TorMap documentation

P4 SS21 - Visualisierung von Tor-Knoten-Informationen

Julius Henke, Tim Kilb

August 2021

TECHNISCHE UNIVERSITÄT DARMSTADT

Contents

1 Introduction

The Tor network currently consists of thousands of nodes which route anonymous internet traffic daily. The nonprofit organization *TorProject*[1] already provides an archive with raw historic data about the network. This raw data is difficult to analyze and grasp. With our app TorMap we want to visualize, group and filter public Tor relays on a world map. The state of the network can be viewed for any day between October 2007 and today. Getting details like IP address, contact or Autonomous System info of a relay is as easy as selecting it on the map.

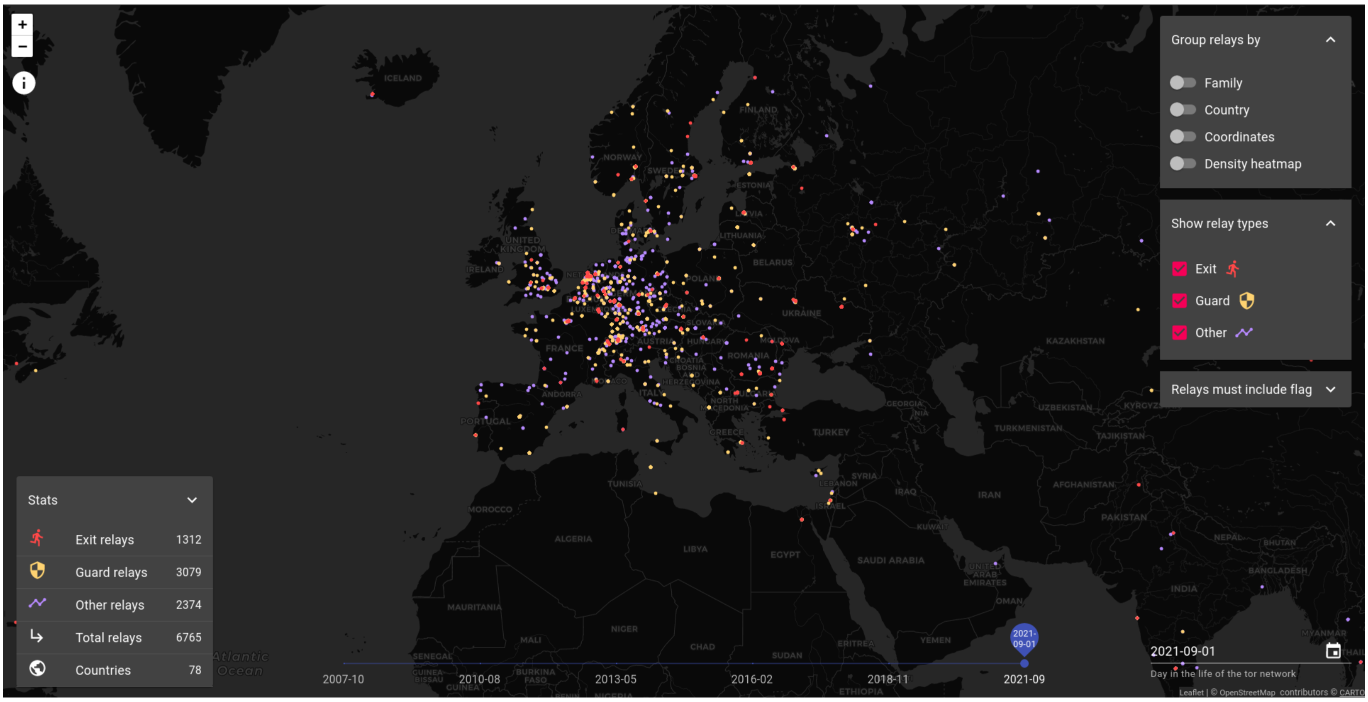

Figure 1: Screenshot of TorMap frontend

[1] https://www.torproject.org/

2 Technical documentation

TorMap consists of a *ReactJS*[2] web app (frontend) and a JVM based web server (backend). The backend periodically fetches data from the *TorProject archive*[3] and stores processed and enriched data in a local database. Currently, the required archive part makes up 33 GB for the available years (2007 - 2021). Processing of a descriptor type only starts after all descriptors of the same type have been downloaded and saved to disk. Processed data can instantly be fetched by the frontend to be displayed on the world map. For further technical information, please also refer to the main **README.md**.

