



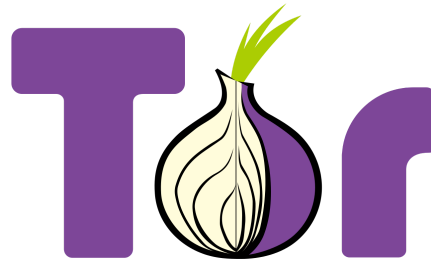
IMT Atlantique

Bretagne-Pays de la Loire
École Mines-Télécom



Technical report: 09 March 2018

09/03/2018



Project 102 - Contributing to TOR Project: improving the Ipv6 Support

Authors:

Meryem ZEROUAL

Caio VALENTE

Tutors:

Jean-Christophe BACH

Santiago RUANO RINCÓN

Summary

I - Introduction	2
II - Context: Tor & IPv6	3
2.1 Tor	3
2.1.1 The Onion Routing	3
2.1.2 The structure of Tor source code	4
2.2 IPv6	5
2.3 Tor and IPv6	5
III - Tools & working procedure	6
3.1 Tor Bug tracker & tickets	6
3.2 Contact with associated developers	6
3.3 Test environment and tools	7
3.3.1 Chutney tests	8
3.3.2 Stem tests	9
3.4 Working cycle	10
3.5 Patch submission process	11
IV - Contributions	12
4.1 Tor Patches	12
4.1.1 Ticket 25081: use get_uptime() consistently	12
4.1.2 Ticket 24714: rename conn->timestamp_lastwritten to conn->timestamp_lastwritable	14
4.1.3 Ticket 25432: remove router.c internal functions from router.h	15
4.1.4 Ticket 24732: Remove unused function fascist_firewall_choose_address_dir_server() and IPv6 DirPort code	18
4.1.5 Ticket 21003: extend_info_describe should list IPv6 address (if present)	19
4.1.6 Ticket 24735: Always check for the null address when calling address functions	20
4.2 Other contributions	22
4.2.1 shadow-plugin-tor, issue 44	22
V - Conclusion	24
VI - Bibliography & References	25

I - Introduction

The Tor Project is a non-profit organization that maintains Tor. Tor is a free software that allows a secure and anonymous connection to the internet. It is used by millions of people around the world, and some depend on this tool for their security.

Tor is open-source and developers who are most of the time not associated with this organization can participate in its development.

Up to now, the IPv6 support of Tor still has some issues, so this project aims to improve this support and to submit these improvements to the developers for revision.

The ongoing project opens the door to experience a new way of learning the basics of the C language, network and routing functions. As well as it highlights the challenges of understanding an existing code and its structure, knowing what every module is responsible for, and consuming the new algorithm structures that it came up with through the developing process.

Our work serves as well in the discovery of the system of contributing to an open-source project. And how this highly recommended experience for students of computer engineering, gave such positive results.

This type of projects is highly recommended for students of computer engineering, since it allows them to code and set their code alive by integrating it into a big project as well as it would allow them to make contact with developers from around the world with different backgrounds and knowledges but with the same aim which is to contribute to the development of a project.

II - Context: Tor & IPv6

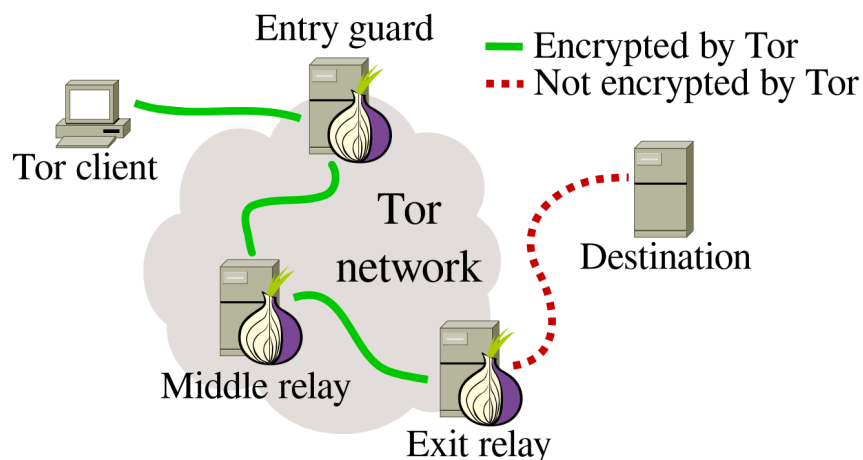
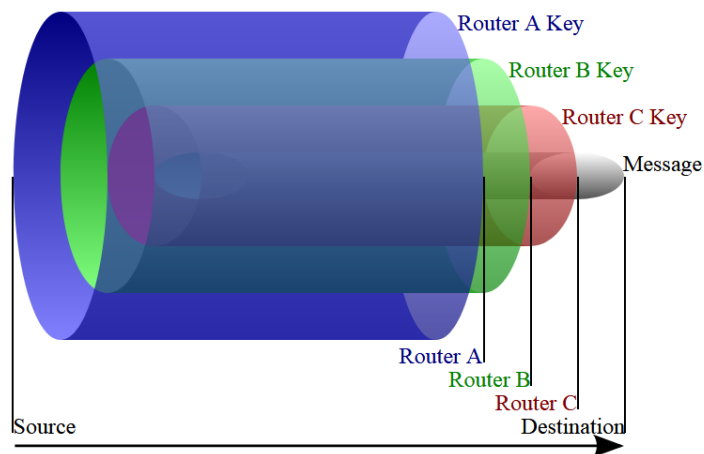
2.1 Tor

2.1.1 The Onion Routing

The Tor software passes the connection through three nodes in the network(entry node, intermediate node, exit relay), with an encrypted communication, so as to hide the actual location of the user. This kind of routing is called the Onion Routing, that is why it is called TOR.

