ERNIE: a tool to study the Tor network – User's Guide –

by Karsten Loesing <karsten@torproject.org>

March 29, 2010

1 Overview

Welcome to ERNIE! ERNIE is a tool to study the Tor network. ERNIE has been designed to process all kinds of data about the Tor network and visualize them or prepare them for further analysis. ERNIE is also the software behind the Tor Metrics Portal `http://metrics.torproject.org/`.

The acronym ERNIE stands for the *Enhanced R-based tor Network Intelligence Engine* (sorry for misspelling Tor). Why ERNIE? Because nobody liked BIRT (Business Intelligence and Reporting Tools) that we used for visualizing statistics about the Tor network before writing our own software. By the way, reasons were that BIRT made certain people's browsers crash and requires JavaScript that most Tor user have turned off.

If you want to learn more about the Tor network, regardless of whether you want to present your findings on a website (like ERNIE does) or include them in your next Tor paper, this user's guide is for you!

2 Getting started with ERNIE

The ERNIE project was started as a simple tool to parse Tor relay descriptors and plot graphs on Tor network usage for a website. Since then, ERNIE has grown to a tool that can process all kinds of Tor network data for various purposes, including but not limited to visualization.

We think that the easiest way to get started with ERNIE is to walk through typical use cases in a tutorial style and explain what is required to set up ERNIE. These use cases have been chosen from what we think are typical applications of ERNIE.

2.1 Visualizing network statistics

Write me.

2.2 Importing relay descriptors into a database

As of February 2010, the relays and directories in the Tor network generate more than 1 GB of descriptors every month. There are two approaches to process these amounts of data: extract only the relevant data for the analysis and write them to files, or import all data to a database and run queries on the database. ERNIE currently takes the file-based approach for the Metrics Portal, which works great for standardized analyses. But the more flexible way to research the Tor network is to work with a database.

This tutorial describes how to import relay descriptors into a database and run a few example queries. Note that the presented database schema is limited to answering basic questions about the Tor network. In order to answer more complex questions, one would have to extend the database schema and Java classes which is sketched at the end of this tutorial.

2.2.1 Preparing database for data import

The first step in importing relay descriptors into a database is to install a database management system. We won't go into the details of installing a database for the various operating systems in this tutorial. Please consult the tutorials and manuals that are out on the Web. For this tutorial, we assume that you have PostgreSQL 8.4 installed. Note that in theory, any other relational database that has a working JDBC 4 driver should work, too, possibly with minor modifications to ERNIE. We further assume a database user called `ernie` that is allowed to define, modify, and query database objects.

First, create a new database schema `tordir` with two tables that we need for importing relay descriptors, plus two indexes to accelerate queries. Note that `$` denotes a shell prompt and `tordir=>` the database prompt.

```
$ createdb -U ernie -O ernie tordir
$ psql -U ernie tordir
tordir=> CREATE TABLE statusentry (
  validafter TIMESTAMP NOT NULL,
  descriptor CHAR(40) NOT NULL,
  isauthority BOOLEAN NOT NULL DEFAULT false,
  isbadexit BOOLEAN NOT NULL DEFAULT false,
  isbaddirectory BOOLEAN NOT NULL DEFAULT false,
  isexit BOOLEAN NOT NULL DEFAULT false,
  isfast BOOLEAN NOT NULL DEFAULT false,
  isguard BOOLEAN NOT NULL DEFAULT false,
  ishsdir BOOLEAN NOT NULL DEFAULT false,
  isnamed BOOLEAN NOT NULL DEFAULT false,
  isstable BOOLEAN NOT NULL DEFAULT false,
  isrunning BOOLEAN NOT NULL DEFAULT false,
  isunnamed BOOLEAN NOT NULL DEFAULT false,
  isvalid BOOLEAN NOT NULL DEFAULT false,
  isv2dir BOOLEAN NOT NULL DEFAULT false,
  isv3dir BOOLEAN NOT NULL DEFAULT false,
  PRIMARY KEY (validafter, descriptor));
tordir=> CREATE TABLE descriptor (
  descriptor CHAR(40) NOT NULL PRIMARY KEY,
  address VARCHAR(15) NOT NULL,
  orport INTEGER NOT NULL,
  dirport INTEGER NOT NULL,
  bandwidthavg BIGINT NOT NULL,
  bandwidthburst BIGINT NOT NULL,
  bandwidthobserved BIGINT NOT NULL,
  platform VARCHAR(256),
  published TIMESTAMP NOT NULL,
  uptime BIGINT);
tordir=> CREATE INDEX statusvalidafter
  ON statusentry (validafter);
tordir=> CREATE INDEX descriptorid
  ON descriptor (descriptor);
tordir=> \q
```

A row in the `statusentry` table contains the information that a given relay (that has published the server descriptor with ID `descriptor`) was contained in the network status consensus published at time `validafter`. These two fields uniquely identify a row in the `statusentry` table. The other fields contain boolean values for the flags that the directory authorities assigned to the relay in this consensus, e.g., the Exit flag in `isexit`. Note that for the 24 network status consensuses of a given day with each of them containing 2000 relays, there will be 24×2000 rows in the `statusentry` table.