2.1 Frontend

The frontend is written in *TypeScript*[4] which is an open-source language that builds on top of JavaScript, and adds static type definitions. This helps catching errors and creating code that is easier to read and understand.

As base for data visualization, we used *Leaflet*[5] which is a popular JavaScript library for interactive maps. The user interface is designed with *Material UI*[6] which is a package for react that implements all the design principles of *Material Design*[7]. It allows designing a UI with minimal effort and great results.

The data for geo located relays is queried from the backend via a REST API. All further manipulations on this data set is done by the frontend before each rendering. Each time the selected date or settings get changed, will result in a new rendering. The pure time for rendering depends on the performance of the device but should be less than 60ms on an average computer. Therefore the limiting factor of the re-render time is the network bandwidth. The frontend is designed to work well on devices with a fast internet connection.

2.1.1 Packet structure

For dependency management and build generation *Yarn*[8] is used. All source code is located in *frontend/src/*.

Point of Entry: The root of our app is *App.tsx* where the general state necessary for the app is managed and all further components are being called.

components: All components that make up the app are located in this folder.

data: Data that is directly included in the app and does not get loaded separately. E.g. GeoJSON for countries

types: All TypeScript definitions used in the app should be located here. Definitions that are used in one component only can also be declared in the components file directly.

util: All helpers, as well as the config-file are located here. Helpers are additional React-Hooks, and functions that got outsourced from their component or don't belong to a specific component.

2.1.2 Config

The main frontend config is located at *frontend/srv/util/config.ts*. Further environment options like enable/disable Browser autostart and default port can be configured in *frontend/.env*. Dependencies are managed with *Yarn* and located in *frontend/package.json*. Compiler options for *TypeScript* are located at *frontend/tsconfig.json*.

[2] https://reactjs.org/
[3] https://metrics.torproject.org/collector.html
[4] https://www.typescriptlang.org/
[5] https://leafletjs.com/
[6] https://material-ui.com/
[7] https://material.io/design
[8] https://yarnpkg.com/

2.2 Backend

The backend is written in *Kotlin*[9] which is a modern approach to writing easy to maintain and save JVM based applications. Since 2019 Google also advises to use Kotlin as the main programming language for *Android apps*[10]. We use the *Spring Boot* Framework[11] as a standalone web server, which provides a REST API to be consumed by the frontend.

2.2.1 Packet structure

For dependency management and build generation *Gradle*[12] is used. All source code is located in *backend/src/main/kotlin/*. The *com.ip2location* package contains an *implementation by IP2Location*[13] to resolve IP addresses to a geographic location. The *org.tormap* package contains the main class *TorMapApplication.kt* which starts the backend server. The package also contains the rest of our backend application:

adapter: This package contains controllers which define the REST API endpoints.

config: This package contains config classes which model and document the user defined config in *src/main/resources/application.properties* The classes attributes are mapped to the user config at runtime.

database: This package contains JPA database entities in *database/entity* which model the DB tables. In *database/repository* repositories can be used to execute abstract Hibernate queries on the JPA entities.

service: This package contains services like the *SchedulerService* or *IpLookupService*. They interact with other services and the DB repositories to process and persist data.

2.2.2 API specification

TorMap uses the *OpenAPI*[14] standard for describing it's API and an interactive view of it called *Swagger*[15]. To use the interactive version start the backend and go to `http://localhost:8080/documentation` or for raw JSON output `http://localhost:8080/documentation/json`.

