uProxy Coding Guide

Introduction to technologies for uProxy development

uProxy is an open source application which runs in a variety of JavaScript environments, chiefly as a Chrome or Firefox browser-extension. We make use of several open source tools that contributors will need to be familiar with:

- We use git for source control, and the uProxy repositories are hosted on Github:
    - To learn about git on github, see Github's git school.
    - We use Travis, linked to our github account, for continuous testing (e.g. see: https://travis-ci.org/uProxy/uproxy-lib)
    - We also have our own continuous integration testing server(s).
- NPM for managing dependencies and publishing our repositories in a reusable way.
    - introduction to npm
- We write code in TypeScript (a kind of superset of JavaScript that is typed):
    - Two good places to start are the TypeScript Tutorial and TypeScript Playground.
    - The most important reference document is the TypeScript Handbook.
- Jasmine provides us with a unit test framework
    - See: introduction to testing with Jasmine.
- Grunt is used for building uProxy.
    - The steps are detailed in the uProxy README.
- Bower to install libraries used by the frontend.
- Polymer is the user interface framework we use.

Development processes

uProxy is an open-source project. We organise ourselves as set out in uProxy Milestone Management document. See the overall uProxy design document for a top-level outline of how uProxy works.

uProxy Coding style

See: uProxy coding style.

Build process and code Layout

See the uProxy Build Setup Document for the details of the design and how the build process works, as well as the principles behind it, and an FAQ on common errors you might see.

uProxy code is broken up into 3 repositories and each repository has its source under the `src` directory. Each directory in `src` contains a conceptually meaningful chunk of functionality.

- How npm dependencies work for us.
- uProxy source code is also divided into several Github repositories

Tests should be placed alongside the file they test. For JavaScript/TypeScript tests, they should be named `*.spec.js`/`*.spec.ts`.

Sample applications (which have ended up mostly being a kind of manually-driven end-to-end tests) are placed in `src/samples/` directory.

Unit Testing

Each repo can be unit tested by running `grunt test`. This will build the necessary modules so that `grunt build` does not need to be run first. Since Travis (which cannot set up our integration framework) runs `grunt test` to determine build status, integration tests are not run by `grunt test`, but by `grunt integration`.

Test files should be placed alongside the file they test to increase visibility of which files are tested, and because this is where the test rules expect the spec files to be.

In uproxy-networking and uProxy, code coverage is calculated using a code coverage template mix-in. See to our testing Q/A plan for more details.

TSD (TypeScript Definitions)

TSD is an effort to provide TypeScript definition files (.d.ts) for popular JavaScript libraries, e.g. jQuery.

uproxy-lib has a TSD "tree" under `third_party/` (as do uproxy and uproxy-networking).

TSD's syntax is...interesting. Read the full guide on their Github page. Here's two common commands:

- To install a new package:
  `tsd query node --action install`
- To search the online directory for angular typings:
  `tsd query angular*`

Once installed, the build process will create a symlink from build/third_party/typings to third_party/typings, allowing modules import TSD definitions from `../third_party/typings/xxx/`. Here's an example in uproxy-lib.

Note: no changes should be made to files in `../third_party/typings/xxx/` (except temporarily). All changes should be pushed upstream to https://github.com/borisyankov/DefinitelyTyped, and then removed from the local tree. The local tree should be populated during the install process command.

Gruntfiles

Our repositories mostly use Coffeescript to write our Gruntfiles.

Some tips:

- `require.resolve()` is quite nice. It allows you to remove the assumption that a dependency is directly within your `node_modules` folder:
  http://nodejs.org/api/globals.html#globals_require_resolve

Github repositories

Here are the most important repositories, along with what they do:

- https://github.com/uProxy/ - the github uProxy project group
  - https://github.com/uProxy/uProxy - frontends (for all platforms)
  - https://github.com/uProxy/uproxy-lib - libraries, used by other repositories
  - https://github.com/uProxy/uproxy-networking - SOCKS server
  - https://github.com/uProxy/sas-rtc - Video/Audio ZRTP-style authentication.

