Masaryk University
Faculty of Informatics

The automated testing of randomness with multiple statistical batteries

Master's Thesis

Ľubomír Obrátil

Brno, Spring 2017

Replace this page with a copy of the official signed thesis assignment and the copy of the Statement of an Author.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ľubomír Obrátil

Advisor: RNDr. Petr Švenda, Ph.D.

Acknowledgement

TODO

Abstract

TODO

Keywords

TODO

Contents

1 Introduction **1**

2 Used third-party statistical software **2**
2.1 *Terminology* . . . 2
2.2 *Batteries supported by RTT* . . . 3
2.2.1 NIST Statistical Test Suite . . . 3
2.2.2 Dieharder . . . 4
2.2.3 TestU01 . . . 4
2.3 *Unexpected behavior and errors of the batteries* . . . 5
2.3.1 Preventive measures in RTT . . . 7

3 Randomness Testing Toolkit **8**
3.1 *Local unified interface* . . . 8
3.1.1 Command-line arguments . . . 9
3.1.2 Toolkit settings . . . 10
3.1.3 Battery configuration . . . 12
3.2 *Storage for the analysis results* . . . 16
3.2.1 File storage . . . 16
3.2.2 MySQL database storage . . . 17
3.3 *Remote service* . . . 18
3.3.1 Database server . . . 20
3.3.2 Storage server . . . 21
3.3.3 Backend server . . . 21
3.3.4 Frontend server . . . 22
3.3.5 Web Interface for RTT . . . 23

4 Analysis of outputs of cryptographic functions, comparison with EACirc **26**
4.1 *Result interpretation* . . . 26

5 Analysis of DIEHARDER results on quantum random data **27**

6 Conclusions **28**

A Sample file with RTT settings **29**

B Example of file storage report file **31**

C Experiment submission online form **33**

1 Introduction

- Randomness, why should we test it (defects, low entropy, etc...)
- Statistical testing of randomness

2 Used third-party statistical software

In this chapter, we will explain terminology specific to Randomness Testing Toolkit; present a quick overview of the statistical software used in the toolkit and describe the observed and unexpected behavior of the statistical software in edge cases. We also list undertaken measures to mitigate the undocumented behaviour in our further experiments.

2.1 Terminology

Throughout the thesis, we are using certain expressions in the context of Randomness Testing Toolkit and the tools and math it uses. We list these terms along with their explanations here.

(Statistical) Battery
A program developed by a third party serving as a tool for evaluation of randomness of arbitrary binary data. A statistical battery usually contains one or multiple statistical tests. The final result of the assessment is based on the results of the tests. Examples of statistical batteries are NIST Statistical Test Suite, Dieharder or TestU01.

(Statistical) Test
A single unit in statistical battery that measures some property of the tested binary stream (e.g. number of zeroes). The test can have multiple variants and subtests, and the result of the test is one or multiple statistics.

Null hypothesis - H_0
The hypothesis H_0 denotes the hypothesis that the tested data stream is random. Based on the results of the test, we can either reject H_0 and say that the data is not random or not reject it. In the latter case, we assume that the tested data is indeed random.

P-value
In our hypothesis testing, a p-value is a probability of obtaining equal or more extreme test result while H_0 holds true. In our situation, a p-value denotes the probability that we would get same or more extreme test results when testing truly random data. Therefore, the closer the p-value is to 0 the less is the probability the tested data is random and vice versa.

Alpha - α
Significance level based on which we either reject or not reject H_0. We can specify some interval (e.g. $[0, \alpha]$) and if the result of the test (p-value) falls into this interval, we will reject the hypothesis that the tested data is random. Alternatively, we can also reject p-values that are too extreme on both sides of the interval (outside of $\left[\frac{\alpha}{2}, 1 - \frac{\alpha}{2}\right]$).

Statistic
The value obtained by certain calculation from first level p-values. Multiple statistics can be obtained from one set of p-values e.g. when testing the set for uniformity we can use Kolmogorov-Smirnov Test or Chi-Square test. Based on values of statistics of a test we can decide rejection of H_0.

Test sample
A single execution of a statistical test. The result of single test execution is one first level p-value. By repeating execution of the test, we obtain multiple p-values. By using a certain statistic (e.g. Kolmogorov-Smirnov test), we can calculate a single second level p-value.

Variant of a test
Many tests can be parametrized in some ways, possibly giving different results with the same input data. We don't treat multiple executions of a single test with different settings as separate units but rather as variants of that test.

Subtest
Some tests, even when executed only once, may measure multiple properties of the data thus providing multiple results. For example, Serial Test from Dieharder battery will measure frequencies of all 1, 2, .., 16-bit patterns in the data. We treat these measurements separately - as subtests of the test. Subtests can have multiple separate statistics.

Job
Single execution of Randomness Testing Toolkit that will analyse single binary data file with single statistical battery.

Experiment
Batch of jobs that analysed single data file with various statistical batteries and settings.

2.2 Batteries supported by RTT

2.2.1 NIST Statistical Test Suite

The battery of statistical tests was developed by National Institute of Standards and Technology (cit.). The battery implements 15 statistical tests for evaluating randomness of input data.

The reference implementation is not used in RTT because it is considerably slower than its optimized counterparts. The faster version of NIST STS used in RTT was developed by Zdeněk Říha and Marek Sýs(cit.).

2.2.2 Dieharder

Dieharder is a battery designed by Robert G. Brown at the Duke University (cit.). The battery features user-friendly console interface with the possibility of fine-grain modification of the test parameters. The fact that Dieharder is included in repositories of some Linux distributions (cit manpage) adds to its popularity and ease of use. Dieharder includes all tests from the older statistical battery Diehard (cit.), three tests from NIST STS and several other tests implemented by the author.

Since the original Dieharder implementation doesn't output all of the information we needed for interpretation and evaluation of the results, we had to modify the source code of the battery. RTT uses this modified Dieharder.

2.2.3 TestU01

This library of statistical batteries was developed at Université de Montréal by Pierre L'Ecuyer et al. (cit.). It contains a wide range of tests from NIST STS, Dieharder, and literature. It also implements various pseudo-random number generators. The statistical tests are grouped into multiple categories each intended for different use-case scenario. We will treat these categories of tests as separate batteries. TestU01 includes following ten batteries.

Small Crush, Crush, Big Crush
Small Crush battery is very fast and needs a relatively small amount of data to run - around 250 megabytes. Small Crush is also the only battery that can be natively used for analysis of data in a binary file, for the use of Crush and Big Crush, the user has to implement PRNG with the TestU01 interface. Crush and Big Crush batteries are more stringent and need gigabytes of data and a few hours to finish while Big Crush is more time and data demanding.

