Masaryk University
Faculty of Informatics

The automated testing of randomness with multiple statistical batteries

Master's Thesis

Ľubomír Obrátil

Brno, Spring 2017

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

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

- 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 more 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 are not random or not reject it. In the latter case, we assume that the tested data are 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. The 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 tests are listed in Appendix A.

The reference implementation is not used in RTT because it is considerably slower than its optimized counterparts. The faster implementation called NIST STS optimised v5.0.2 used in RTT was developed by Zdeněk Říha and Marek Sýs(cit.).

3

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. The tests, along with their default settings, are listed in Appendix B.

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. We modified Dieharder v3.31.1 which is the last stable maintained release on the project page.

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. The tests included in specific batteries are listed in TestU01 documentation(cit.).

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 batteries.

Since TestU01 v1.2.3 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 themselves 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.

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

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

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.

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.

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

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 is equal to 1. More specifically, H_0 will be rejected for all p-values that falls outside of the interval $[\alpha, 1)$.
- 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 on 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, and an execution and interpretaion of the analysis results.

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

Therefore we developed Randomness Testing Toolkit[2], a tool 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 or modified freely.

In the 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 can change in every run; e.g. 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 of the cases, the settings stay the same for all executions of the toolkit regardless of the battery or data used. Changes in this configuration are needed only when the environment changes.

---

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 vary among the executions of the toolkit.
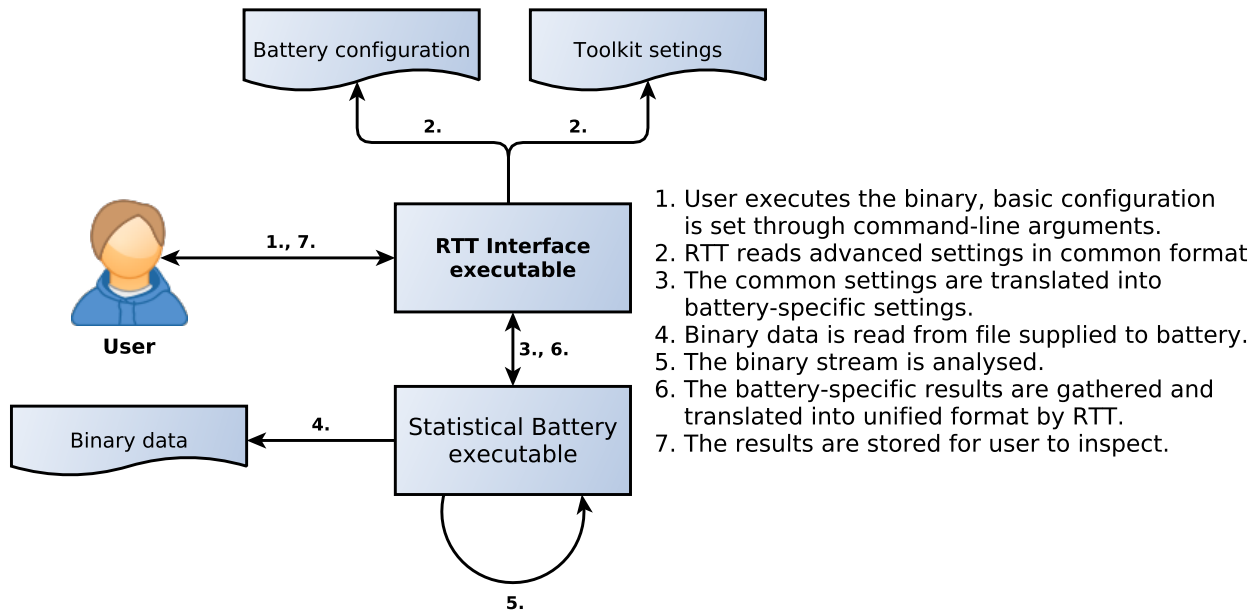
Figure 3.1: Local RTT workflow

3.1.1 Command-line arguments

The basic options of the RTT execution are set through command-line arguments. The executable 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_block-alphabit.
- `-t <test-id>` Optional argument. If set, only single test with id `<test-id>` will be executed in the chosen 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 the file with the binary data that will be analyzed by the chosen 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 is mandatory only when `db-mysql` is set as a result storage; the option is ignored otherwise. Sets ID of the experiment in the database that will be assigned the results of the toolkit execution.

3.1.2 Toolkit settings

The general settings of the toolkit are kept in separate file `rtt-settings.json`. The settings are related to the environment in which the toolkit will be executed – locations of the log files and executables, result storage options and toolkit execution options

The file is created by the user after the toolkit installation and stays the same for all of the 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 of the executable. Basic structure of the file is in Figure 3.2.