The `descriptor` table contains some portion of the information that a relay includes in its server descriptor. Descriptors are identified by the `descriptor` field which corresponds to the `descriptor` field in the `statusentry` table. The other fields contain further data of the server descriptor that might be relevant for analyses, e.g., the platform line with the Tor software version and operating system of the relay.

Obviously, this data schema doesn't match everyone's needs. See the instructions below for extending ERNIE to import other data into the database.

2.2.2 Downloading relay descriptors from the metrics website

In the next step you will probably want to download relay descriptors from the metrics website `http://metrics.torproject.org/data.html#relaydesc`. Download the `v3 consensuses` and/or `server descriptors` of the months you want to analyze. The server descriptors are the documents that relays publish at least every 18 hours describing their capabilities, whereas the v3 consensuses are views of the directory authorities on the available relays at a given time. For this tutorial you need both v3 consensuses and server descriptors. You might want to start with a single month of data, experiment with it, and import more data later on. Extract the tarballs to a new directory `archives/` in the ERNIE working directory.

2.2.3 Configuring ERNIE to import relay descriptors into a database

ERNIE can be used to read data from one or more data sources and write them to one or more data sinks. You need to configure ERNIE so that it knows to use the downloaded relay descriptors as data source and the database as data sink. You have implicitly accomplished the former by creating the `archives/` directory. By default, ERNIE looks for this directory and tries to import everything contained in it. You could change this behavior by explicitly telling ERNIE not to import data from the `archives/` directory by adding a line `ImportDirectoryArchives 0` to the config file, but this is not what we want in this tutorial. But you need to explicitly enable your database as a data sink. Add the following line to your `config` file:

```
WriteRelayDescriptorDatabase 1
```

You further need to provide the JDBC string that ERNIE shall use to access the database schema `tordir` that we created above. The config option with the JDBC string for a local PostgreSQL database might be (without line break):

```
RelayDescriptorDatabaseJDBC
  jdbc:postgresql:tordir?user=ernie&password=password
```

2.2.4 Importing relay descriptors using ERNIE

Now you are ready to actually import relay descriptors using ERNIE. Compile the Java classes and run ERNIE.

```
$ ./download.sh
$ ./run.sh
```

Note that the import process might take between a few minutes and an hour, depending on your hardware. You will notice that ERNIE doesn't progress messages to the standard output. You can either change this behavior by setting `java.util.logging.ConsoleHandler.level` in `logging.properties` to `INFO` or `FINE`. Alternately, you can look at the log file `log.0` that is created by ERNIE.

If ERNIE finishes after a few seconds, you have probably put the relay descriptors at the wrong place. Make sure that you extract the relay descriptors to sub directories of `archives/` in the ERNIE working directory.

If you interrupt ERNIE, or if ERNIE terminates uncleanly for some reason, you will have problems starting it the next time. ERNIE uses a local lock file called `lock` to make sure that only a single instance of ERNIE is running at a time. If you are sure that the last ERNIE instance isn't running anymore, you can remove the lock file and start ERNIE again.

If all goes well, you should now have the relay descriptors of 1 month in your database.

2.2.5 Example queries

In this tutorial, we want to give you a few examples for using the database schema with the imported relay descriptors to extract some useful statistics about the Tor network.

In the first example we want to find out how many relays have been running on average per day and how many of these relays were exit relays. We only need the `statusentry` table for this evaluation, because the information we are interested in is contained in the network status consensuses.

The SQL statement that we need for this evaluation consists of two parts: First, we find out how many network status consensuses have been published on any given day. Second, we count all relays and those with the Exit flag and divide these numbers by the number of network status consensuses per day.

```
$ psql -U ernie tordir
tordir=> SELECT DATE(validafter),
    COUNT(*) / relay_statuses_per_day.count AS avg_running,
    SUM(CASE WHEN isexit IS TRUE THEN 1 ELSE 0 END) /
      relay_statuses_per_day.count AS avg_exit
  FROM statusentry,
    (SELECT COUNT(*) AS count, DATE(validafter) AS date
      FROM (SELECT DISTINCT validafter FROM statusentry)
      distinct_consensuses
      GROUP BY DATE(validafter)) relay_statuses_per_day
  WHERE DATE(validafter) = relay_statuses_per_day.date
  GROUP BY DATE(validafter), relay_statuses_per_day.count
  ORDER BY DATE(validafter);
tordir=> \q
```

Executing this query should finish within a few seconds to one minute, again depending on your hardware. The result might start like this (truncated here):

```
    date    | avg_running | avg_exit
------------+-------------+----------
 2010-02-01 |        1583 |      627
 2010-02-02 |        1596 |      638
 2010-02-03 |        1600 |      654
:
```

In the second example we want to find out what Tor software versions the relays have been running. More precisely, we want to know how many relays have been running what Tor versions on micro version granularity (e.g., 0.2.2) on average per day?