Figure 2: Backend API endpoints

[9]`https://kotlinlang.org/`
[10]`https://developer.android.com/kotlin/first`
[11]`https://spring.io/projects/spring-boot/`
[12]`https://gradle.org/`
[13]`https://github.com/ip2location/ip2location-kotlin`
[14]`https://www.openapis.org/`
[15]`https://swagger.io/`

2.2.3 Database and model

TorMap uses an embedded H2 database, which is saved locally in a single file. This enables us to deploy the backend with an preprocessed DB file, since the process of downloading and enriching all necessary data from the TorProject archive takes a lot of time. The main point of discussion was at what precision processed descriptors should be saved (hour / day / month / year). Since raw descriptor data used for TorMap is quite large (currently aprox. 33GB) and will become ever growing in the future, we decided to keep day precision for geo relays and month precision for details about nodes.

To manage future DB model migrations new SQL changelogs can be added under *src/main/resources/db/migration*. Our DB migration tool *Flyway*[16] then checks if new changelogs need to be executed on backend startup. The DB model is represented in Spring with *JPA entities*[17], which will be validated against the DB on backend startup. We use *Hibernate*[18] to construct simple queries against the DB.

Figure 3: **NODE_DETAILS** - stores a processed version of relay *server descriptors* with monthly precision.
GEO_RELAY - stores geographic locations of relays contained in *consensus descriptors* with day precision.
DESCRIPTORS_FILE - stores which descriptor files have been processed already.
AUTONOMOUS_SYSTEM - stores data imported from an IP2Location CSV file.
flyway_schema_history - used by the Flyway DB migration tool, which checks if migrations need to be applied.

[16] https://flywaydb.org/
[17] https://spring.io/projects/spring-data-jpa
[18] https://hibernate.org/

2.2.4 IP to autonomous systems

If you are not starting with a preprocessed TorMap DB you will need to import a CSV file containing autonomous systems into the local H2 database. It is advised to re import a new CSV file every few months, to keep the IP ranges up to date.

- Create a free account at `https://lite.ip2location.com/sign-up`
- Download latest IPv4 CSV file from `https://lite.ip2location.com/database-asn` or use the CSV file located at *backend/database/ip2location/IP2LOCATION-LITE-ASN.CSV*
- Run following commands on the TorMap DB:
  - **TRUNCATE TABLE AUTONOMOUS_SYSTEM;**
  - **INSERT INTO AUTONOMOUS_SYSTEM SELECT * FROM CSVREAD('<absolute_path_to_csv_file>');**

2.2.5 IP to geo location

TorMap uses a binary DB file from *IP2Location* to map IPv4 addresses of Tor nodes to geo locations. It is advised to replace the binary file every few months, to keep the IP ranges up to date.

- Create a free account at `https://lite.ip2location.com/sign-up`
- Download latest IPv4 BIN file from `https://lite.ip2location.com/database/db5-ip-country-region-city-latitude-longitude`
- Replace old BIN file with new one in *backend/database/ip2location/IP2LOCATION-LITE-DB5.BIN*

2.2.6 Config

The main backend config is located at *backend/srv/main/resources/application.properties*. Logging options can be configured with *backend/srv/main/resources/logback-spring.xml*. Dependencies are managed with Gradle and located at *backend/build.gradle.kts*.

3 Development setup

3.1 Requirements

Make sure you have at least 100 GB of free disk space, since the downloaded archive and local DB will take up a lot of space.

On most Unix systems you can use the install script ./install. It will try to use your package manager to install missing requirements. Depending on your shell you run the script with ./install or bash ./install.

If you use Windows or the ./install script failed, please install these manually:

- *Oracle JDK*[19] / *OpenJDK*[20] >= Java version 11
- *NodeJS*[21]
- *yarn*[22]
- *serve*[23]

Troubleshooting: Make sure **JAVA_HOME** points to a Java JDK version >= 11.