The individual tags and their descriptions are listed here. For detailed overview of the options that can be set within individual tags see Appendix C; for a complete example see Appendix D.

toolkit-settings/logger
Tag containing settings related to the location of log files produced during runtime.
Settings: `dir-prefix`, `run-log-dir`, `<battery>-dir`

toolkit-settings/result-storage/file
Section with settings needed for file result storage. For details about the storage see Section 3.2.1.
Settings: `main-file`, `dir-prefix`, `<battery>-dir`

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.
Settings: `address`, `port`, `name`, `credentials-file`

toolkit-settings/binaries
Section with the locations of the executables of the batteries.
Settings: `<battery>`

toolkit-settings/miscelaneous/nist-sts
Miscellaneous constant settings related to NIST STS battery.
Settings: `main-result-dir`

toolkit-settings/execution
Settings related to the execution of the statistical batteries.
Settings: `max-parallel-tests`, `test-timeout-seconds`

```json
1  {
2    "toolkit-settings": {
3      "logger": {
4        "comment": "Program logger settings",
5        "option": "value", ...
6      },
7      "result-storage": {
8        "file": {
9          "comment": "File result storage settings",
10         "option": "value", ...
11       },
12       "mysql-db": {
13         "comment": "MySQL Database storage settings",
14         "option": "value", ...
15       }
16     },
17     "binaries": {
18       "comment": "Executable binaries locations",
19       "option": "value", ...
20     },
21     "miscelaneous": {
22       "nist-sts": {
23         "comment": "Miscellaneous NIST STS settings",
24         "option": "value", ...
25       }
26     },
27     "execution": {
28       "comment": "Battery execution settings",
29       "option": "value", ...
30     }
31   }
32  }
```

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

3.1.3 Battery configuration

The file with battery configuration defines all settings for the battery and the tests that will be executed during 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 3.3.

11

A single configuration file can contain settings for multiple batteries. This allows the user to have only one configuration for all of his experiments. In each battery-specific section, the user can set the default values of settings for all the tests in the battery. Additionaly, settings specific only to certain tests or test variants can be also defined.

The complete guide to all common and battery specific settings is in Appendix E; for example of possible battery configuration see Appendix F.

```json
 1  {
 2    "randomness-testing-toolkit": {
 3      "<battery>-settings": {
 4        "comment": "Section with <battery> settings",
 5        "defaults": {
 6          "comment": "Default settings in <battery>",
 7          "option": "value", ...
 8        },
 9        "test-specific-settings": [
10          {
11            "test-id": "<id>",
12            "comment": "Settings specific to single test <id>",
13            "option": "value", ...
14            "variants": [
15              {
16                "comment": "Settings of the first "
17                           "variant of test <id>",
18                "option": "value", ...
19              },
20              {
21                "comment": "Settings of the second "
22                           "variant of test <id>",
23                "option": "value", ...
24              }
25            ]
26          }
27        ]
28      }
29    }
30  }
```

Figure 3.3: Basic structure of battery configuration file

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

```
rtt-results
 ├── table.txt
 └── reports
      ├── dieharder
      │    ├── <datetime>-<input-filename>-report.txt
      │    └── ⋮
      ├── nist-sts
      │    └── ⋮
      └── testu01
           ├── smallcrush
           │    └── ⋮
           ├── crush
           ├── bigcrush
           ├── rabbit
           ├── alphabit
           ├── blockalphabit
           ├── smallcrush
           └── smallcrush
```
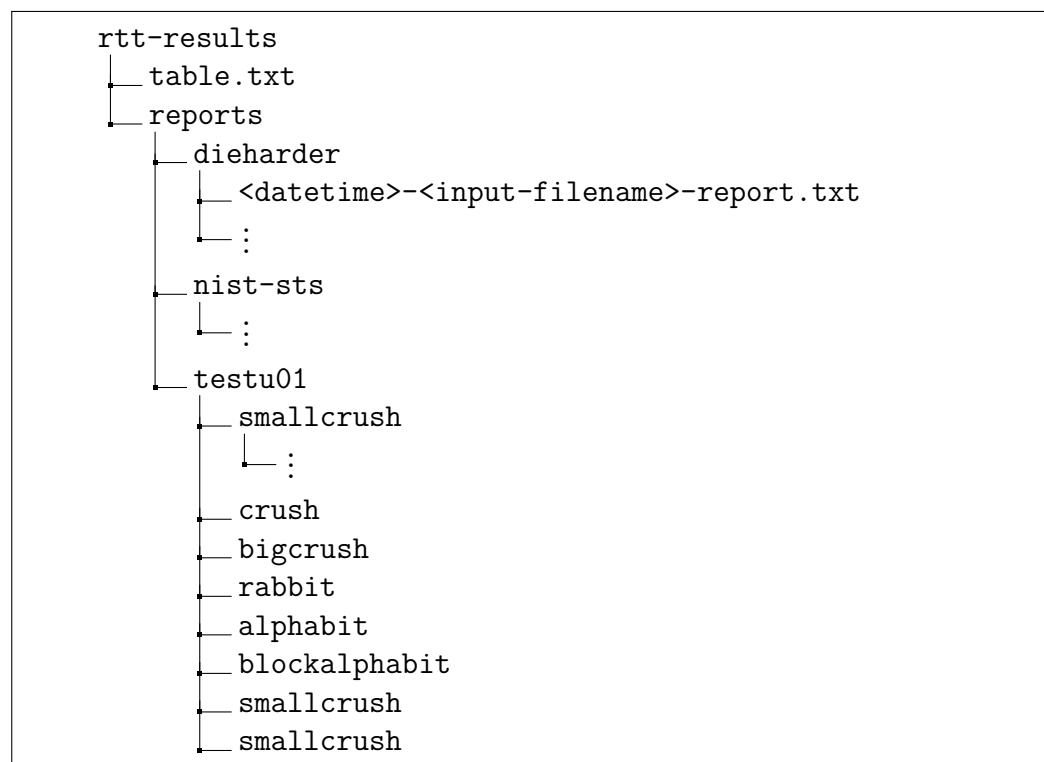
Figure 3.4: Generated files following settings from Appendix D

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 3.4. The newly generated files will have names based on the time of the execution and the filename of the binary data that was analysed. Possible instance of a report file is shown in Appendix G.