As these nodes are placed in different places around the world, this type of routing gives the impression that the user is moving all the time.

The message is sent in an encrypted connection. Indeed, throughout the routing phase, each node of the circuit has a secret key encrypted with a public key dedicated to this node and in the end each node of the circuit will have a secret key of its own and knows only his predecessor and his successor within the circuit. In the end, only the output node knows the message that has been sent to it. In other words, each node has a unique way of encapsulating the connection, so that the message is protected by the three nodes used in its transmission.



2.1.2 The structure of Tor source code

Tor runs as an event-driven network daemon* that links clients to relays over an encrypted network and allows them to communicate by sending and receiving messages. This means it is designed to respond to network events, signals, and timers..

Clients, relays, and directory authorities all use the same codebase, and they can change the configuration in order to get different tor processes: a client, a relay or an authority.

The codebase is divided into these subdirectories:

- src/common -- utility functions.
- src/or -- implements the Tor protocols.
- src/test -- unit and regression tests.
- src/ext -- Code maintained elsewhere but it's kept in the Tor source distribution.
- src/trunnel -- automatically generated code (from the Trunnel).
- tool : used to parse and encode binary formats.

The most important abstractions and keywords of Tor are Connections, Channels, Circuits, and Nodes :

- A 'Connection' represents a stream-based information flow. Connections exist in different types :
"edge" (eg a connection from an exit relay to a destination),
"OR" (a TLS stream connecting to a relay),
"Directory" (an HTTP connection to learn about the network),
"Control" (a connection from a controller).
- A 'Circuit' is a persistent tunnel through the Tor network, it uses the public-key cryptography.
- A 'Channel' is an abstract view of sending cells to and from a Tor relay.
- A 'Node' is the current knowledge of a Tor instance about a Tor relay or bridge.

***Daemon** : is a computer program that runs as a background process, rather than being under the direct control of an interactive user.

2.2 IPv6

Internet Protocol version 6 is the most recent version of the Internet Protocol, it has formally become an Internet Standard on July the 14th 2017 in the **RFC* 8200**.

IPv6 was first mentioned in Internet standard document **RFC 1883**, published in December 1995. Later on that RFC was obsoleted and replaced by **RFC 2460**, which was published in December 1998.

This protocol serves as a link between end devices or machines/servers and allows them to communicate. IPv6 provides an identification and location system for computers on networks and routes traffic across the Internet.

The length of an IPv6 address is 128 bits, compared with 32 bits in IPv4. Which results in an address space of 2^{128} or approximately 3.4×10^{38} addresses.

IPv6 is an Internet Layer protocol for end-to-end datagram transmission across multiple IP networks. IPv6 also implements features which are not present in IPv4. It simplifies aspects of address assignment thanks to SLAAC (stateless address autoconfiguration), network renumbering, and router announcements(Router advertisement/Router discovery) when changing network connectivity providers. It simplifies processing of packets in routers by placing the responsibility for packet fragmentation into the end points.

The IPv6 subnet size is standardised by fixing the size of the host identifier to 64 bits in order to facilitate an automatic mechanism for forming the host identifier from link layer addresses/MAC addresses.

Inspired by Wikipedia

***RFC** : stands for Request for comments. It is a document that includes technical development and standards-setting bodies for the Internet, and it is published by the Internet Engineering Task Force (IETF) and the Internet Society (ISOC). These two organisations are responsible for internet governance.

2.3 Tor and IPv6

Tor works over IPv6, but it requires some manual configuration. Indeed Clients and relays could automatically detect IPv6 availability, and configure themselves appropriately to that.

Yet, the introduction of IPv6 can sometimes impose architectural problems to the Tor network, particularly in terms of network coherence, reachability of all nodes, and choosing the better port addresses..

Some of the open issues, also called tickets, include :

- A Tor relay is not able to determine if the system has a configured IPv6 address: [#5940](#)
- Some IPv6 features are not covered by Chutney automated tests
- There is no IPv6 ORPort reachability check: [#6939](#)
- Bridges binding only to an IPv6 address doesn't work: [#3786](#), [#22863](#)
- Relays publishing descriptors with incorrect IP addresses: [#17782](#)
- Some unused functions can cause code heaviness: [#24732](#), [#24734](#)

Source: <https://trac.torproject.org/projects/tor>

III - Tools & working procedure

3.1 Tor Bug tracker & tickets

Tor developers maintain the platform bug tracker where we can find "tickets", which are the tasks to perform in the project.

So, in order to find ideas about the tor bugs and fixings ideas, the best way is heading to the bug tracker at 'trac.torproject.org' and looking for tickets that have received the "easy" tag: these are the ones that developers think would be pretty simple for a new person to work on. For a bigger challenge, it is recommended to look for tickets with the "lorax" keyword: these are tickets that the developers think might be a good idea to build. Generally the lorax tickets include major changes that are spread across many parts of the codebase or major changes to programming practice or coding style and even sometimes introducing new features or protocol changes.

The platform comes with other features: a wiki subsystem enriched with information and roadmaps particularly : the ipv6 roadmap, and a useful guide: *"The TracGuide which is meant to serve as a starting point for all documentation regarding Trac usage and development"*.