    - Obfuscation:
        - https://github.com/uProxy/libfte
          An encryption library that allows input plaintext and output ciphertext formats to be defined by regular expressions. (paper)
        - https://github.com/kpdyer/regex2dfa
          A command-line utility which converts a regular expression to a DFA. These DFAs are used as input to libFTE.
        - https://github.com/uProxy/uTransformers
          Packages libFTE for use by uProxy. Emscripten is used to convert libFTE's C code into JavaScript.
        - https://github.com/uProxy/turn-relay
          An obfuscated peerconnection provider for Freedom; this repo transports the obfuscated bytes across the untrusted network. Right now, this contains a mostly complete TURN server along with some sample apps. Uses libFTE via uTransformers. (design)
- https://github.com/freedomjs (docs at http://www.freedomos.org) Generic framework and modularization for building web-apps and isolating certain code into a web-worker, being developed primarily by Will and Ray at UoW. Provides a vision for browser-based distributed applications.
    - https://github.com/freedomjs/freedom - main freedom codebase including platform independent providers.
    - https://github.com/freedomjs/freedom-for-chrome - Freedom library with Chrome implementations of providers (e.g. TCP and UDP sockets and storage).
    - https://github.com/freedomjs/freedom-for-firefox - Freedom library with firefox implementations of providers (e.g. TCP and UDP sockets and storage).
    - https://github.com/freedomjs/freedom-social-xmpp - Freedom social provider based on chat networks that support xmpp.

Writing and Committing code

Our Github Branches

We use git branches in the following way:

- The **`master`** branch is only for releases; this should contain only a merge from the alpha branch (and then, possibly P0 bug fixes).
- The **`alpha`** branch is used for staging releases - no features should be landed on this branch. This branch should be a single merge from dev, followed by only high priority bug fixes.
- The **`dev`** branch is where development happens, all commits to this branch should be merged from pull requests. Features and bug fixes should targetted here (unless they are specific to a release or they are an important fix for a forthcoming alpha version).
    - Note: dev may contain merges from alpha or master to pull in bug-fixes targeted at the releases. Alternatively, a second pull request from the bug-fix branch to dev may be created.
- **user-id prefixed branches or github forks** should be used for new features or bug-fixes being developed.

In the future, we may create a second stage branch between alpha and master called `beta`.

Debugging

To turn on debugging in uProxy, from the uproxy main module webworker, you can type this (magic) incantation to set logging levels to debug (which shows all log messages) in the Chrome JavaScript console:

```
var logging_types = browserified_exports.logging_types;
var loggingController = browserified_exports.loggingController;
loggingController.setDefaultFilter(
  logging_types.Destination.console, logging_types.Level.debug);
```

Unless you are debugging something that is Firefox or node specific, it is recommended you debug in Chrome: it has a fantastic debugger.

Firefox is hard to debug. The best we've found so far is to run with their SDK tool, and having logging statements printed to the shell. To set different log levels, you'll need to find the `loggingController` statements in the main freedom module and change those.

Pushing changes to the github repository

1. Make a branch named, we'll refer to it as `BRANCH_NAME`, but you should name it in form of "`your_username-unique_feature_or_fix_name`". That way others know which branches are yours. Make your branch stem from origin/dev:
   ```
   git checkout -b BRANCH_NAME origin/dev
   ```
   Some good habits:
   - Do regular merging to the head of the dev branch branch to make sure you avoid nasty conflicts:
     ```
     git merge origin/dev
     ```
     And fix any conflicts (this is the equivalent of "g4 sync" if you use g4)
   - Push your branch to github regularly so others can see what you are working on:
     ```
     git push origin BRANCH_NAME
     ```
2. Edit what you need to edit, fix what you need to fix, and test it! For fixing bugs, make a test so we know we've fixed the bug, and we don't break it again.
   - Make sure every function has typescript specified input types and output type: callbacks should be fully typed as well.
   - Make sure you have merged in the latest changes from the `dev` branch when you test.
3. Don't break the build!
   - Write unit tests for all new code that you've introduced
   - Run `grunt test` and make sure all tests pass
4. Create a pull-request on the github page for your branch (the green icon by the repository/branch name that says "Compare & review" when you mouse-over; then you'll see the big green pull-request button), and ask the appropriate person to review it. You can always chat/ask any of the github repository contributors or owners. You can generally lookup the last few people to edit a file/directory and ask them to get the best person for review.
   - In the pull-request github description please include:
     - A little info on what the pull request does at a high level.