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

In the file, there is a 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 analysed multiple times by a single battery, the results are overwritten.

3.2.2 MySQL database storage

When using the database storage, RTT writes the results of the run directly into MySQL database. Layout of the database is shown in Figure 3.5. 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.

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 experiment to which they belong. Single job represents a single execution of RTT.

batteries
Each row represents results of a single execution of RTT. The analysis results are written to this table and its sub-tables by the database storage. The results of multiple RTT executions can be assigned to a single experiment.

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

tests
Contains information about the tests that were executed.

variants
Information about the variants of the tests that were executed.
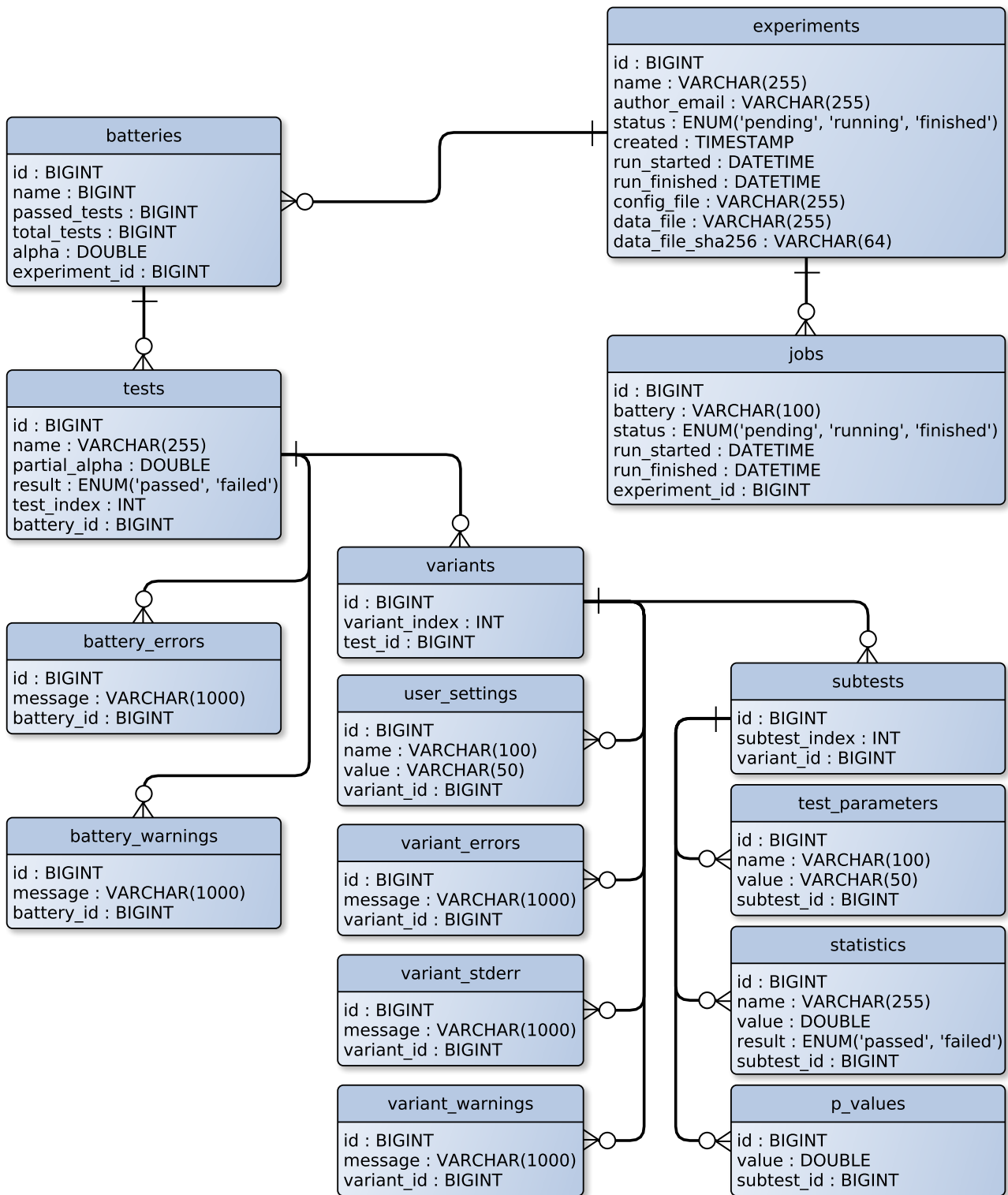
Figure 3.5: ERD Model of the database

variant_errors, variant_stderr, variant_warnings
The errors, warnings and standard error output that were extracted from the outputs of the test variants are stored in the tables.