3.2 Contact with associated developers

The tickets on the bug tracker contain a space for comments and discussions about tasks, where we can get in touch with people who are involved in tasks that interest us.

Generally, the response from the developers doesn't take long, and they are open to discussion and sharing ideas.

In addition to that, we can directly talk with the developers and ask our questions through the IRC #tor-dev channel, available on irc.oftc.net or send them direct emails on the tor-dev mailing list.

3.3 Test environment and tools

Since Tor is a software that runs on the network layer, sometimes it may be hard to test new codes or changes. Also, the level of reliability of the software may be extremely high, given the fact that the privacy of users rely on it.

Depending on the complexity level of a patch, it may require unitary tests, integration tests, or even simulation tests that may be done during the development phase of this patch and may be sent with it.

To assure that the modifications would not break any integration between modules, the many unitary and integration tests may be re-run every time before sending a patch, and the compiler must be configured to consider warnings as errors, to conform with the coding standards of Tor, that does not accept warnings on its compilation.

To run the tests related to **Tor Controller Interface**, we used the **Tor Stem library**. And to run connection tests, we used the **Tor Chutney tool**, that simulates a local Tor network with fake data, for tests purposes.

Another option would be using **Shadow Simulator** with the **shadow-plugin-tor** to simulate a Tor network based on real data. Even though this option provides many more test possibilities, it is also much more complex than the others and requires more time, so this simulator is only used for the patches with a high test complexity. However, we tested this simulator and even contributed to shadow-plugin-tor.

We hopefully had the help of our tutors, who provided a Debian remote machine with an IPv6 interface, so we could test IPv6-only features properly.

So, a normal test routine would be:

```
$ ./autogen.sh && ./configure --enable-fatal-warnings && make -j4
```

To compile the code considering warnings as errors, and then:

```
$ make test-full-online
```

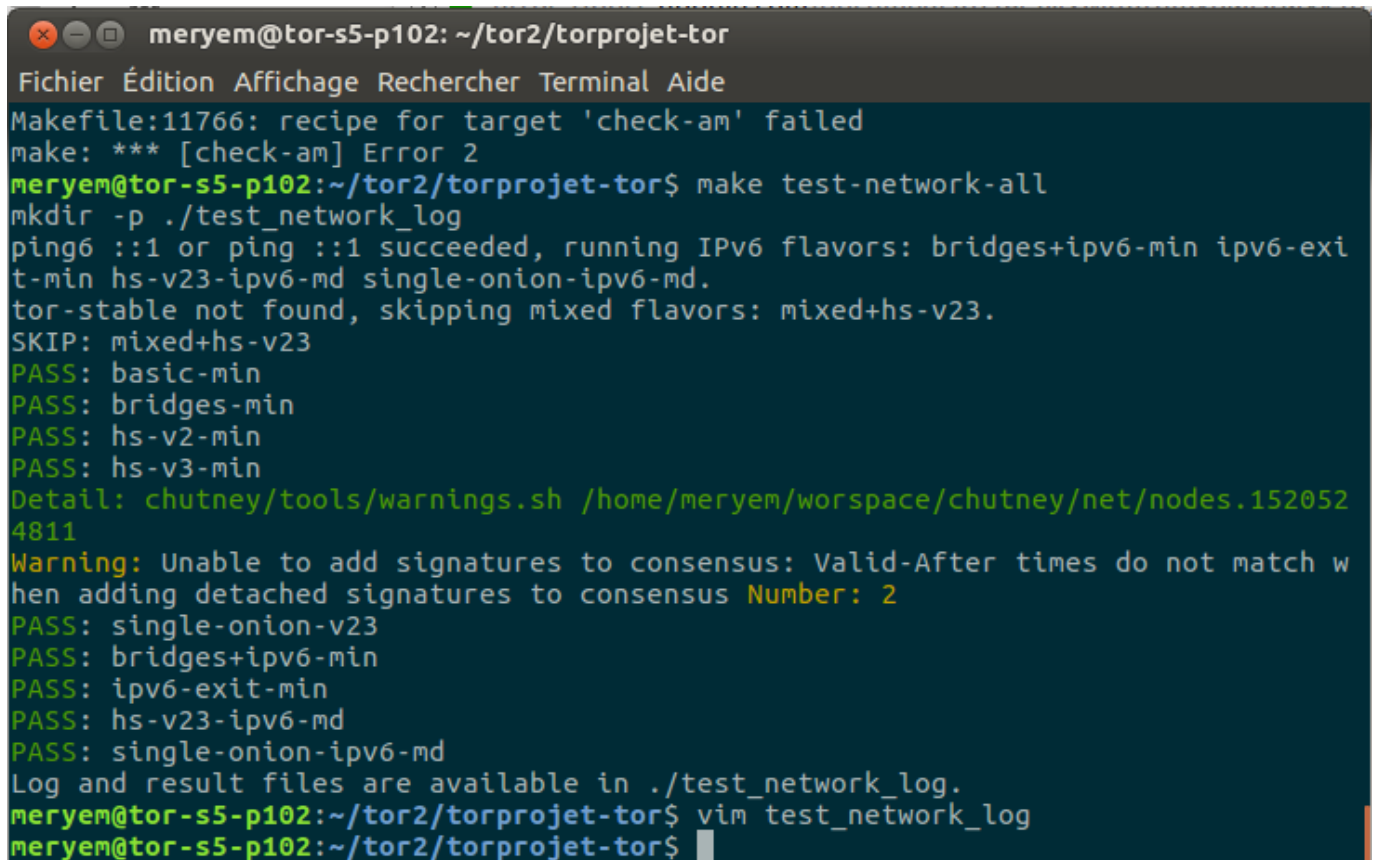
To all tests, including coding standard verifications, **chutney** tests and **stem** tests.