Rabbit, Alphabit, Block Alphabit
Tests in these batteries are suited for testing hardware bit generators and can be applied to an arbitrary amount of data. Data can be provided either as a binary file or PRNG implementing the TestU01 interface.

PseudoDIEHARD, FIPS_140_2
Tests in PseudoDIEHARD imitate DIEHARD battery; FIPS_140_2 battery implements a small suite of tests in NIST standard(cit.) Randomness Testing Toolkit doesn't support these two batteries since they are subsets of other supported batteries.

Since TestU01 is available only as an ANSI C library, we developed a console interface for it. The interface implements a dummy number generator that provides data from a supplied binary file to the battery. Our interface allows us to apply batteries to arbitrary binary data even when the batteries don't support this feature natively.

2.3 Unexpected behavior and errors of the batteries

To examine the boundary behavior of the above-listed tools, we used them to process extremely non-random data streams. The data streams that we used as the input to the batteries were two binary data files consisting of only zeroes and ones respectively. The settings of the batteries remained set to default.

Below we list observed undocumented behavior that differs from the execution of the batteries with non-extreme input.

NIST Statistical Test Suite

Each test in the battery processed 1000 separate data streams. Each data stream was 1000000 bits long.

Tests Random Excursions and Random Excursions Variant are not applicable to all possible data streams. In a regular run with reasonably random data, this doesn't matter much, as the tests are repeated multiple times, and the final result will simply be calculated from a lesser number of p-values.

Neither of the tests can be applied to a stream full of zeroes or ones. This causes absence of results when analyzing such data. The user can find out the fact that the tests are not applicable to provided stream after he inspects logs of the program; otherwise, the interpretation of the missing results is left to him.

Dieharder

When processing extremely non-random data with Dieharder, we observed various erroneous events. The events are summarized in Table 2.1.

Test name	**Stream of zeroes**	**Stream of ones**
STS Runs	No result	No result
DAB Monobit 2	Invalid result	Invalid result
Diehard Minimum Distance (2D Circle)	Stuck execution	-
Diehard 3D Sphere (Minimum Distance)	Stuck execution	-
Marsaglia and Tsang GCD	Stuck execution	-
RGB Generalized Minimum Distance	Stuck execution	-
RGB Kolmogorov-Smirnov	Stuck execution	-
Diehard Craps	-	Stuck execution
RGB Bit Distribution	-	Stuck execution

Table 2.1: Undocumented behavior of tests in Dieharder battery

Observed errors

- **No result** Test didn't provide any p-values that would be used to calculate the final result of the test. The user is not notified of this, and the final result of the test statistic is based on default value (1.0).
- **Invalid result** Test provided resulting statistic and there was no indication of error other than that the result was again default value of the statistic (1.0). Following the definition of p-value, the interpretation of such result is that the analyzed stream was almost certainly random. This is obviously not true, as both streams are just repeating ones or zeroes respectively.
- **Stuck execution** Tests froze at a certain point in execution, did not produce any results and we were forced to kill the processes manually.

TestU01

Batteries Small Crush, Crush, Big Crush, Rabbit, Alphabit and Block Alphabit were executed. Tests that are part of multiple batteries acted in the same way across the batteries. The behavior is summarized in Table 2.2.

Test name	Stream of zeroes	Stream of ones
sstring_Run	No result	No result
sknuth_Gap	No result	-
svaria_SampleProd	All results invalid	All results invalid
svaria_AppearenceSpacings	All results invalid	All results invalid
scomp_LinearComp	All results invalid	All results invalid
scomp_LempelZiv	All results invalid	All results invalid
svaria_SampleCorr	-	All results invalid
sknuth_MaxOft	Some results invalid	Some results invalid
svaria_SampleMean	Some results invalid	Some results invalid
sspectral_Fourier3	Some results invalid	Some results invalid
sstring_HammingWeight2	Some results invalid	Some results invalid
sstring_AutoCor	Some results invalid	Some results invalid
smultin_MultinomialBitsOver	Some results invalid	Some results invalid
sstring_LongestHeadRun	-	Some results invalid
snpair_ClosePairs	Stuck execution	Stuck execution
snpair_ClosePairsBitMatch	-	Stuck execution
svaria_SumCollector	-	Stuck execution
smarsa_GCD	-	Stuck execution

Table 2.2: Undocumented behavior of tests in TestU01 library

Observed errors

- **No results** Tests reported warning and ended without any result. This is probably caused by the tests not being applicable to provided data.
- **All results invalid** All statistics of the test reported p-value very close to 1.0. This could lead the user to the interpretation that the test reports the data as an almost perfect random stream.
- **Some results invalid** Similar situation to the previous one but not all statistics of the test are close to 1.0. Results of tests statistics are either close to 0.0 or 1.0.
- **Stuck execution** Tests froze at a certain point of execution. In some cases, this is preceded by an issued warning. Tests didn't produce any results and had to be killed manually.

2.3.1 Preventive measures in RTT

Since we need to use the batteries in RTT with arbitrary binary data, we implemented following measures that mitigate above-mentioned errors in our experiments.

- Tests that don't produce any results are ignored and treated as if never executed.
- Because some tests give 1.0 as a result of their statistics when the data are clearly not random, we will reject the hypothesis of randomness either when the p-value is too close to 0 or too close to 1. More specifically, H_0 will be rejected for all p-values that falls outside of the interval $\left[\frac{\alpha}{2}, 1 - \frac{\alpha}{2}\right]$. This way we reject all results that are too extreme.
- Each test is executed with the timeout. If the test doesn't finish within defined time limit, we will automatically terminate it and then treat it as if it didn't produce any results.

3 Randomness Testing Toolkit

One of our goals during working in this project was to create a tool that would provide fast and user-friendly analysis of randomness even for users not experienced in the field of statistical randomness testing. Process of statistical testing generally includes finding a suitable tool, installation of said tool, execution and interpretaion of the analysis.

Another motivation was that the developed tool would unify the process of randomness analysis that is often done by reserachers in CRoCS[1]. Before the start of the project, each researcher used his own set of scripts and tools which is difficult to manage and also complicates comparation of the experiments done by different people.

Therefore we developed Randomness Testing Toolkit[2] that serves as an unified interface of the three statistical batteries presented in Chapter 2. The toolkit consists of several connected parts that are intended for various purposes. Apart from interfaces and data storage format, the parts are independent and can be exchanged if needed.

In following sections of this chapter, we will provide detailed description of the implemented parts of the toolkit.

3.1 Local unified interface

The local unified interface is at the lowest level of the developed parts of RTT. The interface is a binary executable that accepts settings and outputs the results in common format for all batteries. The executable handles transformation of general settings into the battery specific ones, running the battery executable and then transforming gathered output into general format. The process is visualized in Figure 3.1.