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

subtests
Table with the subtests that were executed.

test_parameters
Options and parameters that were extracted from the outputs of the subtests.

statistics
List of all extracted statistics and their results.

p_values
List of all extracted p-values.

3.3   Remote service

Randomness Testing Toolkit can be deployed on a 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 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 analysis.

The interaction of the user with the infrastructure is visualized in Figure 3.6. In following section we outline the different purposes of the servers.

1. Experiment and data are submitted.
2. Data is uploaded.
3. Experiment and related jobs are created.
4. Single pending job is selected.
5. Data is downloaded.
6. The data is analyzed.
7. The results of the analysis are stored.
8. The results in the database are presented.
9. Analysis of the data is inspected.
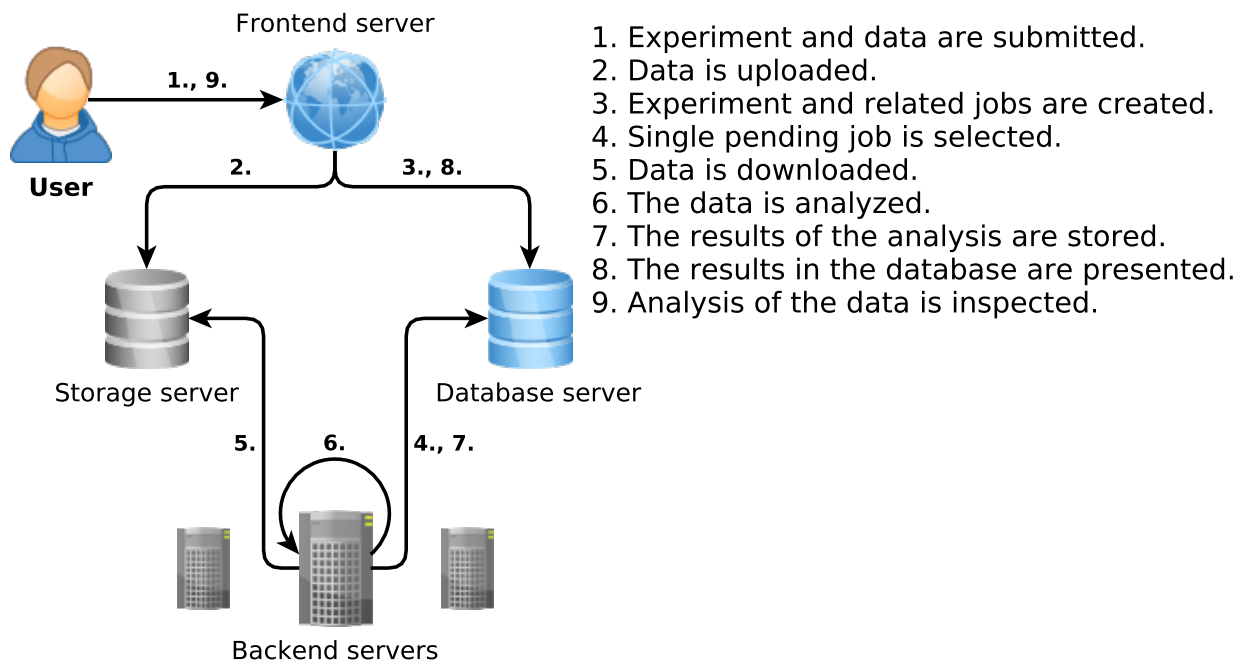
Figure 3.6: Interaction between user and servers

3.3.1 Database server

The machine hosting MySQL server with created database for RTT as shown in Figure 3.5. The database is used for storing the results of the analyses as well as the 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 the installed toolkit and statistical batteries. The analysis of the data is executed on these machines. If there

are multiple backend servers, the computations 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 3.7. 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 3.7: 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 user interacts only with the frontend server. The machine accepts data and requests 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 administrator. 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 the 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 their inclusion 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 in a single file 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 of the 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 is 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 with 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 its own expiration date and is unusable after this date.

Experiment submission

To create new experiment the user will have to fill-out following settings in the online form. The screenshot of the form is included in the Appendix H.

- **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 battery 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. We denote this proportion as x. Then the probability $P(x)$ 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 3.8. Based on the probability, the possible assessments are as follows.

- **FAIL** ($0 < P(x) < 0.001$) – The result is deemed as a failure and the source of the data should not be considered as a quality randomness generator.

- **Suspect** $(0.001 \leq P(x) < 0.01)$ – 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** $(0.01 \leq P(x) < 1)$ – 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 3.8: Experiment evaluation

4 Analysing the cryptographic primitives outputs

In the chapter we present the results of two distinct exeriments and describe the methods and battery configurations used for obtaining the results.

In the first experiment, we analysed large quantities of random data with the statistical batteries implemented in RTT. The motivation for the experiment was that the results of the multiple repeated analyses would provide us with a reference for our further experiments. Our subsequent experiments would differ from the baseline experiment only in the input data but not in the settings of the respective batteries.

In the second experiment, outputs of 15 distinct cryptographic algorithms were analysed. The specific algorithms were chosed based either on their popularity or on their success in cryptographic competitions eSTREAM (cit.) and SHA3 (cit.); among the chosen functions are algorithms such as AES, DES, RC4 or Keccak. The results of the experiment were compared with results of the analysis in EACirc framework. The EACirc framework implements alternative approach to the randomness testing and is developed in CRoCS (cit.).