3.2 Run development servers

Make sure you have installed all requirements. The ./run script will start the backend and frontend servers in separate terminals/tabs. If you have *tmux* installed, a new tmux session named *tormap* will be started. Once you kill the terminal, the server is shutdown. The backend console will stay at an executing percentage below 100%. This is normal behaviour with Gradle and Spring Boot.

- on Linux: Type **./run** or **bash ./run**
- on Windows: Type **run.bat**

If the script fails or you prefer to run the servers manually:

1. Go into **backend** directory and run commands
   - **./gradlew** or on Windows **gradlew.bat**
   - **./gradlew bootRun** or on Windows **gradlew.bat bootRun**

   If you encounter problems with **gradlew** command it might help give it a try using **gradle** instead, without the wrapper.

2. Go into **frontend** directory and run commands
   - **yarn**
   - **yarn start**

In a fresh project without any preprocessed DB or pre-downloaded archive the backend will start to download an archive > 33 GB in size. Once the first 3 GB of consensus descriptors have been downloaded, they will start processing. Once 30 GB of server descriptors have been downloaded, it will take 1-2 days to complete processing. Any missing descriptors released by the TorProject will twice a day be automatically downloaded and processed.

Processing of descriptors does not necessarily happen in a chronological order, but one month of descriptors is always processed together. Different descriptor types are handled in parallel. While the backend is processing descriptors, the frontend will always be able to display finished data. Frontend features like family grouping or relay details will only be available, if the corresponding server descriptors have also been processed.

[19] https://www.oracle.com/java/technologies/javase-downloads.html
[20] https://openjdk.java.net/install/index.html
[21] https://nodejs.org/en/
[22] https://yarnpkg.com/en/docs/install
[23] https://www.npmjs.com/package/serve

4 Host project

First of all make sure you have installed all requirements for development. To be able to host, you should have a copy of a release or just successfully created your own project build.

Releases

- `http://timkilb.com/releases/tormap-version-1.1.0.zip`

Preprocessed databases

- `http://timkilb.com/databases/tormap-full-DB-2021-09-01-version-1.1.0.zip`
- `http://timkilb.com/databases/tormap-only-AS-DB-2021-09-01-version-1.1.0.zip`

Start backend

1. Go into the directory where the **.jar** file is located
2. Run command: **java -jar <backend jar file>**
3. Backend should be available at `http://localhost:8080`

Start frontend

1. Go into the directory where **index.html** is located
2. Run command: **serve -l 3000**
3. Frontend should be available at `http://localhost:3000`

5 Usage manual

TorMap is a web app that can be used in all modern web-browsers where JavaScript is enabled. The app is mainly optimized for usage on devices with a larger screen and good connectivity.

