Quantum Random Bit Generator Service for Monte Carlo and Other Stochastic Simulations

Radomir Stevanović, Goran Topić, Karolj Skala, Mario Stipčević, and Branka Medved Rogina

Rudjer Bošković Institute, Bijenička c. 54, HR–10000 Zagreb, Croatia,
radomir.stevanovic@irb.hr

Abstract. The work presented in this paper has been motivated by scientific necessity (primarily of the local scientific community) of running various (stochastic) simulations (in cluster/Grid environments), whose results often depend on the quality (distribution, nondeterminism, entropy, etc.) of used random numbers. Since *true random numbers* are impossible to generate with a finite state machine (such as today's computers), scientists are forced either to use specialized expensive hardware number generators, or, more frequently, to content themselves with suboptimal solutions (like pseudorandom numbers generators). *Quantum Random Bit Generator Service* has begun as a result of an attempt to fulfill the scientists' needs for quality random numbers, but has now grown to a global (public) high-quality true random numbers service.

1 Introduction

1.1 On Random Number Generation

The random numbers, which are (by their definition) nondeterministic and ruled by some prescribed probability distribution, can only be (as it is generally accepted) extracted from observation of some physical process that is believed to exhibit nondeterministic behavior. Various randomness extraction methods have been used in the past and different processes were being observed, but these usually could be grouped into either (1) measurements of macroscopic effects of an underlying noise ruled by statistical mechanics (e.g. quantum noise manifested as electronic shot noise or quantum effects in optics [1], [3], thermal noise [12], avalanche noise, radioactive decay [10], atmospheric noise [9], etc.), or (2) sampling of a strictly nonlinear process (or iterated function system) that inherently exhibits chaotic behavior and intrinsical sensitivity to initial conditions (which are generally unknown or unmeasurable) and as such is considered to be random for practical purposes (e.g. chaotic electronic circuits like phase–locked loops [5], or chaotic mechanical systems [11], etc.). The only scientifically provable randomness (nondeterminism) sources, at the present state of the art, are quantum systems. Contrary to those, classical physics systems (including chaotic ones) only hide determinism behind complexity.

A cheap alternative to complex true random number generators (in terms of speed of generation or resource requirements, but on account of randomness) are algorithmic generators implemented on a digital finite state machines outputting pseudorandom numbers, i.e. deterministic, periodic sequences of numbers that are completely determined by the initial state (seed). Entropy of an infinite pseudorandom sequence is finite, and upward limited with entropies of the generating algorithm and the seed. This is *not* the case with infinite true random numbers sequences.

Regardless of the randomness acquisition method used, random number sequences are often post-processed [6] to remove bias, shape probability distribution, remove correlation, etc.

1.2 The Need for Random Numbers

Random numbers seem to be of an ever increasing importance – in cryptography, various stochastic numerical simulations and calculations (e.g. Monte Carlo methods), statistical research, various randomized or stochastic algorithms, etc. and the need for them is spanning a wide range of fields – from engineering to physics to bioinformatics. The applications usually put constraints on properties of input random numbers (probability distribution, bias, correlation, entropy, determinism, sequence repeatability, etc.). Consequently, these constraints dictate the choice of a random number generator.

If the quality of simulation or calculation results would be substantially affected, or dominated, by the (lack of) randomness of input random number sequences, then the true random number hardware generator should be used. And that imposes additional project costs, of both financial and time resources.

2 The Random Numbers Service

2.1 Overview

To ease and simplify the acquisition of high quality true random numbers for our local scientific community, we have developed the *Quantum Random Bit Generator Service* (QRBG Service for short), which is based on the *Quantum Random Bit Generator* [1]. This service is now publicly available on the Internet, from [8]. Design requirements for the Service were:

- *true randomness* of data served (nondeterminism and high entropy),
- *high speed* of data generation and serving,
- *high accessibility* of the service (easy and transparent access),
- *great robustness* of the service, and
- *high security* for users that require it.

The development of the QRBG Service is still a work in progress, but all requirements except the last one have been hitherto implemented and tested. If we exclude the security requirement from the list above, it can be said that QRBG

Service tops currently available random number acquisition methods (including existing Internet services like [4], [9] or [10]) in at least one of the remaining categories (to the best knowledge of the authors).

To ensure the high quality of supplied random numbers (true randomness) and the high speed of serving, we have used the Quantum Random Bit Generator (described in Sect. 2.2) and a carefully designed server (described in Sect. 2.3). The server design also gives the Service its robustness, which, in turn, acts as an important factor in service accessibility, namely its temporal availability. Transparent access to random data, or *access modes availability* is achieved through multiple client connectors developed (described in Sect. 2.4 and Sect. 2.5), including "black-box" C/C++ libraries and web service (SOAP) access, but also more user-friendly Mathematica and MATLAB add-ons. To facilitate high security, an SSL wrapper is being implemented and tested which will enable the encryption of transferred random data with user certificates.