4.1 Battery configurations used in the experiments

In this section we will present the configurations of the respective batteries that were used in both above-mentioned experiments. The settings were kept as close as possible to the default settings of the respective batteries; because of this, the results of the batteries executed through RTT are comparable to the results of the batteries when directly executed. The battery configuration file that was used in RTT is in Appendix F.

4.1.1 Common settings

All batteries were provided 8000 MiB[1] long data files for the analysis in each execution. The batteries were not required to process the entirety of the data; however rewinding the file and reading more bytes was not permitted to the batteries. The file size was chosen as a compromise between the amount of data needed by the separate tests in the batteries and our capabilities to process and store such quantities of data.

---

1. The exact length of single file was $8000 \times 1024^2 = 8388608000$ bytes.

NIST Statistical Testing Suite

The settings of the tests in NIST STS was the same as in default execution of the battery. The tests along with their default settings are listed in Appendix A. In single run, each test processed 1000 data streams of size 1000000 bits; in total, each test analysed 125 MB of data in its run.

Dieharder

The test settings of the Dieharder battery follow default settings outlined in Appendix B with the single exception of test RGB Lagged Sums. In default settings, the test would need too much data[2] for the analysis.

The RGB Lagged Sum test is executed in 33 variants with variable parameter n={0..32}. The number of bytes the test variant will process scales with the parameter n of the variant and number of times the test variant will be repeated (p-samples) during the battery execution. To lower the amount of the needed bytes, we reduced the number of repetitions for each respective variant based on its n. The number of repetitions was calculated using following function.

$$\text{Rep}(n) = \min\left(100; \left\lfloor \frac{8000 \times 1024^2}{(n+1) \times 4 \times 10^6} \right\rfloor\right)$$

TestU01

The batteries Small Crush and Crush are executed in their default settings. The default settings of the tests in their respective batteries can be found in TestU01 documentation (cit.).

The battery Big Crush is omitted from our analysis because the tests in the battery would need at least 60 GB of data to be fully executed which is an infeasible amount.

The Rabbit, Alphabit and Block Alphabit batteries are executed in default test settings. The tests are limited to 8000 MiB of data by the parameter nb.

The tests in the batteries Alphabit and Block Alphabit have settings that would cause them to process only certain parts of each 32 bit block in the data. These settings are not used and all bits in 32 bit blocks are analysed by the tests. Additionally, all tests from Block Alphabit are executed in 6 different variants; each test variant analyses differently reordered blocks of 32 bits.

---

2. RGB Lagged Sum test with parameter n=32 will analyse 13.2 GB of data.

4.2   Baseline experiment

Aim of this experiment was to observe behavior and results of the batteries in RTT when analysing truly random data. The observation would be then used for the interpretation of results of our other experiments.

The evaluation of the battery result is based on the number of failed tests in the battery. Under the assumption that we have single second-level p-value per single test, the test is considered failed when its p-value is outside of the interval $[\alpha, 1)$ and its null hypothesis H_0 is rejected. The reason why we chose such interval is explained is Section 2.3.1.

When analysing truly random data repeatedly with a statistical test, we will observe that the resulting p-values are uniformly distributed on the interval $[0, 1]$. The uniformity is caused by the fact that we can't predict output of a TRNG and each possible output is observed with equal probability. Therefore we also can't predict result of the analysis with a statistical test. Since the p-values are uniformly distributed, they will fall outside of the interval $[\alpha, 1)$ with probability equal to α; hence test's H_0 is rejected and the test fails with rate equal to significance level α. Rejection of H_0, when it actually holds true, is called **Type I error** (cit.). The theoretical probability of Type I error should be equal to the value of the significance level.

When we are evaluating a battery analysis based on number of failed tests, we must know how many tests we expect to fail when the H_0 holds true. We can calculate the expected value like so. Let n be the number of tests in a battery. Then we have n independent experiments that will result in either yes (failure of the test) with probability α or no (success of the test) with probability $1 - \alpha$. This situation can be modeled by using binomial distribution (cit.) $B(n, \alpha)$. Let F_{Exp} be a random discrete variable expressing number of failed tests in a battery with expected distribution $B(n, \alpha)$. From the definition of the binomial distribution, the expected number of the failed tests is $E(F) = n \times \alpha$.

However, the expected number of failures is not necesarilly exact, as the above-mentioned calculation treats tests as a independent trials. Separate tests in a battery are not required nor proved to be independent, which can lead to differing distributions of empirical and theoretical random variables representing number of failed tests.

To observe empirical distribution of the number of failed tests F_{Obs} we inspected the number of failures of the respective batteries during analysis of truly random data. In total, approximately 8 terabytes of random data were downloaded from QRNG service provided by Humboldt-Universitaet zu Berlin (cit.). The data were split into 1000 blocks each of size 8000 GiB. All 7 batteries included in this experiment analysed each block of data once. Sample size of 1000 trials is enough for us to compare the distributions of the two random variables F_{Exp} and F_{Obs}.

The counts of failures of respective batteries were inspected in following two scenarios.