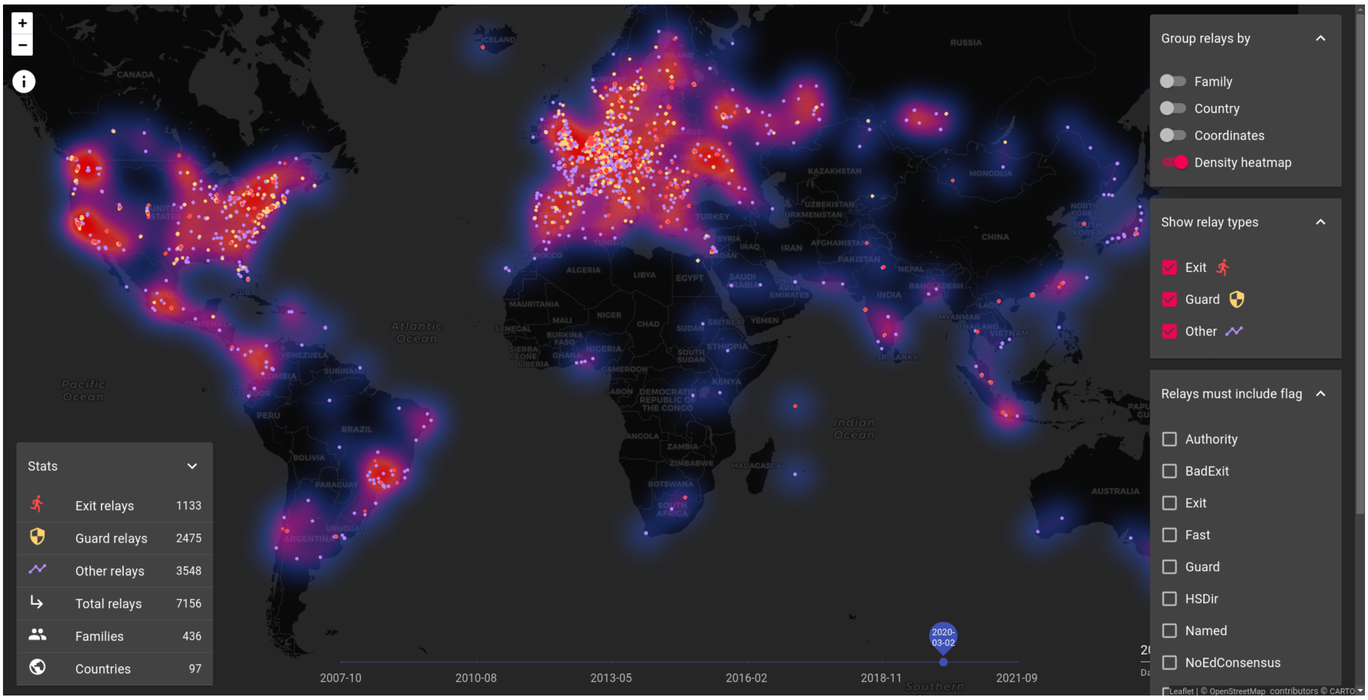

Figure 4: Screenshot of relay density heatmap

5.1 Date selection

The TorMap app visualizes historic data between October 2007 and today and let's the user select any day of interest. By default the latest available day is selected. To change the selected day, there are these options:

- Slider - you can select an date on the slider by moving it.
- Arrow-Keys - if the slider was just in use, you are able to move the slider with arrow-keys for small steps.
- Date picker - the date picker offers two options:
  - Text input - entering a date with number keys
  - Date picker - by clicking on the calendar icon

5.2 Relay selection

There is more information on each relay. By clicking on an relay marker more detailed information on that relay is shown. In case there are multiple relays on that marker position, the dialog contains a list with all those relays.

5.3 Change settings

The settings are located in the top right corner, in form of an accordion-menu. This menu has four parts.

5.3.1 Grouping relays by

- Group by family - will add coloured markers for each family of relays. The size of markers indicates the amount of family members at a specific coordinate. If there is only one family at a given coordinate it can be selected by mouse-click. Should there be multiple different families at a coordinate, a dialog opens where a family can be selected.

- Group by country - will add borders highlighting available countries which are selectable. Additionally all relays of the same country will be rendered in the same color.
- Group by coordinates - if there are more than 4 relays at the same point, a circle is added indicating the amount of relays at this location.
- Group by density heatmap - will visualize the geographic density of relays.

5.3.2 Relay types

The types option allows to exclude relays of a specific type from rendering. The type of a relay is:

- Exit relay - if the relay has the 'exit'-flag
- Guard relay - if the relay has the 'guard'-flag and not the 'exit'-flag
- Default relay - if the relay has neither the 'exit'- or 'guard'-flag

5.3.3 Relay flags

The flags filter allows the user to only view relays that have a specific set of flags.