            - Make sure this has enough detail that anyone getting a notification can decide if they should look at it more closely.
        - A line or two on how it was tested. (e.g. "Tested: add new unit test, ran XXX test command and ran in on the chrome-browser manually".)
5. A reviewer should now review the code, and the author should discuss and make changes. Once reviewer and editor are happy, the editor can merge the pull request into dev, and then they should delete their branch using the github UI so that we don't end up with a million confusing branches lying around.

Code reviews

To keep code quality high (and avoid scary random things appearing in the code) all code must be peer-reviewed and only 'committers' have direct access to commit to the repository.

Assignment of reviewer

You can find a reviewer for your code by looking at the area of code you are working on and seeing who else has worked on that code in the recent past. You can also just send submit pull request to a uproxy-repository and people will volunteer to review it.

- If you want someone particular to review your code you can either:
    - Assign the pull request to them in github, or
    - Add their name in a comment, something like so: "@theirgithubid : requesting review, thanks!"
- If you want to review something (you want to review the code, and give an LGTM before the author of the pull request submits it), then add your name something like so: "Thanks, adding myself to review this lovely code: @yourgithubid".

Writing code reviews

As a code reviewer, you are responsible for making sure uProxy code is high quality, has appropriate tests, is of a reasonable size (e.g. does just what is needed for the feature/fix), and that we like the underlying design.

If you don't feel confident in having done a thorough review, make sure you are not the assigned person for the pull request: un-assign yourself/assign someone more appropriate.

Completing code review and merging your pull request into dev

Make sure to address all comments, and respond to them so that the reviewer knows; e.g. with "done" or "acknowledged" or "I don't think so because ...". Strongly recommended to use https://reviewable.io/

Once all comments are addressed, wait for all reviewers to write 'LGTM' or a give a thumbs-up to the pull request in the comment stream.

If someone just adds a comment, respond to the comment, but unless they are a reviewer (as described above), then you don't need to wait for them to give a LGTM.

It is the responsibility of the assigned reviewer and the author to make sure that all comments are addressed before code is submitted. At that point, you can submit you code. Bear in mind a reviewer will sometimes give you an LTM and trust you to to a few small final things before submission.

External contributors

External contributors should clone/fork and submit pull-requests for review. But before that, discuss the idea informally with the uProxy team by filing an issue for what you'd like to add. If your work is quite involved, you should write a design doc to give a high level explanation of your design and choices.

Quality Assurance: Build-cop & Release Process & CI testing

Our release schedule includes:

- We use Shippable and Travis for our CI testing infrastructure. More details in the uProxy CI testing and QA plan. Every pull request is tested.
- Once a week, a build cop tests dev and makes sure the latest build is healthy; and runs manual tests.
- Once every three weeks[1] the uProxy release process is carried out, and webstore versions are updated.

Build cop

Every week someone should be build-cop - this is someone who, on Monday/Tuesday, builds and tests uProxy. Build-cop rotation is decided on the monday weekly meeting. If something is broken, they figure out what it is and try to fix it. If things go well, building and testing should not take any more than 30 minutes.

Build-cop involves:

- Updating npm version numbers for dependent repositories.
- Testing (maybe fixing) and tagging the dev branch.

Updating dependent repositories

Every 3 weeks, we update the various repository versions to get them in sync and we include these changes in the new version of uProxy merged to the master branch on github.

To update the github version:

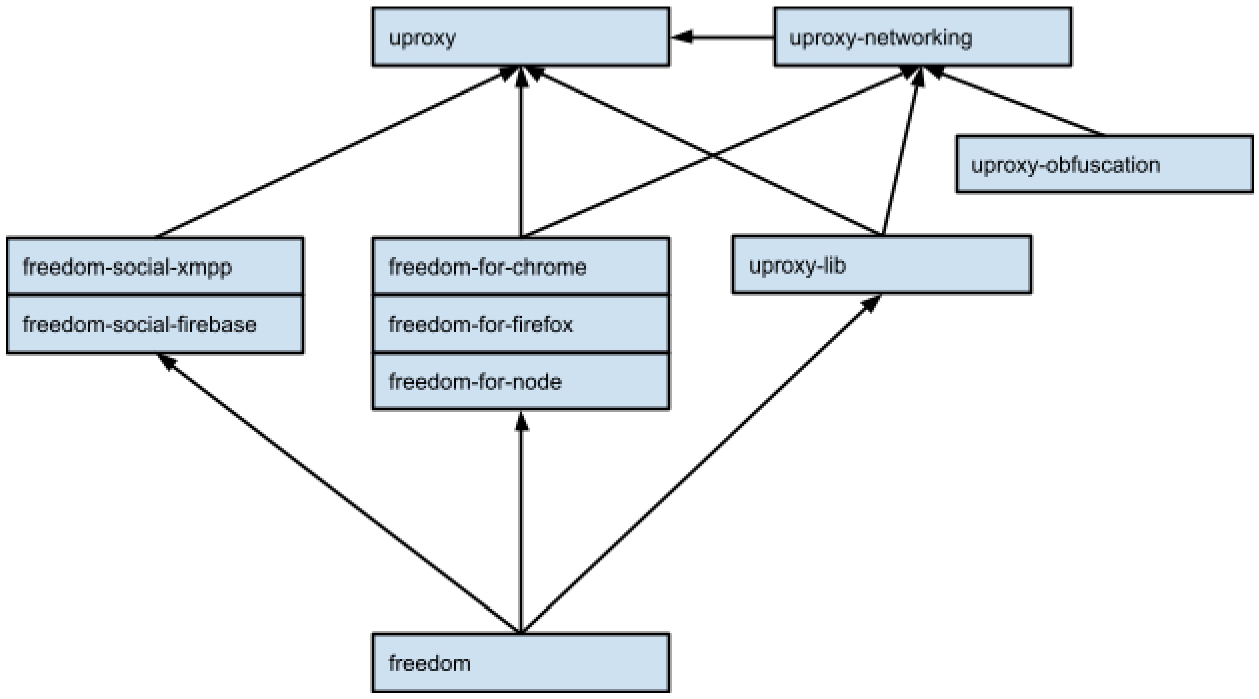
- You need to have committer access in the github uProxy group (or Freedomjs if you are releasing a freedom update) and appropriate repositories (You can ask Lucas or Ray to add you).

To update the NPM version:

- You'll need to have an NPM account: https://www.npmjs.org/signup
- You can find the latest release version on www.npmjs.com. For example here, the latest version of uproxy-networking is listed on the right.
- You need to have npm owner access to the npm package you are updating.
    a. Ask someone else on the team and NPM module owners, (e.g. Lucas or Ray), to add you as a maintainer of the package in question, e.g. see uproxy-lib.
- Look at the `package.json` file to see the various npm versions of things.

[1] (or sooner if we decide to do a quicker release cycle for some reason)

The repositories and their dependencies looks like this:



See the npm outdated command to get a list of packages to update using the command line npm command. Make sure that all dependencies are up to date (unless for some reason we have to be out of date for one of them).

See the uProxy semantic versioning guide for more details on NPM and versioning.

For external repositories, make sure to quickly scan the edits/commit history to avoid malicious code getting into uProxy from a dependent repository.

Testing & Tagging the dev branch

1. Download a uProxy fresh clone, e.g.
   `git clone https://github.com/uProxy/uProxy.git`
   This will get you the latest version on the dev branch.
2. Run the setup/build scripts (follow the instructions from the uProxy README) & *Fix anything that is broken with the instructions & email a short summary to the uproxy-eng list.*
3. *Test it.*
   - Run the unit tests; make sure it passes! `grunt test`
   - Do the end-to-end test for both Chrome and Firefox.
   - File bugs as uProxy github issues. If there's a bug, figure out who to talk to (if unsure, email *uproxy-eng list*), and help get it fixed, and add a unit test.
4. *Update these instructions if they are broken, and email it the uproxy-eng list.*
5. If the manual end-to-end and automatic unit and integrations tests all pass, use git tags to tag the version that you have just tested. It should be tagged according the format:

"build-cop-YYYY-MM-DD". You can do this with the following bash commands:

```
git tag "build-cop-$(date +%F)"
git push --tags
```

6. Email the uproxy-eng list and provide a link to the latest version you've just tagged. Something like this:
   Subject:
   Body: """
   ```
   I'm the buildcop, yeah. Latest build-cop approved version:
   https://github.com/uproxy/uproxy/tree/build-cop-$(date +%F)
   ```
   """

If you find something is broken, go back to the last build-cop tagged version and if that doesn't work, ask the build cop what's happening and to help save the day.

Accidentally added the wrong tag?

```
git tag -d THE_WRONG_TAG
```

Responsibility for breaking things

It happens to everyone, so we work together to help fix breakages.