Raw second-level p-values failures
In this scenario, we assumed that all second-level p-values produced by a specific battery are independent. Therefore, all statistics calculated by the battery were treated as separate tests. For example, if some test is executed in two variants, each of those variants has two subtests and each subtest will calculate 2 statistics, in total we will get $2^3 = 8$ second-level p-values; we consider these eight p-values as the results of eight independent tests each of whose calculated single p-value.

Corrected p-values failures
Whit this second scenario, we tried to mitigate the effect of dependent p-values in our set by grouping them together. We put the p-values that are most likely dependent into single set or group. We assumed that the p-values that are most likely inter-connected are those that are produced by a single statistical test. Therefore, all p-values belonging to a single test were reduced to only one p-value, leaving us with a single result per distinct test in a statistical battery. For the grouping of the p-values we used Šidák's correction (cit.) and the process is shown in Definition 4.2.1.

Definition 4.2.1. For n grouped p-values $\{p_1, .., p_n\}$ with significance level α and null hypothesis H_0 we will calculate the corrected significance level $\alpha_0 = 1 - (1 - \alpha)^{\frac{1}{n}}$. If $\forall i \in [1, .., n], \exists p_i : p_i < \alpha_0$, then H_0 is rejected for $\{p_1, .., p_n\}$ and the whole group result is treated as a failure.

TODO: tell them about observed and expected counts and chi-square comparison

4.3 Results of the analysis

- Testing is the same as before baby
- Show bigass table with results - compare to EACirc
- Conlusions - limits of EACirc, limits of RTT, differences, Rabbit, Grain are lit af.

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 NIST STS Tests

Common parameters of all tests

- **Stream size** – 1000000
- **Stream count** – 1000

ID	Test name	Block size	Subtests
1	Frequency (Monobit)	–	1
2	Frequency within a Block	128	1
3	Cumulative Sums (Cusums)	–	2
4	Runs	–	1
5	Longest Run of Ones in a Block	–	1
6	Random Binary Matrix Rank	–	1
7	Discrete Fourier Transform (Spectral)	–	1
8	Non-overlapping (Aperiodic) Template Matching	9	148
9	Overlapping (Periodic) Template Matching	9	1
10	Maurer's "Universal Statistical"	–	1
11	Approximate Entropy	10	1
12	Random Excursions	–	8
13	Random Excursions Variant	–	18
14	Serial	16	2
15	Linear Complexity	500	1

B Dieharder Tests

Note: Tests labeled as not used are excluded from our experiments; they are flagged as suspicious or not working in Dieharder documentation. Tests with plus sign next to the stream size will process variable amount of data on each run. The value is only orientational.

ID	Test name	Stream size (byte)	P-samples	Arguments
Diehard tests				
0	Birthdays	153 600	100	–
1	OPERM5	4 000 020	100	–
2	32×32 Binary Rank	5 120 000	100	–
3	6×8 Binary Rank	2 400 000	100	–
4	Bitstream	1 048 584	100	–
5	OPSO **(Not used)**	–	–	–
6	OQSO **(Not used)**	–	–	–
7	DNA **(Not used)**	–	–	–
8	Count the 1s (stream)	256 004	100	–
9	Count the 1s (byte)	5 120 000	100	–
10	Parking Lot	96 000	100	–
11	Minimum Distance (2D Circle)	64 000	100	–
12	Minimum Distance (3D Sphere)	48 000	100	–
13	Squeeze	10 000 000+	100	–
14	Sums **(Not used)**	–	–	–
15	Runs	400 000	100	–
16	Craps	10 000 000+	100	–
Marsaglia and Tsang tests				
17	GCD	80 000 000	100	–
STS tests				
100	Monobit	400 000	100	–
101	Runs	400 000	100	–
102	Serial (Generalized)	400 000	100	–
RGB tests				
200	Bit Distribution	$n \times 800\,000 + 4$	100	-n {1..12}
201	Generalized Minimum Distance	$n \times 40\,000$	100	-t 10000 -n {2..5}
202	Permutations	$n \times 400\,000$	100	-n {2..5}
203	Lagged Sum	$(n{+}1) \times 4\,000\,000$	100	-n {0..32}
204	Kolmogorov-Smirnov	40 000	1000	–
DAB tests				
205	Byte Distribution	614 400 000	1	–
206	DCT	51 200 000	1	–
207	Fill Tree	500 000 000+	1	–
208	Fill Tree 2	200 000 000+	1	–
209	Monobit 2	260 000 000	1	–

C Overview of options in toolkit settings

The tags that are listed in the top-level description must contain JSON object with key-value pairs with options and their values. For complete example of the JSON see Appendix D.

toolkit-settings/logger

- **dir-prefix** *Optional* – If set, value of this tag will become prefix of all log files locations. Can be used when all logger directories should have the same parent directories.
- **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 their values set locations of files that will contain raw outputs of the respective executed batteries.

toolkit-settings/result-storage/file *Optional*
For details about the file 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/result-storage/mysql-db *Optional*
For details about the MySQL database 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 the file that contains login information for the database. The example credentials file is shown in Figure C.1.

toolkit-settings/binaries

- **<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

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

toolkit-settings/execution

- **max-parallel-tests** Sets maximum number of concurrently running test processes. The higher value may cause the analysis to finish faster but will cause higher strain on the system.
- **test-timeout-seconds** Sets time period after which the running tests will be considered stuck and will be killed.

```
1 {
2   "credentials": {
3     "username": "jane_doe",
4     "password": "password"
5   }
6 }
```