5.4 Statistics

The statistics are located in the lower left corner, in form of an accordion-menu. For the calculation of the statistics only those relays get counted which match the current selection. Therefore if Germany is selected, only relays who's IP address belongs to Germany (according to the used IP to geolocation service) will be counted. If a family (relays that belong to the same organisation/owner) is selected, only the family members are counted. As families can span across multiple countries, if a family and a country are selected only the family members in the selected country are counted. If no family or country is selected, the amount of different selectable families and countries is displayed.

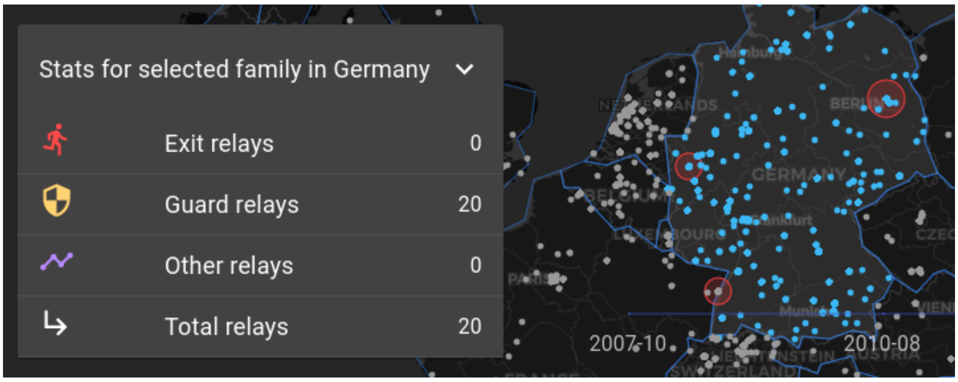

Figure 5: Screenshot of statistics

6 Feature list

This section lists all available features for the end user:

- Date selection
  - via slider
  - via text input
  - via date picker
- Filter relays
  - by type
  - by flags
- Group Relays
  - by family
  - by country
  - by coordinates
  - by density heatmap
- Show statistics for selected day
  - Amount of relays for each relay type
  - Total amount of relays
  - Total amount of different families
  - Total amount of different countries where at least one relay is hosted
- Show a map with markers for selected day and selected settings
  - Markers for relays
  - Markers for families (if selected in settings)
  - Country borders (if grouping by countries is selected in settings)
  - Markers for multiple relays on the same coordinate (if selected in settings)
- Show details for any relay on the map

7 Conclusion and outlook

The TorMap app visualizes the ever growing Tor network in form of an interactive world map with a bunch of different visualizations. We see a bunch of potential for further implementations in form of new endpoints for the REST API of the backend as well as more visualizations and filters in the frontend that would illustrate the fast transformations happening in the structure of the Tor network.

7.1 Range slider

At the moment it is only possible to query the data for an specific day in the life of the Tor network. The option to compare two different days would give a lot of further insights in to the changes happening in the network.

7.2 Bandwidth

Tor relays keep track of how much bandwidth they have spend on incoming and outgoing traffic. Visualization on the bandwidth of nodes could give insight to the flow of the network traffic of the Tor network.

7.3 Major events

We could put additional marks on the slider, marking major international and political events that had a 'higher' impact on the Tor network. For example 2009-09-24 to 2009-09-25, when over night 100+ relays disappeared in China.

7.4 Comparison to country metadata

For now the frontend includes an *GeoJSON*[24] file which contains data about countries. This is used to draw the country boarders on the map. The file also includes meta data about country's, e.g. their population. If this data can be queried for selected days, it's possible to create views that compare the Tor network activity to properties of a country.

7.5 Auto moving slider

We could add a function to let the slider move automatically trough history to visualize changes of size and distribution of the Tor network.

7.6 Detailed search

We could add a search dialog where relays can be searched by date, nickname, fingerprint, IP address, Autonomous System, etc. Matching relays would be displayed in a list and could be selected for detailed information.

[24] `https://geojson.org/`