The toolkit accepts settings needed for its run from three following sources:

- **Command-line arguments** – Basic settings that change for each run, for example path to input file with binary data or to file with battery configuration.
- **Toolkit settings file** – General settings for the toolkit related to the system environment e.g. paths to executables of the statistical batteries. In most cases the settings stay the same for all executions of the toolkit regardless of the battery or data used.

1. Centre for Research of Cryptography and Security
2. Also referenced as RTT or just the toolkit.

- **Battery configuration file** – Configuration file that fully configures input settings for single or multiple batteries. The configuration of the batteries can change between the executions of the toolkit.

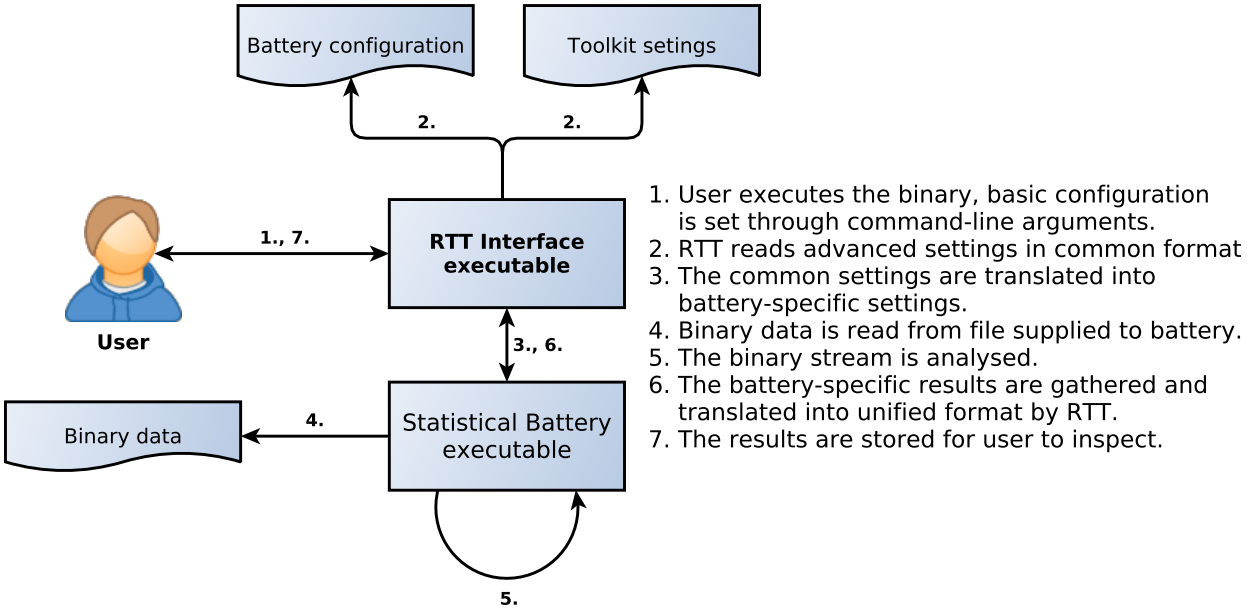

Figure 3.1: Local RTT workflow

3.1.1 Command-line arguments

The basic options of the RTT execution are set through command-line arguments. RTT recognizes following options.

- `-h` Usage of the toolkit will be printed.
- `-b <battery>` Sets the battery that will be executed during the run. Values accepted as `<battery>`: dieharder, nist_sts, tu01_smallcrush, tu01_crush, tu01_bigcrush, tu01_rabbit, tu01_alphabit and tu01_blockalphabit.
- `-t <test-id>` Optional argument. If set, only single test with id `<test-id>` will be executed in the battery.
- `-c <config-path>` Path to file with configuration of the battery. Details about battery configuration are in Section 3.1.3.
- `-f <data-path>` Path to file with the binary data that will be analyzed by the executed battery.
- `-r <result-storage>` Sets the type of the result storage that will be used. Accepted values are `file-report` or `db-mysql`. The option doesn't have to be set; `file-report` is then used as default option. More information about result storages is in Section 3.2.

- `-eid <experiment-id>` Option mandatory only when `db-mysql` is set as result storage, is ignored otherwise. Sets to which experiment in the database will be tied the results of the analysis.

3.1.2 Toolkit settings

The general settings of the toolkit are kept in separate file `rtt-settings.json` and are related to environment in which the toolkit will be executed (e.g. locations of the files and directories). The file is created by the user after the toolkit installation and stays the same for all subsequent executions of the program unless the environment changes (as opposed to input file with configuration of the batteries that can be different for each data analysis). The file `rtt-settings.json` must be located in the working directory from which the program is executed.

Basic structure of the file is in Figure 3.2. A complete example file can be found in Appendix A.

The description of the individual tags and the options that can be set within them follows.

toolkit-settings/logger
Tag containing settings related to location of log files produced during runtime.

- **dir-prefix** *Optional* – If set, value of this tag will become prefix of all log locations. Can be used when all logger directories should have same parent directory.
- **run-log-dir** – Sets location of the main log file.
- **<battery>-dir** – Value **<battery>** is replaced by values dieharder, nist-sts, tu01-smallcrush, tu01-crush, tu01-bigcrush, tu01-rabbit tu01-alphabit and tu01-blockalphabit. All of these tags are mandatory and set locations of files that will contain raw outputs of the respective executed batteries.

toolkit-settings/result-storage/file *Optional*
Section with settings needed for file result storage. All values of tags except tag **main-file** are treated as directories. For details about the storage see Section 3.2.1.

- **main-file** – Sets path of the file with final results of the program run. If the file already exists, results will be added to it, if not, new result file will be created.
- **dir-prefix** – If set, all directory values will have this prefix.
- **<battery>-dir** – Value **<battery>** can be replaced by values dieharder, nist-sts, tu01-smallcrush, tu01-crush, tu01-bigcrush, tu01-rabbit, tu01-alphabit and tu01-blockalphabit. The tags set locations of files with detailed results of the respective battery.

```
{
  "toolkit-settings": {
    "logger": {
      "comment": "Program logger settings"
    },
    "result-storage": {
      "file": {
        "comment": "File result storage settings"
      },
      "mysql-db": {
        "comment": "MySQL Database storage settings"
      }
    },
    "binaries": {
      "comment": "Executable binaries locations"
    },
    "miscelaneous": {
      "nist-sts": {
        "comment": "Miscellaneous NIST STS settings"
      }
    },
    "execution": {
      "comment": "Battery execution settings"
    }
  }
}
```

Figure 3.2: Basic structure of `rtt-settings.json`

toolkit-settings/result-storage/mysql-db *Optional*
Section with settings needed for MySQL Database storage. For details about the storage see Section 3.2.2.