If the dev-branch is broken (it doesn't build, tests fail, or some core functionality is not working), file an issue and label it `` `P0 - broken dev` ``. Try to figure out who the owner or last committer was and work with them to help get it fixed.

More generally, working together:

- Consider writing a test to check for the breakage in the future.
- Try to get the breakage fixed asap so as not to hold others up.

You should write tests when there is interesting or non-trivial behaviour or interactions. When other people edit code, and your code breaks, but there wasn't a test, that means you have a shared responsibility for making a working unit test and fixing the bug.

We all share responsibility for fixing things and making them beautiful!

Updating the website

Our website www.uproxy.org should be updated as needed.

The www.uproxy.org domain is a reflection of uproxysite.appspot.com, which is equivalent to our website's default version on App Engine. The `master` branch of the github repo should always match what the default version on App Engine is serving.

Prerequisites

- Access to the uproxy-website github repo (let Daniel know if you need access)
- Admin access to uproxysite appengine app (let Lucas know if you need access)
  - We're trying to figure out how to add non-google.com admins (issue #11)

Once you have the proper permissions, you can update the site as follows:

1. Install the AppEngine SDK for Python, available at https://cloud.google.com/appengine/downloads#Google_App_Engine_SDK_for_Python
2. Clone the github repo, e.g.
   `git clone https://github.com/uProxy/uproxy-website.git`
   If you want to make edits against a specific branch (e.g. the `beta` branch), instead run:
   `git clone -b <branch name> https://github.com/uProxy/uproxy-website.git`
3. Checkout your own branch of the branch just cloned, e.g.
   `git checkout -b <your branch name> master`
4. Run `bower install`
5. Open the app.yaml file and increase the version number. For example, bump `master-1-0-4` to `master-1-0-5`.
6. Make local changes and test by starting a local server with dev_appserver.py:
   `dev_appserver.py --port=9999 <path to directory containing app.yaml>`
   and then going to localhost:9999 in your browser.
7. Create a pull request to merge your changes into the corresponding branch on github.
8. After the pull request is reviewed and merged, publish using:
   `appcfg.py --oauth2 update <path to directory containing app.yaml>`
9. If you are updating `master`, change the default version at https://appengine.google.com/deployment?&app_id=s~uproxysite to your newest version to see your changes live on uproxy.org.

Major website changes

Future versions of the website that differ significantly from the current version should be created as entirely new branches instead of as pull requests against `master`. In this case, follow the same steps as above, but change the version name entirely to `<new version name>-1-0-0` instead of bumping the master version number. Also, for the first time you push this new branch, you can skip step 6, but make sure you have someone take a look at your code before you publish in step 7.

When the newer version is ready, to update uproxy.org:

1. Change the version in your app.yaml from `<new version name>-x-y-z` to the next major `master` version. Meaning if the current app.yaml at the head of `master` has the version `master-a-b-c` you want to set the version to `master-(a+1)-0-0` in the app.yaml you're about to push.
2. Make `<new version name>` the new `master` branch on github.
3. Follow steps 7 and 8 from above.

Currently, the `beta` branch of the github repo is under development and should always match what beta-dot-uproxysite.appspot.com is serving. This is done by publishing the most recent code on the beta branch twice: once to the appropriate `beta-x-y-z` version, and again to just the `beta` version, such that the `beta` version is *always* in sync with the highest `beta-x-y-z` version. When the beta site is ready, it will become the new master branch (per instructions above).

Bugs, Issues and workflow

All uProxy related issues should be filed in the main uProxy repository issue tracker:

https://www.github.com/uproxy/uproxy/issues
(sometimes you'll want to also file an issue in another bug tracker, e.g. freedom or chrome, if you do that include the link to the external tracked bug your uProxy issue)

Assignment: work that is actively happening should be assigned to a person who's taking responsibility for it. Every strand of work being carried out by someone on the team should have an issue.

Try to make issues into something you can do in less than two weeks. Update the github issue for it at least every week so that other's can see how progress is going and step in to help or give advice.

TODOs in code: TODO's in the code should have a link to the github issue.

Work ready to start being coded up is tagged `T:ReadyForCoding` ; You can see all work ready to happen listed with this link:
https://github.com/uProxy/uproxy/labels/T:ReadyForCoding

For filing bugs, see the template/guideline for bug reporting.

What do the issue labels/tags mean?