We could run separately the tests that check some coding standards, such as indentation and function naming, using:

```
$ make check
```


3.3.1 Chutney tests

The results of chutney tests are kept in the file `test_network_log`. As it is shown in the picture below:



```
meryem@tor-s5-p102: ~/tor2/torprojet-tor
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
Makefile:11766: recipe for target 'check-am' failed
make: *** [check-am] Error 2
meryem@tor-s5-p102:~/tor2/torprojet-tor$ make test-network-all
mkdir -p ./test_network_log
ping6 ::1 or ping ::1 succeeded, running IPv6 flavors: bridges+ipv6-min ipv6-exi
t-min hs-v23-ipv6-md single-onion-ipv6-md.
tor-stable not found, skipping mixed flavors: mixed+hs-v23.
SKIP: mixed+hs-v23
PASS: basic-min
PASS: bridges-min
PASS: hs-v2-min
PASS: hs-v3-min
Detail: chutney/tools/warnings.sh /home/meryem/worspace/chutney/net/nodes.152052
4811
Warning: Unable to add signatures to consensus: Valid-After times do not match w
hen adding detached signatures to consensus Number: 2
PASS: single-onion-v23
PASS: bridges+ipv6-min
PASS: ipv6-exit-min
PASS: hs-v23-ipv6-md
PASS: single-onion-ipv6-md
Log and result files are available in ./test_network_log.
meryem@tor-s5-p102:~/tor2/torprojet-tor$ vim test_network_log
meryem@tor-s5-p102:~/tor2/torprojet-tor$
```

Result of \$ make test-network-all

3.3.2 Stem tests

The results of stem tests are kept in the file `test_network_log`.

If a test fails the first thing is the console showing that there is an error. After that a file, called `test-suite.log`, is created where the results of the test are being kept so that you can find where exactly the test has failed.

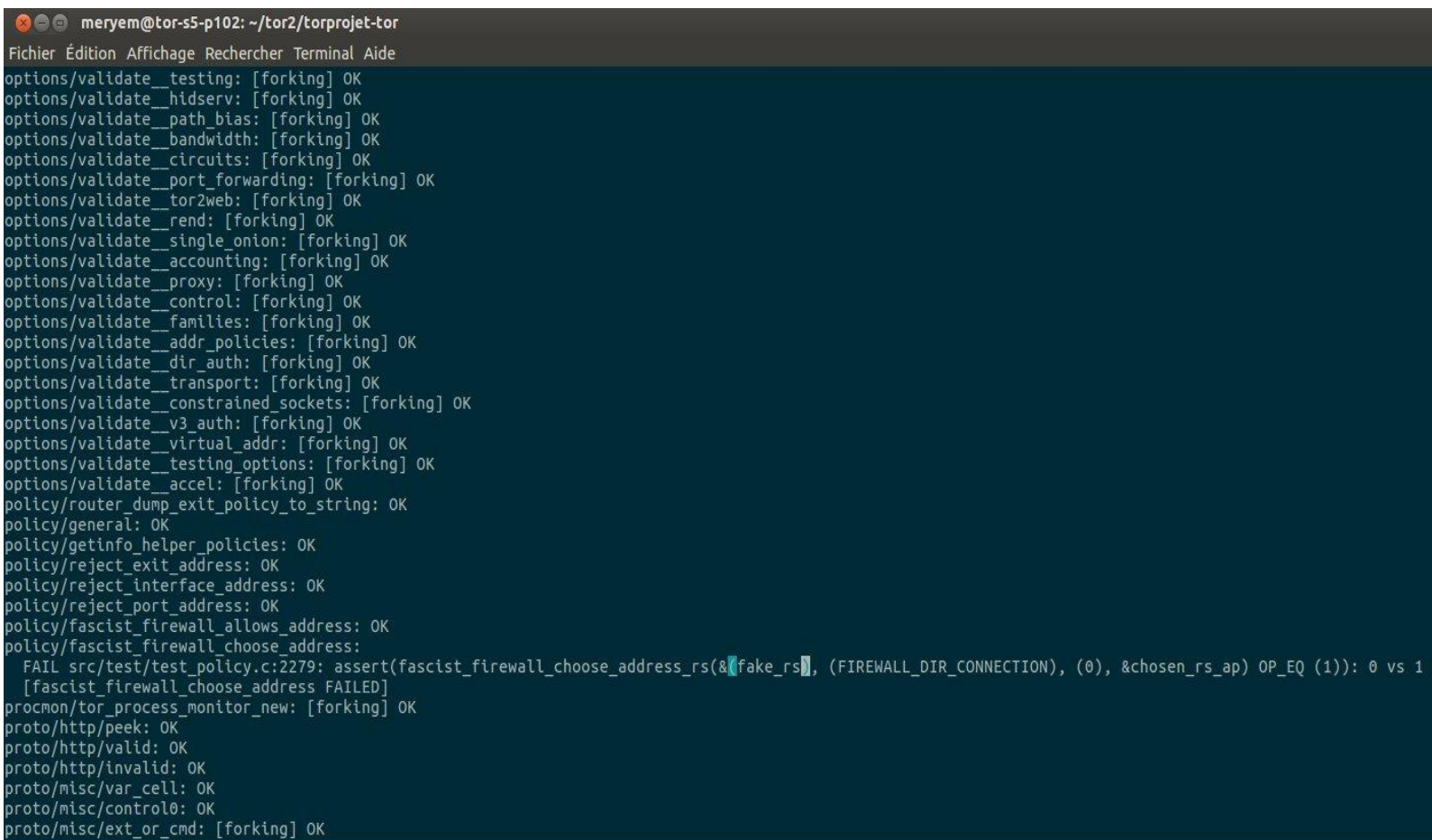
The structure of the created `test-suite.log` file is as following:

It is composed of lines where the name and location of the test is mentioned followed by “ok” for every successful test.