- **address** Address of the MySQL server with created RTT database.
- **port** Port on which the MySQL server is accessible.
- **name** Name of the database scheme with RTT tables.
- **credentials-file** Path to file that contains login information for the database. The example credentials file is shown in Figure 3.3.

toolkit-settings/binaries
Section with the locations of the executables of the batteries.

- **<battery>** – Value **<battery>** is replaced by nist-sts, dieharder and testu01. All of the tags are mandatory. The values of the tags sets the locations of the executables of the respective batteries.

toolkit-settings/miscelaneous/nist-sts
Miscellaneous constant settings related to NIST STS battery.

- **main-result-dir** – Sets where NIST STS stores its result files.

toolkit-settings/execution
Settings related to the execution of the statistical batteries.

- **max-parallel-tests** Sets maximum number of concurrently running test processes.
- **test-timeout-seconds** Sets time period after which the running tests will be considered stuck and will be killed.

```
{
  "credentials": {
    "username": "john_doe",
    "password": "password"
  }
}
```

Figure 3.3: Example of MySQL database credentials file

3.1.3 Battery configuration

The file with battery configuration defines all settings for the battery and the tests that will be executed in the program run. Each execution of RTT can have different battery configuration; the path to file with the configuration is set through command-line argument `-c`. The basic structure of the configuration file is outlined in Figure 3.4. The complete example of configuration is in Appendix TODO.

We list the possible common and battery specfic tags here along with their required parent tags and descriptions.

Common settings

`randomness-testing-toolkit`
Root tag of the configuration file.

`<battery>-settings`
Possible parent tags: `randomness-testing-toolkit`
<battery> can be dieharder, nist-sts, tu01-smallcrush, tu01-crush, tu01-bigcrush, tu01-rabbit, tu01-alphabit and tu01-blockalphabit. This tag contains settings for specific battery.

```
{
  "randomness-testing-toolkit": {
    "dieharder-settings": {
      "comment": "Settings specific to Dieharder battery",
      "defaults": {
        "comment": "Default settings"
      },
      "test-specific-settings": [
        {
          "comment": "Settings specific for single test",
          "variants": [
            {
              "comment": "The first variant"
            },
            {
              "comment": "The second variant"
            }
          ]
        }
      ]
    }
  }
}
```

Figure 3.4: Basic structure of battery configuration file

`defaults`
Possible parent tags: `<battery>-settings`
Options inside this tag are considered as the default options for the battery specified by the parent tag.

If certain test is executed in the battery and needs to have some option defined but it does not, the option falls back to the value defined in the `defaults` tag.

`test-ids`
Possible parent tags: `defaults`
The option defines which tests will be executed in the specific battery by default. The default tests are executed only if no test was specified through command-line arguments.

Value of this tag must be array of strings. Each string can contain either a single number or range of numbers. For example, should the value of the tag be `["3", "10-13", "15"]`, then the default IDs of tests in the battery will be 3, 10, 11, 12, 13, 15.

test-specific-settings
Possible parent tags: `<battery>-settings`
This tag can contain list of single or several JSON objects. Each object defines options that are specific to a single test and the test's variants in a battery specified by the parent tag. Each of the objects must contain tag `test-id` for assigning the options to the test.

If the execution of a test requires a certain option that is not defined for the test in `test-specific-settings`, the value of the option is taken from the tag `defaults`.

test-id
Possible parent tags: `test-specific-settings`
The tag must be present in each JSON object inside of the `test-specific-settings` tag. The value of the tag is used for assigning the options in the object to actual test from the battery.

variants
Possible parent tags: `test-specific-settings`
The tag can be present in the JSON object in `test-specific-settings`. The tag can contain list of single or multiple JSON objects. Each of the variant objects can have defined additional options for test variants. The options that will be undefined by the variant object will be taken from the test object or the `defaults` tag. The variants are tied to the test which is specified by the `test-id` tag in the test object. Should the test object contain `variant` tag with list of three objects, then three variants of the test will be executed.

NIST STS settings

stream-size
Possible parent tags: `defaults, test-specific-settings, variants`
Sets bit size of single stream of data that will be used for testing.

stream-count
Possible parent tags: `defaults, test-specific-settings, variants`
Sets number of streams of data that will be used for testing. E.g. if this value would be 100, then the test would be repeated 100 times each time with different stream of length stream-size

block-length
Possible parent tags: `defaults, test-specific-settings, variants`
Some tests split data into blocks, you can modify bit length of the blocks. For details on which tests can be parametrized in this way, see documentation of tests in NIST STS.

Dieharder settings

psamples
Possible parent tags: `defaults, test-specific-settings, variants`
Sets how many times will be the test repeated before final statistic is calculated.

arguments
Possible parent tags: `defaults, test-specific-settings, variants`
Allows you to pass additional arguments to the DIEHARDER binary. Format is expected as single string of argument-value pairs separated by spaces. For summary of arguments allowed by DIEHARDER, see its documentation.

TestU01 settings

repetitions
Possible parent tags: `defaults, test-specific-settings, variants`
Sets how many times will be a test repeated. Setting this number to more that 1 will produce several subtests of the test.

bit-nb
Possible parent tags: `defaults, test-specific-settings, variants`
This option is valid only with rabbit, alphabit and block alphabit batteries. Sets how many bytes can be processed by a test. The test will not process more bytes.

bit-r
Possible parent tags: `defaults, test-specific-settings, variants`
This option is valid only with alphabit and block alphabit batteries. Sets the offset in each group of 32 bits. For example, if set to r then first r bits from each 32 bits will be ignored. Only the rest of the bits in each 32-bit block will be processed.

bit-s
Possible parent tags: `defaults, test-specific-settings, variants`
This option is valid only with alphabit and block alphabit batteries. Sets how many bits from each 32-bit block will be processed, starting from `bit-r`. For example, if we set $r = 5, s = 10$, then only 10 bits will be processed in each 32-bit block, starting from the sixth bit.

bit-w
Possible parent tags: `defaults, test-specific-settings, variants`
This option is valid only block alphabit battery. The value must be lesser or equal to `bit-s`, otherwise the option will be ignored. The option sets the reordering of the bits that will happen before processing. For details about the reordering see TestU01 documentation.

parameters
Possible parent tags: `defaults, test-specific-settings, variants`
This option is valid only with small crush, crush and big crush batteries. The value of the tag must be JSON object that will contain key-value pairs that

will represent test's parameters and their values. All parameters of the test must be specified. Each test has different settable parameters, the parameters for each test can be found in the TestU01 documentation. Example of the tag setting parameters for test smars_BirthdaySpacings is shown in Figure 3.5.

```
{
  "parameters": {
    "N": "1",
    "n": "10000",
    "r": "0",
    "d": "1000000",
    "t": "2",
    "p": "1"
  }
}
```

Figure 3.5: Parameters setting for smarsa_BirthdaySpacings test.