Figure C.1: Example of MySQL database credentials file

D Sample file with RTT settings

```json
1  {
2    "toolkit-settings": {
3      "logger": {
4        "dir-prefix": "rtt-results/logs",
5        "run-log-dir": "run-logs",
6        "dieharder-dir": "dieharder",
7        "nist-sts-dir": "niststs",
8        "tu01-smallcrush-dir": "testu01/smallcrush",
9        "tu01-crush-dir": "testu01/crush",
10       "tu01-bigcrush-dir": "testu01/bigcrush",
11       "tu01-rabbit-dir": "testu01/rabbit",
12       "tu01-alphabit-dir": "testu01/alphabit",
13       "tu01-blockalphabit-dir": "testu01/blockalphabit"
14     },
15     "result-storage": {
16       "file": {
17         "main-file": "rtt-results/table.txt",
18         "dir-prefix": "rtt-results/reports",
19         "dieharder-dir": "dieharder",
20         "nist-sts-dir": "niststs",
21         "tu01-smallcrush-dir": "testu01/smallcrush",
22         "tu01-crush-dir": "testu01/crush",
23         "tu01-bigcrush-dir": "testu01/bigcrush",
24         "tu01-rabbit-dir": "testu01/rabbit",
25         "tu01-alphabit-dir": "testu01/alphabit",
26         "tu01-blockalphabit-dir": "testu01/blockalphabit"
27       },
28       "mysql-db": {
29         "address": "127.0.0.1",
30         "port": "3306",
31         "name": "rtt",
32         "credentials-file": "credentials.json"
33       }
34     },
35     "binaries": {
36       "nist-sts": "/home/rtt-statistical-batteries/nist-sts",
37       "dieharder": "/home/rtt-statistical-batteries/dieharder",
38       "testu01": "/home/rtt-statistical-batteries/testu01"
39     },
40     "miscelaneous": {
41       "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 }
```

E Guide to the settings in battery configuration

In following list we describe common as well as battery specific tags that can be present in battery configuration file. Each tag is listed with its description and list of its allowed parent tags. For example of complete configuration see Appendix F.

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. These tags contain settings for respective batteries.

`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 the 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 the 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 in the test object, 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 object that is in `test-specific-settings` object list. The value of the tag is used for assigning the options in the test object to actual test from the battery.

variants
Possible parent tags: `test-specific-settings`
The tag can be present in the test 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. For example, 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 will be 100, then the test will be repeated 100 times, each time with different input data of length `stream-size`.

block-length
Possible parent tags: `defaults`, `test-specific-settings`, `variants`
Some tests in NIST STS analyse the data streams in blocks. This option sets the bit size of the blocks. For details on which tests accept this parameter, see Appendix A.

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 (e.g. `"-x 2048 -y 30"`). For summary of arguments accepted 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 option to more that 1 will produce several multiple of the test.

`bit-nb`
Possible parent tags: `defaults, test-specific-settings, variants`
This option is valid only for 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 for 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 for 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 for 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 (TODO cite).

`parameters`
Possible parent tags: `defaults, test-specific-settings, variants`
This option is valid only for 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 in respective batteries can be found in TestU01 documentation (cit.). Example of the `parameters` tag for test smars_BirthdaySpacings is shown in Figure E.1.

```
1  {
2    "parameters": {
3      "N": "1",
4      "n": "10000",
5      "r": "0",
6      "d": "1000000",
7      "t": "2",
8      "p": "1"
9    }
10 }
```