The line that describes the failure begins with FAIL and it shows the part of the test file that couldn't be successfully tested and it explains where exactly the conflict has been produced.

This screenshot shows an example of the `test-suite.log` file after a `$ make check` has failed.

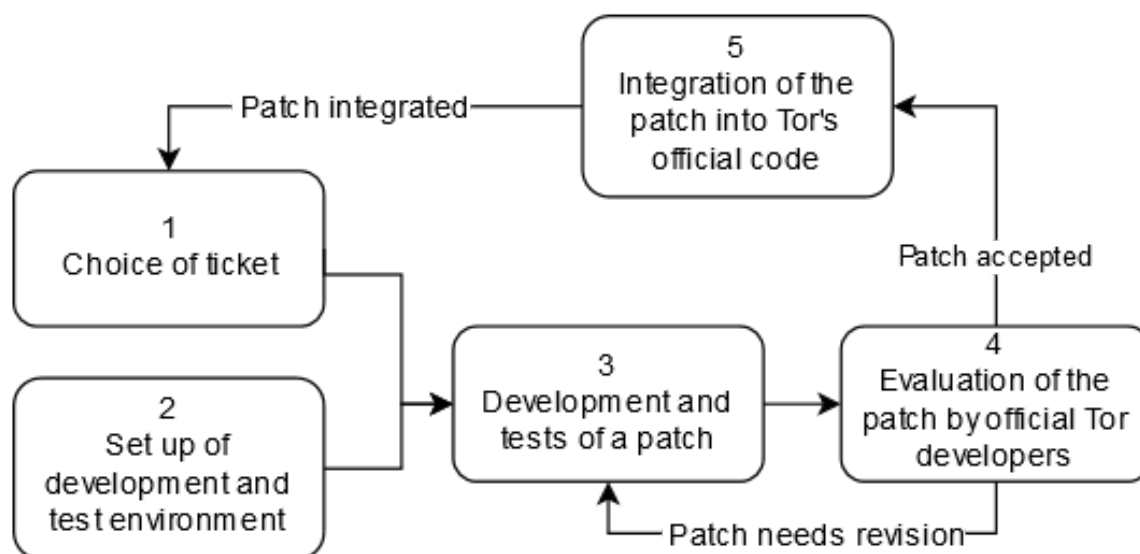
The line that holds the FAIL tag shows where and what part of the `test_policy` module is responsible for the problem.



```
meryem@tor-s5-p102: ~/tor2/torprojet-tor
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
options/validate__testing: [forking] OK
options/validate__hidserv: [forking] OK
options/validate__path_bias: [forking] OK
options/validate__bandwidth: [forking] OK
options/validate__circuits: [forking] OK
options/validate__port_forwarding: [forking] OK
options/validate__tor2web: [forking] OK
options/validate__rend: [forking] OK
options/validate__single_onion: [forking] OK
options/validate__accounting: [forking] OK
options/validate__proxy: [forking] OK
options/validate__control: [forking] OK
options/validate__families: [forking] OK
options/validate__addr_policies: [forking] OK
options/validate__dir_auth: [forking] OK
options/validate__transport: [forking] OK
options/validate__constrained_sockets: [forking] OK
options/validate__v3_auth: [forking] OK
options/validate__virtual_addr: [forking] OK
options/validate__testing_options: [forking] OK
options/validate__accel: [forking] OK
policy/router_dump_exit_policy_to_string: OK
policy/general: OK
policy/getinfo_helper_policies: OK
policy/reject_exit_address: OK
policy/reject_interface_address: OK
policy/reject_port_address: OK
policy/fascist_firewall_allows_address: OK
policy/fascist_firewall_choose_address: OK
  FAIL src/test/test_policy.c:2279: assert(fascist_firewall_choose_address_rs(&fake_rs, (FIREWALL_DIR_CONNECTION), (0), &chosen_rs_ap) OP_EQ (1)): 0 vs 1
  [fascist_firewall_choose_address FAILED]
procmon/tor_process_monitor_new: [forking] OK
proto/http/peek: OK
proto/http/valid: OK
proto/http/invalid: OK
proto/misc/var_cell: OK
proto/misc/control0: OK
proto/misc/ext_or_cmd: [forking] OK
```

File: `test-suite.log`

3.4 Working cycle



Since Tor runs on a network layer, sometimes it may be hard to test new codes or changes. At the same time, it is a big project, developed by many people, which means that some parts of its documentation may be outdated or the information we need can be hard to find. That's why the setup of the development and test environment required more time than we had planned in the beginning of the project.

After the setup (2) was done, we were able to start working on tickets.