3.2 Storage for the analysis results

The result storages are modules in RTT interface that handle saving of the execution results for later inspection by the user. Based on the user's settings, the results can be saved in one of the two following ways.

The file storage is intended to be used when RTT is executed locally and manually by the user. After the execution, the user can inspect the human-readable reports that were generated by the module.

The MySQL database storage is used when RTT is deployed as a service on remote server. The results are stored in the database and can be viewed through web interface. It is also possible for user to install MySQL server on his local machine and setup the module so that it will use this database. The user then can work with the stored results as he sees fit.

The user can also implement his own version of the result storage module that will output the results in arbitrary format.

3.2.1 File storage

The file storage will store results into human-readable reports. Each execution of the RTT interface will generate single battery-specific report as shown in Figure 3.6. The newly generated files will have names based on the time when was RTT executed and the filename of the binary data that was used for analysis. Possible report file is shown in Appendix B.

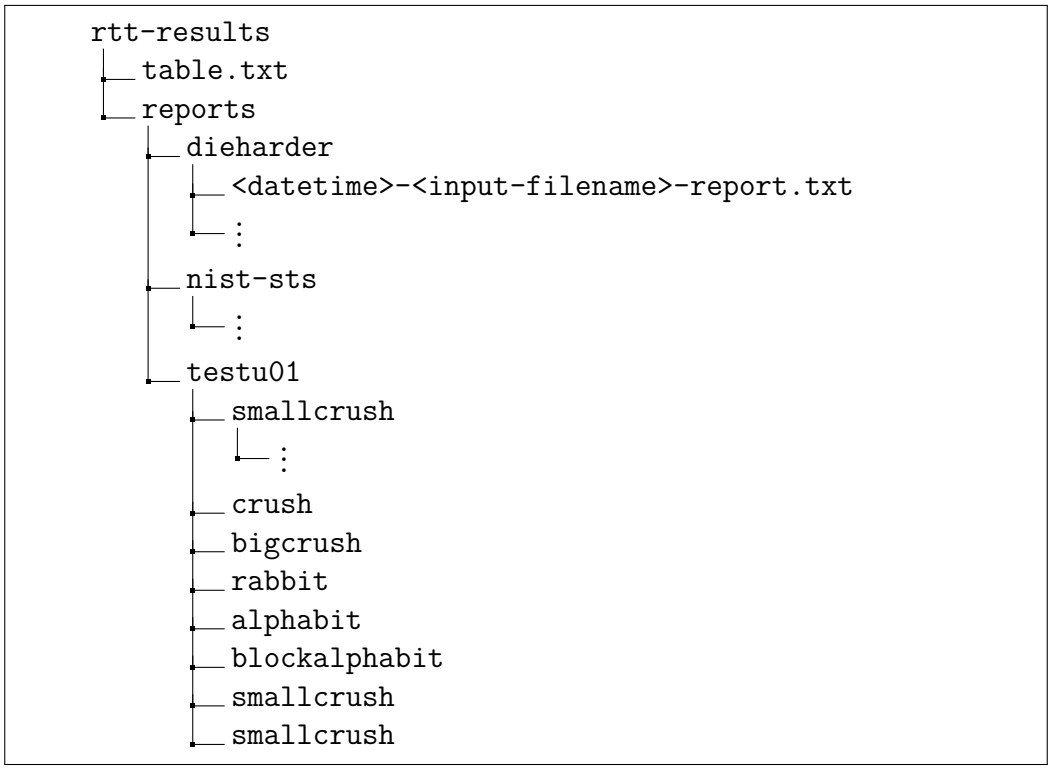

Figure 3.6: Generated files following settings from Appendix A

The main result file (in this case `table.txt`) contains summary of the result of all previous RTT executions since last deletion of the file. Therefore, if the file does not exist in time of the execution it will be generated and only modified otherwise.

In the file there is simple ASCII table that contains single row for each input data file with distinct name that was analyzed. The rows contain name of the input file, time of the last modification of the row and a proportion of passed and total number of tests for each battery. In case that a single file is analyzed multiple times by specific battery, the results are overwritten.

3.2.2 MySQL database storage

By using the database storage, RTT writes the results of the run directly into MySQL database. Layout of the database is shown in Figure 3.7. The tables `experiments` and `jobs` are not strictly required by the toolkit as the results are written only into table `batteries` and its sub-tables. The additional tables were added because we anticipated deployment of RTT on multiple servers and the tables store needed metadata about the execution and scheduling information.

`experiments`
We treat multiple analyses of single data stream by various batteries as a

single experiment. The table stores metadata about the experiment such as name of the analysed file or used configuration file.

`jobs`
The table is used for scheduling execution of RTT on remote server. It holds information about the batteries that will be executed and to what experiment the results will be tied. Single job represents single execution of RTT.

`batteries`
Represents single execution of RTT. The results of the analysis are written to this table and the tables below by the database storage. The results of multiple batteries can be assigned to single experiment.

`battery_errors`, `battery_warnings`
Stores information about errors or warnings respectively that happened during RTT runtime.

`tests`
Contains information about all tests that were executed in single battery.

`variants`
Information about all variants of a single test that was executed.

`variant_errors`, `variant_stderr`, `variant_warnings`
The errors, warnings and standard error output that were extracted from the output of a test variant execution are stored in the tables.

`user_settings`
Settings that were set by the user in the battery configuration file.

`subtests`
All subtests of a single variant.

`test_parameters`
Options and parameters that were extracted from the output of the subtest.

`statistics`
List of all statistics and their results that are tied to a single subtest.

`p_values`
List of all p-values that were extracted from the output of the subtest execution.