The structural overview of the Service is depicted in Fig. 1. Implementation details of the Service components are given in the following sections.

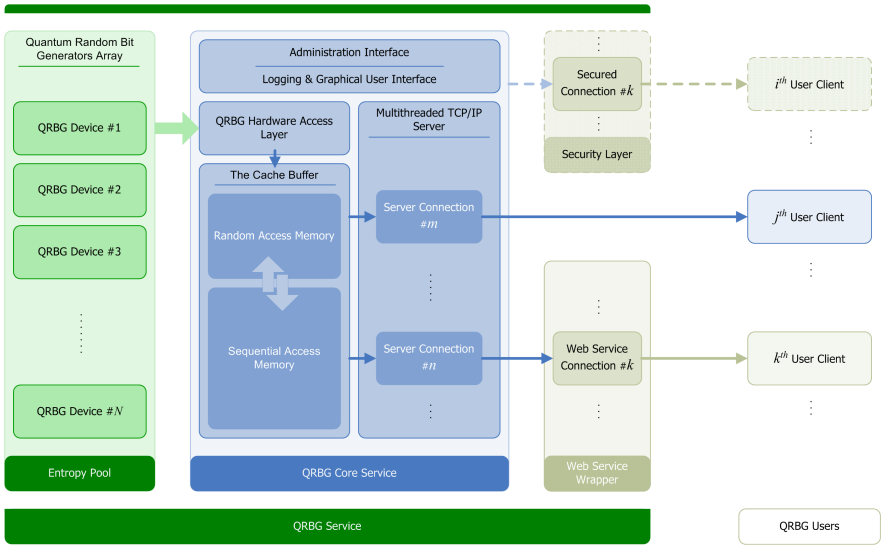

Fig. 1. The structure of the Service

2.2 Randomness Source

As a source of random numbers, an array of *Quantum Random Bit Generators* (QRBGs) is being used. The QRBG device was designed and developed at the Rudjer Bošković Institute, in the Laboratory for Stochastic Signals and Process Research, and is still in its prototype phase [1].

QRBG is a fast, nondeterministic and novel random number generator whose randomness relies on intrinsic randomness of the quantum physical process of photonic emission in semiconductors and subsequent detection by the photoelectric effect. The timing information of detected photons is used to generate binary random digits – bits, with efficiency of nearly 0.5 bits per detected random event. Device consists of a light source (LED), one single–photon detector and fast electronics for the timing analysis of detected photons providing random output numbers (bits) at (currently) 16 Mbit/sec. By using only one photodetector (in contrast to other similar solutions) there is no need to perform any fine-tuning of the generator, moreover, the method used is immune to detector instability problems, which fosters the autonomous work of the service (without the usually required periodic calibrations of the generator). For the purpose of eliminating correlations, a restartable clock method is used for time interval measurement.

The collection of statistical tests (including NIST's "Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" and DIEHARD battery of strong statistical randomness tests) applied to random numbers sequences longer than 1 Gb produced with this quantum random number generator presents results which demonstrate the high quality of randomness resulting in bias[1] less than 10^{-4}, autocorrelation[2] consistent with zero, near maximal binary entropy and measured min–entropy near theoretical maximum. For much more details on these and other performed tests results, see [2].

2.3 QRBG Core Service

The core of the QRBG Service has been written as a native Microsoft Windows NT (XP / 2003 Server) stand-alone multithreaded server application with a graphical user interface, in the C++ programming language with the support of the standard Microsoft Foundation Classes library.

All application components (servers, database and device access layers, user interface and logging, client authentication and quota management) communicate using a thread-safe notification message based queue-like structure which enables decoupling and asynchronous work of independent components (refer to Fig. 1). The flow of random data from randomness generator(s) to end–users could be seen as standard FIFO buffer–centric producer(s) – buffer – consumer(s) problem, and solved accordingly. However, to maximize aggregate throughput, transition from a FIFO to randomly accessed buffer has to be made. This enables asynchronous buffer *writers* and *readers*, and thus not only greatly improves the overall performance, but also transfers the scheduling problem of equally prioritized readers (end–users) from the application to the operating system.

[1] defined as $b = p_1 - 0.5$, where p_1 is probability of ones
[2] defined in [7]

Hardware Access Layer. The QRBG device connects to a computer via USB 2.0 interface. This enables connecting several QRBG devices and achieving random numbers acquisition speeds much higher than those of a single device (16 Mbps), since the USB standard allows connecting of up to 127 devices onto one host controller and upstream speed of up to 480 Mbps.