We chose a ticket (1), worked on it (3) and, once we had a solution, we sent it for a review (4). Each patch must be reviewed by an associated developer of The Tor Project before being integrated (5) by another associated developer, and if the reviewer had some suggestion, we would revise the patch and resend it, until it had been accepted.

Given that the process of revision and integration can require some days, we avoided working on a single ticket at a time: while a ticket was waiting for review, we could start working on other tickets.

The total time of the setup (2) was around one month. We spent two weeks trying to build Tor on Windows, but then we decided to move to Linux, since the development of Tor is mainly done over this infrastructure.

After being able to build Tor from source, we dedicated two weeks to trying the test tools of Tor, such as Shadow, Stem and Chutney.

The development and the submission of the first two patches required more time than the others patches, because we had to adapt ourselves to the standards of coding and versioning used by Tor developers, but in the end of the project we were able to quickly reproduce a bug and submit a patch for it, so this way we could dedicate almost our whole time to the development of the solution for the tickets.

3.5 Patch submission process

The versioning of Tor is made with the **Git tool**.

We kept forking of Tor on our own github accounts* and we used those repositories to share our patches. That's the fastest way of sending patches, because the associated developers of The Tor Project can simply fetch our commits into the official git repository, merging our patch with the official code.

For each ticket, we created a branch, using the naming convention "t-xxxxx" where xxxxx represents the identifier of the ticket on bug tracker.

We also add a changes file to /changes/ticket-xxxxx for each patch that may close or help closing a ticket, describing the patch from a user point of view. Those changes files are automatically merged into a single changeset file when a version of Tor is released, so they may respect some standards.

*Our forks of Tor, used for sending patches:

<https://github.com/valentecaio/torproject-tor>

<https://github.com/mary-em/torprojet-tor>

*The latest version of Tor source is kept in Git in order to maintain the version control:

<https://git.torproject.org/git/tor>

IV - Contributions

4.1 Tor Patches

4.1.1 Ticket 25081: use get_uptime() consistently

Ticket: <https://trac.torproject.org/projects/tor/ticket/25081>

Patch: <https://github.com/valentecaio/torproject-tor/tree/t-25081>

#25081 closed defect (fixed)

Opened 5 weeks ago
Closed 3 weeks ago

use get_uptime() consistently

Reported by:	arma	Owned by:	
Priority:	Medium	Milestone:	Tor: 0.3.4.x-final
Component:	Core Tor/Tor	Version:	
Severity:	Normal	Keywords:	easy
Cc:	aegis	Actual Points:	
Parent ID:		Points:	
Reviewer:		Sponsor:	

Description

We have this nice function `get_uptime()` that shields the global variable `stats_n_seconds_working` from the rest of the Tor files.

But then we undermine that by saying

```
extern long stats_n_seconds_working;
```

in `main.h` and we just start using that variable directly all over.

There are a few lonely users of `get_uptime()`.

We should move everybody over to using `get_uptime`, and get rid of the scary `extern`.

(Also check out how we're **writing** to the `extern` variable, in `hibernate.c`. There's no way that could confuse anybody down the road!)

The first ticket we did concerned an encapsulation refactoring.

The variable **stats_n_seconds_working** was declared as `extern` in the `or/main` module and it was being used and/or modified directly by the modules `or/hibernate`, `or/dirserv` and `or/router`.

Since this variable represents the uptime of a running client or node, other modules should not be able to modify it to values different from zero.

At same time, the variable already had a get function implemented on or/main module, called `get_uptime()`, but the mentioned modules were not using it anyway.

Our solution was the encapsulation of the variable in `get_uptime()` and `reset_uptime()` functions, where this last always sets the variable to zero.

We removed the extern declaration of the variable from `or/main.h` header, and replaced the accesses to it on other modules by calls to `get_uptime()` and `reset_uptime()` functions.

This patch has been reviewed and approved in two weeks and it has been merged with the official code in 16/02/2018, in the commit `3ca04aada2916ce6963358007476eccd851e67cc` .

4.1.2 Ticket 24714: rename conn->timestamp_lastwritten to conn->timestamp_lastwritable

Ticket: <https://trac.torproject.org/projects/tor/ticket/24714>

Patch: <https://github.com/valentecaio/torproject-tor/tree/t-24714>

#24714 closed defect (implemented)

Opened 2 months ago
Closed 3 weeks ago

rename conn->timestamp_lastwritten to conn->timestamp_lastwritable

Reported by:	arma	Owned by:	valentecaio
Priority:	Medium	Milestone:	Tor: 0.3.4.x-final
Component:	Core Tor/Tor	Version:	
Severity:	Normal	Keywords:	easy, refactor
Cc:		Actual Points:	
Parent ID:		Points:	
Reviewer:		Sponsor:	

Description

"lastwritten" sure sounds like we actually flushed something or we actually added a new thing to write.

But actually,

```
time_t timestamp_lastwritten; /**< When was the last time libevent said we
                             * could write? */
```

It seems that we changed our definition of this word somewhere along the line, but we left the old word in place. How confusing.

We might want to change lastread to lastreadable too.

Our second ticket concerned the renaming of two fields of a struct..

The fields **timestamp_lastwritten** and **timestamp_lastread** of the struct `connection_t` actually represent the “last time this connection had permission to write” and the “last time this connection had permission to read”, respectively.

After a discussion with teor and arma, associated developers of The Tor Project, we decided that the new names would be **timestamp_last_write_allowed** and **timestamp_last_read_allowed**.

We refactored the fields to the new names and wrote the changes file for the patch that closed the ticket.