Priority tags: how urgent/important w.r.t. the milestone.

- `P0` - Urgent: stop whatever else is happening and make sure this is fixed.
- `P1` - Must be fixed for the milestone it is tied to; it is part of what that milestone means.
- `P2` - Intended and expected to work for the milestone, but can be moved back if it really has to.
- `P3` - Nice to have for the milestone.

T:* tags refer to the type of issue; this defines what kind of work is needed on the issue:

- T:ReadyForCoding - an improvement, bug-fix, or feature to be added that is well enough defined to start coding up.
- T:Needs-Discussion - more discussion is needed to figure out what to do.
- T:Needs-Tech-Research - needs some technical research is needed.
- T:Needs-UX - indicates that the issue is waiting on UI/UX design work.
- T:Organization - related to team organization.
- T:Wontfix - we are not planning to do anything; all these should be closed issues.
- T:Duplicate - the issue thread should say what it is a duplicate of. The issue that this issue is a duplicate of should not be marked as duplicate; all issues with this tag these should be closed issues.

C:* tags refer to the category of code/work the issue is relevant to. These are somewhat fluid. Add category tags as appropriate, but when you do send an email to uproxy-eng@googlegroups.com to let others know.

Updating tags, milestone, and priorities

Please add a comment when you change this meta-data with the reason: that way everyone subscribed gets a notification on the change.

When to close an issue

It's important to respect the person who filed the issue; you should work with them and let them be the one who closes the issue. It should be an exception when the developer closes the issue, e.g. because the person who filed it didn't reply for two weeks.

uProxy Developer Community

- uProxy github org
- Public uProxy G+ page
- https://www.uproxy.org
- (Informal, rarely used) IRC: #uproxy on Freenode (e.g. use: http://webchat.freenode.net/)
- Our shared calendar (which is public) -- click on the link to add the calendar your calendars list under "other calendars" at the bottom left. Add uProxy meetings you are ok listed publicly to it, e.g. standups (hopefully most meetings).

Reference Links and Documentation

Typescript tricks

The best documentation is probably the TypeScript Handbook.

Working with Sublime Text and TypeScript

- Do use: the best package to provide reasonable syntax highlighting/editor integration is.
  - Better Typescript: https://packagecontrol.io/packages/Better%20TypeScript
- Don't bother with:
  - typescript-tools: https://github.com/clausreinke/typescript-tools
  - T3S (as of Feb 2015) doesn't work well.

How to create a declaration file (*.d.ts) for a typescript file (*.ts)

```
# Move to your module's directory (MODULE_NAME)
cd build/typescript-src/MODULE_NAME
# Compile all the non-spec and non-declaration typescript with the -d flag.
tsc --mondule commonjs -d `ls | grep -e '\.ts$' | grep -v '\.d\.ts$' | grep -v
'\.spec\.ts$'`
```

Now you have a bunch of '`.d.ts`' files generated for you that you can copy back to where you want them in your `src/MODULE_NAME` directory. You probably want to remove the junky JS files that got produced too (that will later get compiled into the right place in `build/` - see above).

OAuth workflows

- FB

    - Chat: https://developers.facebook.com/docs/chat/
- Google:
    - G+ signin: https://developers.google.com/+/web/signin/
    - OAuth2: https://developers.google.com/accounts/docs/OAuth2
    - JS auth: https://developers.google.com/api-client-library/javascript/features/authentication

XMPP

- http://xmpp.org/

WebRTC

- Standard: http://dev.w3.org/2011/webrtc/editor/webrtc.html
- Basic demo: https://bemasc.makes.org/thimble/the-worlds-smallest-webrtc-demo
- WebRtc Test site: https://test.webrtc.org/
- Firefox data channel tester: http://mozilla.github.io/webrtc-landing/data_test.html

JSON/JS tools

- For testing validity of random JSON fragments, this is handy: http://jsonlint.com/

Network Address Translation (NAT) Firewalls

- Master's project that gives a nice outline of NAT types: http://www.confusticate.com/school/cs710/
- Network Address Translation (NAT) Behavioral Requirements for Unicast UDP: http://tools.ietf.org/html/rfc4787
- Flexible NAT Emulation Server: https://code.google.com/p/flexnes/
- A web-tool for telling you something about your NAT type: http://cc.rtmfp.net/
- The uproxy-probe project is a chrome-app that identifies the type of NAT you are using.