QRBG hardware access (layer) is implemented as a dynamic link library that communicates directly with the QRBG device driver. Random data from all the input devices in the array[3] are being constantly read (at maximum speed) and stored into the application cache buffer. All status messages are dispatched into the application notify queue.

Application Cache Buffer. To optimize data transfer for speed, a Buffer class has been implemented with the behavior similar to that of `malloc` (the standard C library memory allocation routine). Namely, the complete buffer storage space is split into blocks of variable size. Each block can be either: *loaded* or *empty* (state flag), and at the same time: *ready* or *busy* (access flag). The state flag specifies whether the block contains random data or it's empty, and the access flag tells if some reader or writer thread is allowed to access the block. Interface of the Buffer component features two sets of grasp/release methods – for loaded and empty block: `graspEmpty(size,...)`, `releaseEmpty(...)`, `graspLoaded(size,...)` and `releaseLoaded(...)`. All of these methods work with block descriptors only. When a loader thread wants to copy data from QRBG device(s) into the buffer, it "grasps" an empty block in the buffer, copies the data, and then "releases" the block. Whenever a reader thread needs random data, it similarly "grasps" an loaded block (with random data), serves the data, and then "releases" the block (which is after that considered empty). The Buffer component takes care of joining consecutive free or loaded blocks. Also, it transfers data between randomly accessed (and faster) and sequentially accessed (and slower) parts of the buffer, when needed.

TCP/IP Core Server. When a user (directly, or indirectly through some of the QRBG extension services) requests random data from the QRBG Service, it is served (over TCP/IP network protocol) by the QRBG Core Server. Upon connection attempt, if a client is allowed to connect (IP is allowed and server isn't overloaded), communication begins. The communication protocol is inspired by the Simple Authentication and Security Layer protocol [14] but is extremely simplified – only two binary messages are exchanged. First, the client sends a request packet (with requested operation, login credentials and other data specific for requested operation – usually number of requested random bytes). Server responds with status message followed by a requested amount of random data (if the user was authenticated and his download quotas weren't exceeded) and closes the connection. Users' login credentials and all usage statistics are stored in a MySQL database through the QRBG Database Access Layer.

[3] due to a high price of a single QRBG device, we currently have only one device in the array

Due to random data caching, the data transfer rates achieved with QRBG Core Server (on a mid-range Microsoft Windows XP computer), exceed 45 MiB/s in loopback configuration and 11 MiB/s on a 100 Mbps local network. With empty cache, transfer rate falls bellow QRBG Device theoretical speed limit, adding some 20–30% overhead.

HTTP Status Server. While the Service is running, various statistics are collected concerning the clients connected, quantity and quality of data generated and served, etc. Simple HTTP server is running on the same address as the Core Server is and it enables users to inspect the status of the Service from their web browsers.

Administration User Interface. Configuration and administration of the Service is performed locally, through its graphical user interface. Screen capture of application windows is given in Fig. 2.

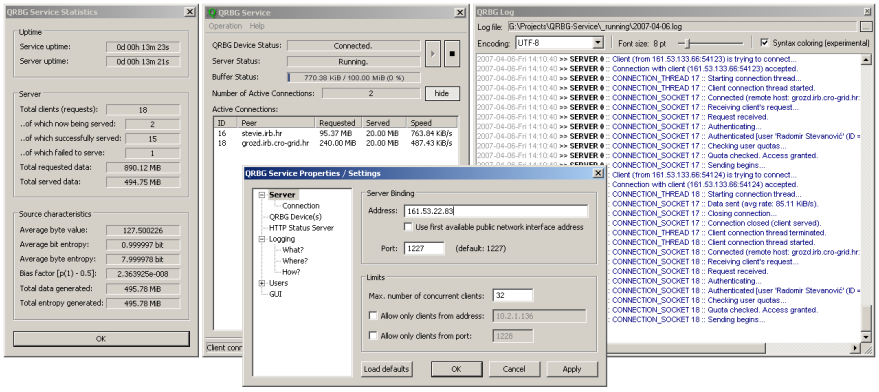

Fig. 2. The application's user interface (from left to right: statistics box, main window, settings dialog and log output.)

2.4 Extension Services

Additional features, new protocol support and other extensions of the QRBG Core Service can be easily implemented as QRBG Extension Services. Two such extensions are developed.

Web Service Wrapper. Provides SOAP protocol extension [15] of the basic QRBG Service. The web service is implemented in a standard fashion and executes in a stand-alone web server. Various random data fetching methods

(getByte, getInt, getFloat, etc.) simply relay user requests to the QRBG Core Service and return SOAP–encoded results to the user. Slowdown due to relaying is irrelevant (non-significant), since web services are inherently slow. The only purpose of implementing the web service wrapper was to simplify connectivity to the Service from environments that have natural support for the web services (this could extend the QRBG usage to high-level web and similar scripting applications).