3.3 Remote service

Randomness Testing Toolkit can be deployed on single or multiple servers as a service, allowing the users to perform the binary data analysis without the need for the installation of the toolkit and without execution of the tool on their local machine. The statistical testing is, in certain configurations, demanding on computational time and resources. Having the RTT installed on remote machines allows us to scale the resources available for the toolkit

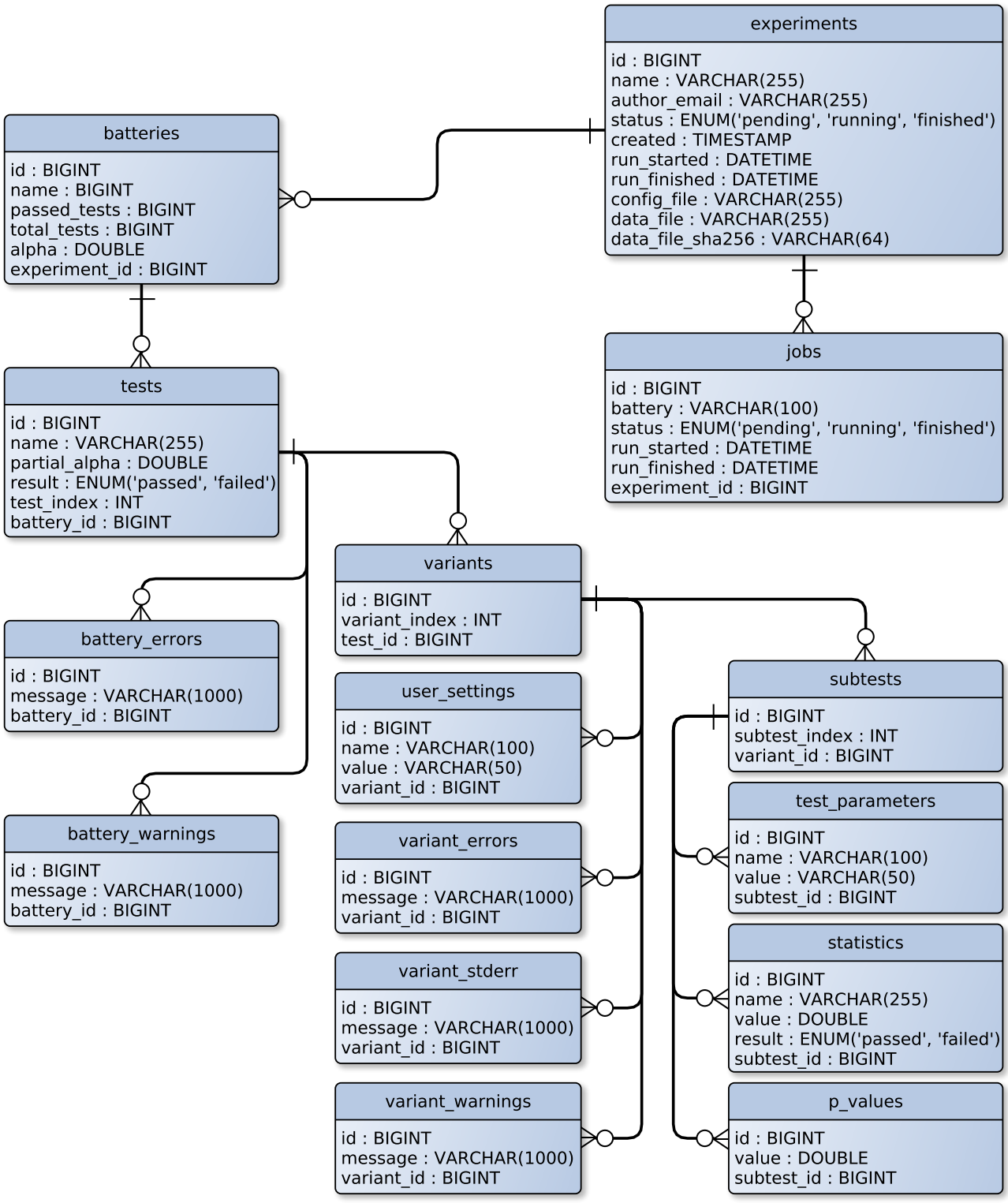

Figure 3.7: ERD Model of the database

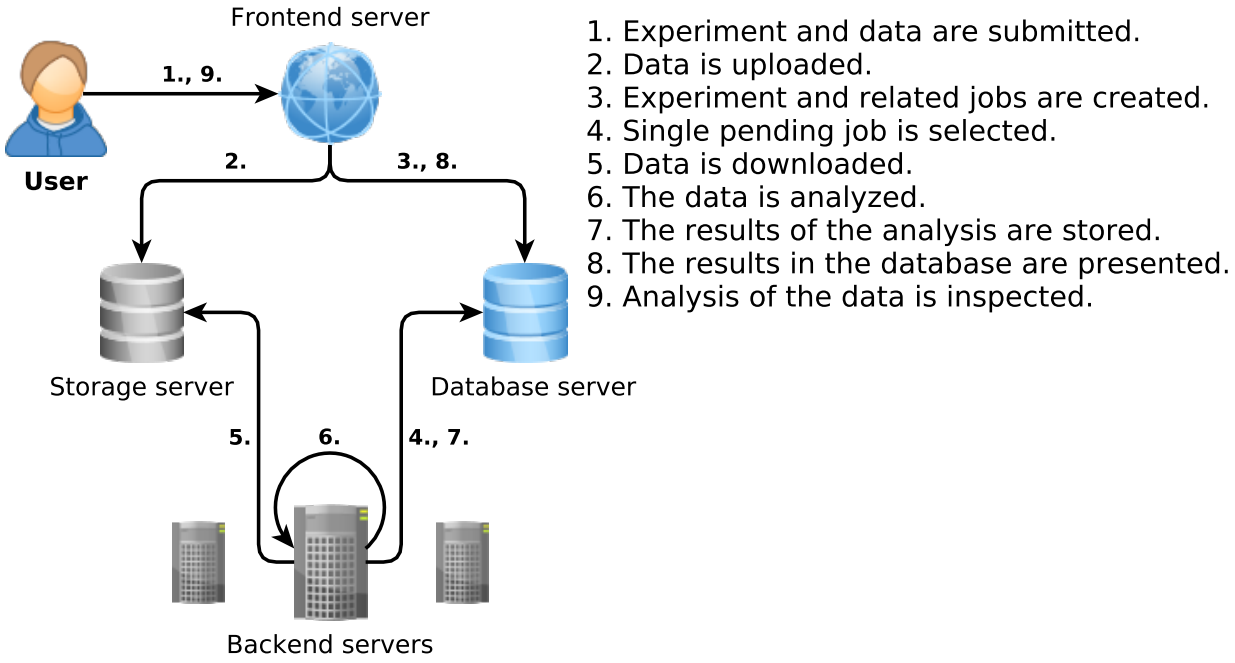

Figure 3.8: Interaction between user and servers

and further speed-up the analysis. The user only needs to provide the data for the toolkit and choose (or provide his own) configuration for the batteries. After the testing, the user is notified and can inspect the results.

In the deployment part of the project, we developed an utility project that handles setup and installation of the local interface on single or multiple machines. The deployment project also handles setup of auxiliary scripts and tools that are required for result database and data storage and that will be used by the machines used for statistical testing computation. The database and storage can be deployed on separate machines or on single server that will handle all of the tasks along with the computation of the results.