This patch has been reviewed and approved in two weeks and it has been merged with the official code in 16/02/2018, in the commit 3d7bf98d13ffc090e5ba1b918f53668018690dce .

4.1.3 Ticket 25432: remove router.c internal functions from router.h

Ticket: <https://trac.torproject.org/projects/tor/ticket/25432>

Patch: <https://github.com/valentecaio/torproject-tor/tree/t-25432>

#25432 needs_review enhancement

Opened 33 hours ago

Last modified 15 hours ago

remove router.c internal functions from router.h

Reported by:	valentecaio	Owned by:	valentecaio
Priority:	Low	Milestone:	Tor: 0.3.4.x-final
Component:	Core Tor/Tor	Version:	
Severity:	Minor	Keywords:	easy refactor
Cc:	caio-valente@...	Actual Points:	
Parent ID:		Points:	
Reviewer:		Sponsor:	

Description

These functions (group A):

```
A
const char *router_get_description(char *buf, const routerinfo_t *ri);
const char *node_get_description(char *buf, const node_t *node);
const char *routerstatus_get_description(char *buf, const routerstatus_t *rs);
const char *extend_info_get_description(char *buf, const extend_info_t *ei);
```

Do the same as these (group B):

```
B
const char *router_describe(const routerinfo_t *ri);
const char *node_describe(const node_t *node);
const char *routerstatus_describe(const routerstatus_t *ri);
const char *extend_info_describe(const extend_info_t *ei);
```

With the difference that those last allocate a buffer, instead of forcing the caller to create and pass the buffer as a parameter.

The functions from group B are an abstraction to the ones from group A: they are better because they always generate buffers of enough size (the size is `NODE_DESC_BUF_LEN`). So, we should avoid using group A.

By now, both groups are declared in the header, and there is only one use of a function of group A (`router_get_description` is used on `dirserv.c`).

Also, all those functions are abstractions to `format_node_description`, that should also not be used externally, so we could also remove this one from the header.

The constant `NODE_DESC_BUF_LEN` is not necessary externally either.

Our third ticket was created by ourselves. The ticket was reported as an encapsulation of functions, but finally we decided to suppress those functions.

While working on the ticket 21003, we found some functions and a constant that could be internal to the module or/router but were declared in its header.

We created the ticket 25432 for this refactor, and it was approved by nickm, associate developer of The Tor Project.

A group of four functions was actually an abstraction for other four functions that were only called by those first four functions.

The function **router_get_description()**, by instance, would be called by **router_describe()**, that would simply create a buffer of size **NODE_DESC_BUF_LEN** and pass it as parameter to **router_get_description()**:

```
const char *routerstatus_get_description(char *buf, const routerstatus_t *rs)
{
    if (!rs)
        return "<null>";
    return format_node_description(buf,
                                   rs->identity_digest,
                                   rs->is_named,
                                   rs->nickname,
                                   NULL,
                                   rs->addr);
}

const char *routerstatus_describe(const routerstatus_t *rs)
{
    static char buf[NODE_DESC_BUF_LEN];
    return routerstatus_get_description(buf, rs);
}
```

Since both functions were declared on router.h header, a module that imported this header would be able to use either **router_describe()** and **router_get_description()** to do the same job, which means that it was useless to have both functions. And the same situation would happen between the other functions of these groups.

As the constant **NODE_DESC_BUF_LEN** is only used by those functions and most of the other modules were already calling the functions that don't take the buffer as parameter, we decided that it would be better to keep those and suppress the others. This way, we were able to encapsulate the constant, moving it from the header to the source file.

So, we merged the functions that took the buffer as parameters into the ones that created the buffer. An example of result would be:

```
const char *router_describe(const routerinfo_t *ri)
{
    static char buf[NODE_DESC_BUF_LEN];

    if (!ri)
        return "<null>";
    return format_node_description(buf,
                                   ri->cache_info.identity_digest,
                                   0,
                                   ri->nickname,
                                   NULL,
                                   ri->addr);
}
```

The function **format_node_description()** was also removed from the header, given that the functions mentioned above were an abstraction to this one.

This patch was sent to a branch on our fork of Tor and is currently waiting for a review.

4.1.4 Ticket 24732: Remove unused function `fascist_firewall_choose_address_dir_server()` and IPv6 DirPort code

Ticket: <https://trac.torproject.org/projects/tor/ticket/24732>

Patch: <https://github.com/mary-em/torprojet-tor/compare/b-t24732>

#24732 **needs_revision enhancement**

Ouvert il y a 2 mois
Dernière modification il y a 13 secondes

Remove unused IPv6 DirPort code

Rapporté par :	teor	Responsable :	
Priorité :	Medium	Jalon :	Tor: 0.3.4.x-final
Composant :	Core Tor/Tor	Version :	
Sévérité :	Normal	Mots-clés :	ipv6, tor-relay
Copie à :		Actual Points :	
Parent ID :	#24403	Points :	1
Reviewer :	teor	Sponsor :	SponsorV-can

Description

IPv6 DirPorts aren't used by Tor: clients use IPv4/IPv6 ORPorts, and relays use IPv4 DirPorts (and IPv4 ORPorts). [Répondre](#)

There is code that implements this algorithm in commit 36ba50c820 of my bug23975_tree branch.

The patch still needs to be reviewed by the developers.