Secure Access Wrapper. Provides SSL protocol extension of the QRBG Core Service. Like the web service wrapper, it relays user requests, but in addition enables on–demand content encryption. Also, user authentication is carried out using certificates and challenges, a much more secure method than Core Service's username/password authentication. We, however, do not expect too heavy usage of the secure access, since it (1) will slow down transfer, and (2) won't provide absolute security for sensitive cryptography (which can only be achieved with a local quantum random number generator). A foreseen primary usage is on behalf of the users that already have certificates in their cluster/Grid environments and whose Virtual Organization becomes an authorized user of the QRBG Service. The secure wrapper is still under testing.

2.5 End–User Interface

To maximally simplify the acquisition of high-quality random numbers, we have tried to make the access to the QRBG Service as transparent as possible from as many platforms/environments we could. The work in this segment is in no way over, and many more clients (service connectors) are still to be written.

Basic Access. A simple C++ class for transparent access to the QRBG Service has been developed. It features the acquisition of standard data types and a local cache of user–defined size. Since it is written in a standard, widespread language, it compiles cross–platform (Windows/Linux, 32/64–bit) and makes a good starting point for both users and developers. To illustrate the simplicity of its usage, we quote here a complete code segment that acquires a 100 double precision floating point numbers uniformly distributed on interval $[0, 1\rangle$.

```
QRBG random;
random.defineUser("username", "password");
double x[100];
random.getDoubles(x, 100);
```

Command-line Utility. Based on the C++ client, a powerful and option–rich cross–platform command–line tool has been written. It is intended primarily, although not exclusively, for Linux users. A GUI–enabled version was written for Windows users.

QRBG Toolbox for MathWorks MATLAB. For the users of this powerful engineering platform we have also developed seamlessly integrateable QRBG extension. Its main function, `qrand`, has a syntax similar to the MATLAB built–in `rand` function, with notable semantic difference – it returns a matrix of *true* random numbers (64-bit floats from $[0, 1\rangle$), and not pseudorandom numbers.

```
> qinit('username', 'password');
> m = qrand(5, 5);
```

QRBG Add-on for Wolfram Mathematica. Similar to the client examples above, we also developed a Mathematica add-on based on C/C++ MathLink Software Developer Kit [17].

3 Conclusions and Future Work

We have presented a solution for the problem of simple acquisition of high quality true random numbers – an online random number service. While using a fast nondeterministic quantum random number generator and writing a robust and scalable, performance–tuned application around it, we were driven by requirements of true randomness delivery, fast serving, high access transparency and high service availability.

The development of the QRBG Service is still a work in progress, and future work will include: extending access transparency by creating more client access modes (connectors), testing and opening a secure wrapper of the Service, and opening the Service to a wider public.

References

1. Stipčević, M., Medved Rogina, B.: Quantum random number generator based on photonic emission in semiconductors. Review of Scientific Instruments **78**, 045104 (2007)
2. Stipčević, M.: Quantum Random Bit Generator (QRBG). `http://qrbg.irb.hr`
3. Quantis: Quantum random number generator. `http://www.idquantique.com/`
4. Quantis: Quantum RNG online service. `http://www.randomnumbers.info`
5. Bernstein, G. M., Lieberman, M. A.: Secure random number generation using chaotic circuits. IEEE Trans. Circuits Syst. **37** (1990) 1157-1164
6. Proykova, A.: How to improve a random number generator. Computer Physics Comm. **124** (2000) 125–131
7. Knuth, D. E.: The Art of Computer Programming, Vol. 2, Semi-numerical Algorithms, 3^{rd} edition, (Addison-Wesley, Reading, 1997)
8. Stevanović, R.: QRBG Service Online. `http://random.irb.hr/`
9. Haahr, M.: Random.org – An atmospheric noise based online true random numbers service. `http://random.org/`
10. Walker, J.: Hotbits – An radioactive decay based online true random numbers service. `http://www.fourmilab.ch/hotbits/`

11. Silicon Graphics: Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system (also known as Lavarand). U.S. Patent 5732138
12. Intel 80802 Firmware Hub chip with included thermic noise based RNG. http://www.intel.com/design/software/drivers/platform/security.htm
13. VIA C3 CPU with included chaotic electronic system based RNG. http://www.via.com.tw/en/initiatives/padlock/hardware.jsp
14. The Internet Engineering Task Force: Simple Authentication and Security Layer (SASL). RFC 2222. http://www.ietf.org/rfc/rfc2222.txt
15. W3C: Web Services Architecture. http://www.w3.org/TR/ws-arch/
16. MathWorks MATLAB Documentation: MATLAB's interface to DLLs.
17. Wolfram Mathematica Documentation: MathLink.