We need to combine network status consensuses with server descriptors to find out this information, because the version information is not contained in the consensuses (or at least, it's optional to be contained in there; and after all, this is just an example). Note that we cannot focus on server descriptors only and leave out the consensuses for this analysis, because we want our analysis to be limited to running relays as confirmed by the directory authorities and not include all descriptors that happened to be published at a given day.

The SQL statement again determines the number of consensuses per day in a sub query. In the next step, we join the `statusentry` table with the `descriptor` table for all rows contained in the `statusentry` table. The left join means that we include `statusentry` rows even if we do not have corresponding lines in the `descriptor` table. We determine the version by skipping the first 4 characters of the platform string that should contain `"Tor "` (without quotes) and cutting off after another 5 characters. Obviously, this approach is prone to errors if the platform line format changes, but it should be sufficient for this example.

```
$ psql -U ernie tordir
tordir=> SELECT DATE(validafter) AS date,
    SUBSTRING(platform, 5, 5) AS version,
    COUNT(*) / relay_statuses_per_day.count AS count
  FROM
    (SELECT COUNT(*) AS count, DATE(validafter) AS date
    FROM (SELECT DISTINCT validafter
      FROM statusentry) distinct_consensuses
    GROUP BY DATE(validafter)) relay_statuses_per_day
  JOIN statusentry
    ON relay_statuses_per_day.date = DATE(validafter)
  LEFT JOIN descriptor
    ON statusentry.descriptor = descriptor.descriptor
  GROUP BY DATE(validafter), SUBSTRING(platform, 5, 5),
    relay_statuses_per_day.count, relay_statuses_per_day.date
  ORDER BY DATE(validafter), SUBSTRING(platform, 5, 5);
tordir=> \q
```

Running this query takes longer than the first query, which can be a few minutes to half an hour. The main reason is that joining the two tables is an expensive database operation. If you plan to perform many evaluations like this one, you might want to create a third table that holds the results of joining the two tables of this tutorial. Creating such a table to speed up queries is not specific to ERNIE and beyond the scope of this tutorial.

The (truncated) result of the query might look like this:

```
    date    | version | count
------------+---------+-------
 2010-02-01 | 0.1.2   |    10
 2010-02-01 | 0.2.0   |   217
 2010-02-01 | 0.2.1   |   774
 2010-02-01 | 0.2.2   |    75
 2010-02-01 |         |   505
 2010-02-02 | 0.1.2   |    14
 2010-02-02 | 0.2.0    |    328
 2010-02-02 | 0.2.1    |   1143
 2010-02-02 | 0.2.2    |    110
:
```

Note that, in the fifth line, we are missing the server descriptors of 505 relays contained in network status consensuses published on 2010-02-01. If you want to avoid such missing values, you'll have to import the server descriptors of the previous month, too.

2.2.6 Extending ERNIE to import further data into the database

In this tutorial we have explained how to prepare a database, download relay descriptors, configure ERNIE, import the descriptors, and execute example queries. This description is limited to a few examples by the very nature of a tutorial. If you want to extend ERNIE to import further data into your database, you will have to perform at least two steps: extend the database schema and modify the Java classes used for parsing.

The first step, extending the database schema, is not specific to ERNIE. Just add the fields and tables to the schema definition.

The second step, modifying the Java classes used for parsing, is of course specific to ERNIE. You will have to look at two classes in particular: The first class, `RelayDescriptorDatabaseImporter`, contains the prepared statements and methods used to add network status consensus entries and server descriptors to the database. The second class, `RelayDescriptorParser`, contains the parsing logic for the relay descriptors and decides what information to add to the database, among other things.

This ends the tutorial on importing relay descriptors into a database. Happy researching!

2.3 Aggregating relay and bridge descriptors

Write me.

3 Software architecture

Write me. In particular, include overview of components:

- *Data sources and data sinks*
- *Java classes with data sources and data sinks*
- *R scripts to process CSV output*
- *Website*

4 Tor Metrics Portal setup

Write me. In particular, include documentation of deployed ERNIE that runs the metrics website. This documentation has two purposes: First, a reference setup can help others creating their own ERNIE configuration that goes beyond the use cases as described above. Second, we need to remember how things are configured anyway, so we can as well document them here.