IN-2194 Project: Jugendschutz

1 General Information

Preventing kids from accessing inappropriate materials online is a difficult issue given the increasingly dynamic nature of the web. Current centralized solutions involving blacklists often lag significantly behind the development of the websites and can amount to censorship for the general population. Relying on individual operators to rank their content (USK, FSK) fails to take into consideration that most sites include user-generated content and that cultural standards differ widely, making it virtually impossible for most operators to provide culturally appropriate rankings for all potential users of the site.

The goal of this project is to implement a Peer-to-Peer ranking system tha allows “parents” to submit content rankings based on their own cultural beliefs and to configure the browsers of their “children” to either suppress content ranked as inappropriate by (culturally close) parents, to allow content ranked as age-appropriate by (culturally close) parents or to demand the local “parent” to rank the site “now” using a password prompt. Additional features might include allowing site-operators to give a minimum/maximum rating for their own site, in case they are willing and able to undergo the necessary procedure to rate their content.

This is a P2P project because of course your system should have no central server or services. A centralized design would likely neither scale to billions of websites nor be able to integrate privacy protections for those accessing the system. The suggested high-level approach is that your system should store site ratings in a DHT.

For full points, you should find good, creative solutions to one or more of the following challenges:

- Ranking of entire domains is clearly too coarse-grained; for many URLs, rankings might even be different for various sub-areas of the site (i.e. embedded news videos on zeit.de might be less appropriate than the text). Also, the contents offered at a particular URL may change. How should this affect the way rankings are associated with content on sites? How should the user-interface (browser plugin) visualize rankings for such “embedded” content?

- The P2P system should make it difficult for an adversary to determine the surfing behavior of kids using the system. How can you protect their privacy? At the most basic level, hashing the URLs (or whatever identifier you choose) can make it less obvious, but strong anonymity might be even better...

- Parents might similarly want privacy for their rankings. On the other hand, some strong identification (ePA?) might be needed to avoid Sybil-parents from trolling by ranking sites inappropriately. Furthermore, parents are supposed to be possibly from different cultural backgrounds, and the system is supposed to take this into consideration. How can the system help find other parents that rank “similarly” without leaking personally-identifiable, private information on parents?

Note that the resulting system would not be a censorship system, as its use is voluntary (at least by the parents) and its actions are also socially transparent.

You must submit your solutions using Subversion by **August 15th 2012**. Late submissions will receive a score of zero unless you notify the instructor about a valid reason (such as the entire group having been abducted by pirates) **prior** to the deadline.

1.1 Subversion Access

In order to receive Subversion access, you must submit a request at

`https://projects.net.in.tum.de/projects/tum2194/register`.

In this request you must include the login name and desired password. Your login name should be your lastname followed by the last two digits of your student identification number.

Once your account(s) have been created and teams have been assigned, you can access the repository at

`https://projects.net.in.tum.de/svn-ext/tum2194/TEAMNAME/`.

You must submit a request for account and team creation by **Mai 15th 2012**. We want you to use Subversion for development, not just submission! Note that only the version of your code that is HEAD in the repository on August 15th 2012 at 23:59 will be graded.

1.2 Coding Style

You should read the GNU coding standards for a reasonable guideline for writing good C code. Additional coding style guidelines are available in the GNUnet developer handbook. In general, a shorter and simpler implementation will receive higher marks than a complex and long implementation. Naturally, comments and testcases will not be counted against you when we evalute the size of the code (in fact, good testcases and comments may earn you style points).

Your implementation must not perform busy-waiting and must not use pthreads (!). You should also try to avoid (unnecessary) copying of data. You should stick to the GNUnet coding style as much as possible (if your implementation would be good enough to be merged without significant changes into the GNUnet code base itself, you will get full points).

2 The GNUnet Framework

Your implementation should work in the context of the GNUnet P2P framework. Specifically, you will be using the 0.9.x development branch which can be found at `https://gnunet.org/svn/gnunet/`. Some documentation is available online at `https://gnunet.org/`. The framework already contains a DHT, including a C API `src/include/gnunet_dht_service.h` that is used by other components to access the DHT and command line tools (`gnunet-dht-put` and `gnunet-dht-get`).

In GNUnet, each component is executed as a separate process. Your implementation will likely use the *dht* service and include a *block* plugin.

Each of the services listed above provides a service access library (`libgnunetSERVICE.so`) to communicate with the service process that actually performs the operations. For this, the service process listens on a socket for requests from the service access library. Service processes are managed by the *arm* service.

For your own services, you need to write a service access library. For your implementation, you can choose to write the service process in C/C++ or Java. However, the Java API for some of the access libraries (to *core*, *dht*, *statistics* and others) is still under heavy development; thus the recommended development language is C/C++. If you choose to write your code in Java, you may have to deal with (more) bugs or other limitations in the existing access libraries.

Finally, you can ask questions not answered in the FAQ on the webpage in the `#gnunet` IRC channel on `irc.freenode.net`. We will answer non-FAQ questions in the public channel quickly. Developers in the channel speak English and German (and other languages); however, English is the preferred language.

3 Design

The design of the P2P protocol and your overall architecture is up to your team. However, you should include:

- A browser plugin for either Firefox or Chrome which can be switched between "kid at age X" mode and "parent" mode using a password
- A local, persistent (!) storage mechanism for ratings done by the local user (the DHT does not provide persistent storage!)
- Command-line tools to add/query rankings in the P2P network
- Documentation on your design and, if applicable, P2P protocol (and why your design is efficient and how good it is at achieving its objective, how you addressed some of the challenges given above)
- Installation instructions and screenshots of your tool in action.

4 Grading

You are expected to form teams of two to three students, but you can also work alone. Teams of four and more students are not permitted. Within each team, you are free to determine which team member is best for which task. Your overall grade will be determined primarily by the overall quality of the deliverable, but we will also discuss your contributions with you during the final demonstration and make sure that individual students are not penalized if some team members failed to contribute.

Grading will be based on the following main points:

12 Basic browser plugin and integration with DHT/persistent storage

4 Correct implementation of block validation plugin for the DHT backend

2 Command-line tools for manual ranking/querying

4 Persistent storage for rankings

12 Interesting solutions to the listed challenges

4 Documentation of the design

4 Technical soundness of the design

6 Overall correctness of the implementation of the design

2 Automated integration test

The project is thus worth a total 50 points.