Figure E.1: Parameters setting for smarsa_BirthdaySpacings test

F Example of battery configuration

Following battery configuration was used in both experiments described in Chapter 4.

```
1  {
2    "randomness-testing-toolkit":{
3      "dieharder-settings":{
4        "defaults":{
5          "test-ids":["0-4", "8-13", "15-17", "100-102", "200-209"],
6          "psamples":100
7        },
8        "test-specific-settings":[
9          {
10           "test-id":200,
11           "variants":[
12             {"arguments":"-n 1"},
13             {"arguments":"-n 2"},
14             {"arguments":"-n 3"},
15             {"arguments":"-n 4"},
16             {"arguments":"-n 5"},
17             {"arguments":"-n 6"},
18             {"arguments":"-n 7"},
19             {"arguments":"-n 8"},
20             {"arguments":"-n 9"},
21             {"arguments":"-n 10"},
22             {"arguments":"-n 11"},
23             {"arguments":"-n 12"}
24           ]
25         },
26         {
27           "test-id":201,
28           "psamples":1000,
29           "variants":[
30             {"arguments":"-n 2 -t 10000"},
31             {"arguments":"-n 3 -t 10000"},
32             {"arguments":"-n 4 -t 10000"},
33             {"arguments":"-n 5 -t 10000"}
34           ]
35         },
36         {
37           "test-id":202,
38           "variants":[
39             {"arguments":"-n 2"},
40             {"arguments":"-n 3"},
41             {"arguments":"-n 4"},
```

```json
42          {"arguments":"-n 5"}
43        ]
44      },
45      {
46        "test-id":203,
47        "variants":[
48          {"arguments":"-n 0"},
49          {"arguments":"-n 1"},
50          {"arguments":"-n 2"},
51          {"arguments":"-n 3"},
52          {"arguments":"-n 4"},
53          {"arguments":"-n 5"},
54          {"arguments":"-n 6"},
55          {"arguments":"-n 7"},
56          {"arguments":"-n 8"},
57          {"arguments":"-n 9"},
58          {"arguments":"-n 10"},
59          {"arguments":"-n 11"},
60          {"arguments":"-n 12"},
61          {"arguments":"-n 13"},
62          {"arguments":"-n 14"},
63          {"arguments":"-n 15"},
64          {"arguments":"-n 16"},
65          {"arguments":"-n 17"},
66          {"arguments":"-n 18"},
67          {"arguments":"-n 19"},
68          {"arguments":"-n 20", "psamples":99},
69          {"arguments":"-n 21", "psamples":95},
70          {"arguments":"-n 22", "psamples":91},
71          {"arguments":"-n 23", "psamples":87},
72          {"arguments":"-n 24", "psamples":83},
73          {"arguments":"-n 25", "psamples":80},
74          {"arguments":"-n 26", "psamples":77},
75          {"arguments":"-n 27", "psamples":74},
76          {"arguments":"-n 28", "psamples":72},
77          {"arguments":"-n 29", "psamples":69},
78          {"arguments":"-n 30", "psamples":67},
79          {"arguments":"-n 31", "psamples":65},
80          {"arguments":"-n 32", "psamples":63}
81        ]
82      },
83      {
84        "test-id":204,
85        "psamples":1000
86      },
87      {
88        "test-id":205,
89        "psamples":1
```

```
 90          },
 91          {
 92            "test-id":206,
 93            "psamples":1
 94          },
 95          {
 96            "test-id":207,
 97            "psamples":1
 98          },
 99          {
100            "test-id":208,
101            "psamples":1
102          },
103          {
104            "test-id":209,
105            "psamples":1
106          }
107        ]
108      },
109
110      "nist-sts-settings":{
111        "defaults":{
112          "test-ids":["1-15"],
113          "stream-size":"1000000",
114          "stream-count":"1000"
115        }
116      },
117
118      "tu01-smallcrush-settings":{
119        "defaults":{
120          "test-ids":["1-10"],
121          "repetitions":1
122        }
123      },
124
125      "tu01-crush-settings":{
126        "defaults":{
127          "test-ids":["1-96"],
128          "repetitions":1
129        }
130      },
131
132      "tu01-rabbit-settings":{
133        "defaults":{
134          "test-ids":["1-26"],
135          "repetitions":1,
136          "bit-nb":"8388608000"
137        }
```

```
138        },
139
140        "tu01-alphabit-settings":{
141          "defaults":{
142            "test-ids":["1-9"],
143            "repetitions":1,
144            "bit-nb":"8388608000",
145            "bit-r":"0",
146            "bit-s":"32"
147          }
148        },
149
150        "tu01-blockalphabit-settings":{
151          "defaults":{
152            "test-ids":[
153              "1-9"
154            ],
155            "repetitions":1,
156            "bit-nb":"8388608000",
157            "bit-r":"0",
158            "bit-s":"32"
159          },
160          "test-specific-settings":[
161            {
162              "test-id":1,
163              "variants":[
164                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
165                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
166              ]
167            },
168            {
169              "test-id":2,
170              "variants":[
171                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
172                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
173              ]
174            },
175            {
176              "test-id":3,
177              "variants":[
178                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
179                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
180              ]
181            },
182            {
183              "test-id":4,
184              "variants":[
185                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
```

```
186                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
187            ]
188        },
189        {
190            "test-id":5,
191            "variants":[
192                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
193                {"bit-w":"8"},{"bit-w":"16"}, {"bit-w":"32"}
194            ]
195        },
196        {
197            "test-id":6,
198            "variants":[
199                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
200                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
201            ]
202        },
203        {
204            "test-id":7,
205            "variants":[
206                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
207                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
208            ]
209        },
210        {
211            "test-id":8,
212            "variants":[
213                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
214                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
215            ]
216        },
217        {
218            "test-id":9,
219            "variants":[
220                {"bit-w":"1"}, {"bit-w":"2"}, {"bit-w":"4"},
221                {"bit-w":"8"}, {"bit-w":"16"}, {"bit-w":"32"}
222            ]
223        }
224    ]
225   }
226  }
227 }
```

G 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
      ============
  ###########

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


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

H Experiment submission online form

General

Experiment name

E-Mail

Binary data that will be analysed

Choose File   No file chosen

The data is deleted after the analysis.

Configuration

☐ Use default configuration (recommended)

Choose configuration

----------   ▼

Configuration file (advanced user)

Choose File   No file chosen

For configuration file description read our wiki.

Battery application

☐ **Switch all**

☐ NIST Statistical Testing Suite

☐ Dieharder

☐ TestU01 Small Crush

☐ TestU01 Crush

☐ TestU01 Rabbit

☐ TestU01 Alphabit

☐ TestU01 Block Alphabit

☐ TestU01 Big Crush (requires at least 20GB of data, 60GB for full run)

Submit

You will be notified by email when the experiment finishes.