithmetic Mean	122.885	127.5
Monte Carlo Value for PI	3.088126835 (error 1.70 percent)	Value of PI
Serial Correlation Coefficient	-0.058927	0.0

TABLE I
RNG Analysis.

IV. Discussions and Future Work

This RNG seems to be efficient and produces apparently random numbers. However, it must be subjected to a rigorous theoretical analysis to ensure its correctness. Further more, the results presented in Table I indicate this RNG could be improved. The most important issue is its poor performance in the Chi-square test. Moreover, the performance of this RNG must be tested in the sensor network environment to ensure that multiple devices in the network don't go through the same sequence of random numbers. More importantly, since we use CRC (or checksum) of the packets to update the keys, we must make sure that the apparently independent processes don't inadvertently synchronize with each other [10].

V. Acknowledgements

We sincerely thank Dr. James McBride of MIT Media Lab, Dr. Sheng Liu of Millennial Net and Mr. Peter Gonzalez of BlueEl technology for invaluable discussions and suggestions.

References

[1] S. Rhee, D. Seetharam, and S. Liu, "i-bean network: An ultra-low power wireless sensor network," in *UBICOMP Adjunct Proceedings*, 2003.
[2] Millennial, "Millennial net," http://www.millennial.net.
[3] I. . Committee, "802.15.4: Wireless medium access control and physical layer specifications for low-rate wireless personal area networks," http://www.ieee802.org/15/pub/TG4.html.
[4] T. Ritter, "Random number machines: A literature survey," http://www.ciphersbyritter.com/RES/RNGMACH.HTM.
[5] P. L'ecuyer, *Handbook of Computational Statistics*. Springer Verlag, 2004, ch. Random Number Generation.
[6] ——, "Uniform random number generation," *Annals of Operations Research*, no. 53, pp. 77 – 120, 1994.
[7] B. Schneier, *Applied Cryptography*. John Wiley and Sons, 1996.
[8] S. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. Trickey, and P. Winterbottom, "The inferno operating system," *Bell Labs Technical Journal*, vol. 2, no. 1, pp. 5 – 18, Winter 1997.
[9] J. Walker, http://www.fourmilab.ch/random.
[10] J. McBride, "Synchronization of apparently independent processes in a network." Personal Conversations.