The interaction of the user with the infrastructure is visualized in Figure 3.8. In following section we outline the different purposes of the servers.

3.3.1 Database server

The machine hosting MySQL server with created database for RTT as shown in Figure 3.7. The database is used for storing the results of the analyses as well as information used for scheduling and distribution of jobs to backend servers.

3.3.2 Storage server

The server hosts a storage space that acts as a temporary deposit for the data that is not yet distributed and analysed on the backend servers. After the distribution and analysis, the data is removed from the machine.

Scripts on the server

- `clean_cache.py` – Script that handles removal of the data files that were already analysed. The script is executed periodically.

3.3.3 Backend server

Single or multiple machines that host installed toolkit and batteries. The analysis of the data is executed on these machines. If there are multiple backend servers, the analyses are scheduled based on the information stored in the database. Prior to the analysis, the data is downloaded from the storage and stored in local cache. After the computation, the results are stored on the database server and the data is deleted.

Both jobs and experiments can be in one of the three states – `pending`, `running` and `finished`; the descriptions of the states are in Table 3.9. Scheduling of the computation is based on the state of the experiments and jobs.

Status	Job	Experiment
`pending`	The job is waiting for execution.	All of the jobs related to the experiment are waiting for execution.
`running`	The job is being executed on backend server.	Some or all related jobs are being executed.
`finished`	The jobs execution ended.	The execution of all related jobs ended.

Table 3.9: States of jobs and experiments

Scripts on the server

- `clean_cache.py` – Script that handles removal of the locally cached data files that were already analysed. The script is executed periodically.
- `run_jobs.py` – Script responsible for choosing the correct computational job, downloading the data from the storage server and the execution of the analysis. Th script is executed periodically. The jobs are picked from the database table `jobs` with following priority:

1. Jobs of those experiments whose data was downloaded earlier and the data is present in the local cache of the server.
2. Any jobs of the experiments that are in pending state.
3. Any jobs that are in pending state.

The rules are designed in such a way that a minimal amount of data is transferred over the network. If there are enough distinct pending experiments for all servers then each server will pick single experiment and execute all its jobs and the data of the experiment is transferred only once.

3.3.4 Frontend server

The frontend server is the only server that the user interacts with. The machine accepts data and request for creating an experiment. After the request, the server creates the experiment and the related jobs in the database and uploads the data into the storage. The server also presents the results from the database to the users.

The server can be directly accessed through SSH. The direct access is meant for experienced users that need to analyze big volumes of data. Additionaly, an optional web interface can be deployed on the frontend server. The web interface can be used for interactive browsing of the analysis results as well as for submitting undemanding amount of data and experiments by the users.

The direct access to the server is granted to the user by the system administration. The RTT users can only access part of the system that is separated from the main system using `chroot jail` (cit.). The isolated environment contains only the tools and software that is necessary for submitting and processing the experiments. The data for the testing can be uploaded to the server from user's local machine, downloaded from other network location using `sftp` or generated directly on the server. The data generation is realized with help of the generator tool (cit.) that was developed at CRoCS and is capable of producing outputs of numerous round-reduced cryptographic primitives.

Experiment submission utility

To create a new experiment, the user has to use utility `submit_experiment` that is installed on the server. The utility will transfer the user's data into the storage and will insert new computational jobs into the database. The utility accepts following command-line arguments.

- `-h, --help` Prints the usage of the utility.
- `-n, --name <name>` Sets the name of the experiment.

- `-e, --email <address>` Optional argument. When set, an email with brief results of the analysis will be sent to `<address>` after end of the computation.
- `-c, --cfg <config-path>` Path to file with configuration of the batteries that will be included in the experiment. Details about configuring the batteries are in Section 3.1.3.
- `-f, --file <data-path>` Path to file with binary data that will be analysed by the statistical batteries included in the experiment.
- `-a, --all_batteries` Flag argument that will include all of the defined batteries except TestU01 Big Crush in the experiment.
- `--nist_sts` Switch for NIST Statistical Test Suite
- `--dieharder` Switch for Dieharder
- `--tu01_smallcrush` Switch for TestU01 Small Crush
- `--tu01_crush` Switch for TestU01 Crush
- `--tu01_bigcrush` Switch for TestU01 Big Crush
- `--tu01_rabbit` Switch for TestU01 Rabbit
- `--tu01_alphabit` Switch for TestU01 Alphabit
- `--tu01_blockalphabit` Switch for TestU01 Block Alphabit

The flags for the batteries switch inclusion of respective batteries in the experiment. For example, if we use arguments `--all_batteries --dieharder`, then the batteries included in the experiment will be the batteries added by the `--all_batteries` flag **except** the Dieharder battery. However, if we would use only argument `-dieharder`, then Dieharder will be the only battery used is the experiment.

3.3.5 Web Interface for RTT

The web interface is aiming to be useful in the basic use-case, where the user has some binary data that he wants to analyse. The user will upload the data through the website and will wait for the end of the analysis; the results will be shown on the website after the computation. The web interface is not dependent on or required by the underlying server infrastructure that executes analysis of the data. It was developed only to serve as a user-friendly access point to Randomness Testing Toolkit. For development, we used Django (cit) framework for its ease of use.

In most cases, one does not have to create new configuration for the batteries that will be executed during the analysis; a suitable configuration will be chosen for him automatically. Alternatively one can also manually choose the desired configuration or create his own configuration from the scratch according to his preferences.

The limitation of the interface is that the manual creation of large number of experiments would be too time-consuming and error-prone for the user, as the process can't be trivially automated. However the web interface not

intended to be used in such situations; creating the experiments through direct access to the frontend server would be much more viable.

User roles

We separate users into three categories in the context of the web interface application. It is worth noting that the user accounts existing in the web interface are independent of the accounts existing on the frontend server. In the following list we summarize the roles and their privileges.

- **Anonymous user privileges**
  - Browse results of all previous experiments.
  - Create an experiment if an unique access link was shared to the user.
- **Authenticated user privileges**
  - Create experiments.
  - Change details and password of the account.
- **Administrator privileges**
  - Creating new and modifying existing users.
  - Adding administrator privileges to user accounts.
  - Adding or modifying predefined battery configurations.
  - Adding or modifying unique access links to the experiment submission. Each access link has his own expiration date and is unusable after this date.

Experiment submission

To create new experiment the user user will have to complete following settings in the online form. The screenshot of the form is included in the Appendix C.

- **Experiment name** – Name identifying the experiment, doesn't have to be unique.
- **E-Mail** – Required only when the user is not logged in. After the computation, e-mail will be sent to the address, notifying the user about the results.
- **Binary data file** – Data that will be analysed; uploaded from the user's machine.
- **Configuration** – User can use default battery configuration, choose one of the predefined configurations or upload his own configuration file. The configuration files are described in Section 3.1.3.
- **Battery application** – Set of switches that defines which batteries will be included in the experiment.

Result presentation

The web interface also presents and assesses the results of the past experiments of all users. Each experiment is listed along with the batteries that were included in it. Battery results are assessed separately; the assessment is based on the proportion of passed and total number of tests included in the battery. Then the probability of achieving such proportion when testing truly random data is calculated. The calculation process is described in Section TODO. The example of a experiment evalution is shown in Figure 3.10. Based on the probability, the possible assessments are as follows.

- **FAIL** ($0 < x < 0.001$) – The result is deemed as a failure and the source of the data should not be considered as a quality randomness generator.
- **Suspect** ($0.001 \leq x < 0.01$) – The result is considered suspisious. The user should generate more data from the source and test it again. This result is possible to happen by chance even when the tested data is produced by a good source.
- **OK** ($0.01 \leq x < 1$) – The result is evaluated as a good result and the data is considered random.

Analysis done on data by statistical batteries

Name	Assessment	Passed tests	Total tests	
Dieharder	OK	25	27	Detail
NIST Statistical Test Suite	Suspect	13	15	Detail
TestU01 Alphabit	FAIL	2	4	Detail
TestU01 Block Alphabit	OK	3	4	Detail
TestU01 Crush	FAIL	28	32	Detail
TestU01 Rabbit	FAIL	13	16	Detail
TestU01 Small Crush	OK	10	10	Detail

Figure 3.10: Experiment evaluation

4 Analysis of outputs of cryptographic functions, comparison with EACirc

4.1 Result interpretation

todo

- How the data were tested
- List functions
- List interesting (differing) results - Dieharder, NIST STS, TestU01, EACirc, polynomials(???)

5 Analysis of DIEHARDER results on quantum random data

- Statistical intro, uniformity, first vs. second level p-value, etc...
- Two experiments - continuous p-values, blocks of 2nd level
- Results - non-uniform, where it will begin to show on 2nd level results

6 Conclusions

- Developed user-friendly tool for easy analysis of arbitrary binary data - Randomness Testing Toolkit
- Interpretation of results
- Comparison of batteries with EACirc, polynomials
- Defects in Dieharder, their relevance, etc...
- Future work, same analysis on TestU01, dependence between tests(?), continuous development of RTT, call for flawless statistical battery (:))

A Sample file with RTT settings

```
{
  "toolkit-settings": {
    "logger": {
      "dir-prefix": "rtt-results/logs",
      "run-log-dir": "run-logs",
      "dieharder-dir": "dieharder",
      "nist-sts-dir": "niststs",
      "tu01-smallcrush-dir": "testu01/smallcrush",
      "tu01-crush-dir": "testu01/crush",
      "tu01-bigcrush-dir": "testu01/bigcrush",
      "tu01-rabbit-dir": "testu01/rabbit",
      "tu01-alphabit-dir": "testu01/alphabit",
      "tu01-blockalphabit-dir": "testu01/blockalphabit"
    },
    "result-storage": {
      "file": {
        "main-file": "rtt-results/table.txt",
        "dir-prefix": "rtt-results/reports",
        "dieharder-dir": "dieharder",
        "nist-sts-dir": "niststs",
        "tu01-smallcrush-dir": "testu01/smallcrush",
        "tu01-crush-dir": "testu01/crush",
        "tu01-bigcrush-dir": "testu01/bigcrush",
        "tu01-rabbit-dir": "testu01/rabbit",
        "tu01-alphabit-dir": "testu01/alphabit",
        "tu01-blockalphabit-dir": "testu01/blockalphabit"
      },
      "mysql-db": {
        "address": "127.0.0.1",
        "port": "3306",
        "name": "rtt",
        "credentials-file": "credentials.json"
      }
    },
    "binaries": {
      "nist-sts": "/home/rtt-statistical-batteries/nist-sts",
      "dieharder": "/home/rtt-statistical-batteries/dieharder",
      "testu01": "/home/rtt-statistical-batteries/testu01"
    },
    "miscelaneous": {
      "nist-sts": {
```

```
42          "main-result-dir": "experiments/AlgorithmTesting/"
43        }
44      },
45      "execution": {
46        "max-parallel-tests": 8,
47        "test-timeout-seconds": 3600
48      }
49    }
50  }
```

B Example of file storage report file

```
***** Randomness Testing Toolkit data stream analysis report *****
Date:    05-04-2017
File:    data.bin
Battery: Dieharder

Alpha:   0.01
Epsilon: 1e-08

Passed/Total tests: 1/1

Battery errors:

Battery warnings:


------------------------------------------------------------
Diehard Runs Test test results:
    Result: Passed
    Test partial alpha: 0.00250943

    Variant 1:
        User settings:
            P-sample count: 10
        ************

        Subtest 1:
            Kolmogorov-Smirnov statistic p-value: 0.78507772 Passed
            p-values:
                0.05899849 0.14303914 0.25813353 0.26205674 0.29277325
                0.35258302 0.56733382 0.65239775 0.72661662 0.97024989
            ============
        #############

        Subtest 2:
            Kolmogorov-Smirnov statistic p-value: 0.98126677 Passed
            p-values:
                0.06708958 0.13845018 0.24143043 0.30884227 0.34883845
                0.45856887 0.50099963 0.72226328 0.79368168 0.94420838
            ============
        #############

    ^^^^^^^^^^^^
```

```
    Variant 2:
        User settings:
            P-sample count: 20
        ************

        Subtest 1:
            Kolmogorov-Smirnov statistic p-value: 0.94522158 Passed
            p-values:
                0.02636030 0.05899849 0.07020394 0.11462969 0.14303914
                0.25813353 0.26205674 0.29277325 0.35258302 0.42201984
                0.49409172 0.50249791 0.50852031 0.56733382 0.65239775
                0.72661662 0.87379533 0.94805962 0.95109797 0.97024989
            ============
        #############

        Subtest 2:
            Kolmogorov-Smirnov statistic p-value: 0.97098651 Passed
            p-values:
                0.05940416 0.06708958 0.10176118 0.13845018 0.18591525
                0.20171335 0.24143043 0.30884227 0.34883845 0.45856887
                0.47711566 0.50099963 0.51679868 0.57821548 0.72226328
                0.73270941 0.79368168 0.91155189 0.94420838 0.99216956
            ============
        #############

    ^^^^^^^^^^^^

------------------------------------------------------------
```

C Experiment submission online form