This ticket concerns the removal of the code related to IPv6 DirPort since they are no longer used and completely deleting the function `fascist_firewall_choose_address_dir_server()`. The IPv6 DirPort are directory ports of relays and directory servers. Relays and directory servers are being contacted over their IPv4 Dirport and IPv4 ORPort. The IPv6 Dirport is being deprecated now. The contribution aims to obsolete `ClientPreferIPv6DirPort`, and remove all `ipv6DirPort` addresses related code. In fact, IPv6 DirPorts aren't used by Tor anymore; Clients are now using IPv4/IPv6 ORPorts, and relays use IPv4 DirPorts/ORPorts.

The `IPv6DirPort` code was present in the `src/or` folder and it was being used and/or modified directly by the modules `or/nodelist`, `or/policies`, `or/config` and `or/connection`. And it was used for tests in test files such as `test_policy` and `test_options` in the `src/test` folder.

An example of the work done in the patch is deleting the function `node_ipv6_dir_preferred()` that returns if the parameter node has a valid IPv6 address and checks cases where we should prefer `ipv6 OrPort` address using `node_has_ipv6_orport()` or `ipv6 DirPort` using `node_has_ipv6_dirport()`. This function is not useful since we tend to use `ipv6OrPort` with both clients and relays from now on. Another example is the `test-policy` file where the tests of the policies defined in the `policies.c` module can be done with only `OrPort` addresses when on `Ipv6` support.

4.1.5 Ticket 21003: extend_info_describe should list IPv6 address (if present)

Ticket: <https://trac.torproject.org/projects/tor/ticket/21003>

#21003 new defect

Opened 15 months ago
Last modified 2 hours ago

extend_info_describe should list IPv6 address (if present)

Reported by:	teor	Owned by:	
Priority:	Medium	Milestone:	Tor: unspecified
Component:	Core Tor/Tor	Version:	
Severity:	Normal	Keywords:	easy intro ipv6 logging

Description (last modified by teor) Δ

When running an IPv6-only client using:

```
src/or/tor ClientUseIPv4 0 DataDirectory `mktemp -d` UseMicrodescriptors 0
```

I get log messages like

```
[info] add_an_entry_guard(): Chose $906933A309F15D7CF379127078A6791DF9B0C763~Unnamed at 91.121.82.25 as new entry guard
```

But I'm really contacting them on their IPv6 address. So we should list it along with the IPv4 address.

This was our first patch related to IPv6, and we are still working on it.

There are some situations where the logs of Tor should show IPv6 but they show IPv4 instead. Some nodes can have both IPv4 and IPv6 addresses, and Tor should be able to define with one of them is being used in the connection.

Since the IPv4 is commonly the main one, the options would be showing only the IPv4 address or showing both addresses.

By now, nodes are being shown in the format:

```
"$FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF~xxxxxxxxxxxxxxxxxxxx at ffff:ffff:ffff:ffff:ffff:ffff"
```

when the node is being contacted by IPv6, or:

```
"$FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF~xxxxxxxxxxxxxxxxxxxx at 255.255.255.255"
```

when the node is being contacted by IPv4.

We decided, by talking with the associated developer Teor, that they should be shown as:

```
"$FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF~xxxxxxxxxxxxxxxxxxxx at 255.255.255.255 or [fff:ffff:ffff:ffff:ffff:ffff]" (IPv4 followed by " or [IPv6]").
```

4.1.6 Ticket 24735: Always check for the null address when calling address functions

Ticket: <https://trac.torproject.org/projects/tor/ticket/24735>

Patch: <https://github.com/mary-em/torprojet-tor/tree/t-24735>

#24735 new defect Ouvert il y a 3 mois
Dernière modification il y a 0 seconde

Always check for the null address when calling address functions

Rapporté par :	teor	Responsable :	
Priorité :	Medium	Jalon :	Tor: 0.3.4.x-final
Composant :	Core Tor/Tor	Version :	
Sévérité :	Normal	Mots-clés :	ipv6, tor-relay
Copie à :		Actual Points :	
Parent ID :	#24403	Points :	1
Reviewer :		Sponsor :	SponsorV-can

Description

These address functions have never had return values:

- `node_get_prim_dirport()`
- `node_get_pref_ipv6_orport()`

We should make sure we always check for the null address when we call them.

Répondre

This patch needs to be reviewed by the developers.

The ticket tackles an important point that is to check for the null address when we call a void address functions .

In the older versions of Tor, a wide range of address functions did have a return value. For example: `node_get_prim_dirport ()` which is responsible for copying an ipv4 directory port address in the pointer (address pointer) we set in the parameter did have a return value. So, when it was returning 1 or 0, we were able to know if the address result is valid or not.

The transforming of these functions to void in order to optimise the code and diminish the heaviness, created some bugs in terms of using these functions because we don't know that the address pointer result is null or not when we call one of these address functions.

One of the changes is in the function below `node_get_pref_orport()` that renders either ipv4 orport or ipv6 orport when assigning addresses to the nodes:

When the `node_ipv6_or_preferred(node)` is set to 1, we have a preference for ipv6 orport, so we must get a valid ipv6 orport address and port using the `node_get_pref_ipv6_orport()`. Yet, since it is a void function we can't use it this way: `return node_get_pref_ipv6_orport();` which would give 1 if a valid ipv6 orport address is assigned or 0 if not. So, in order to know that we must check for the null address after calling that function, as shown below:

```

/** Copy the preferred OR port (IP address and TCP port) for
 * <b>node</b> into *<b>ap_out</b>. */
void
node_get_pref_orport(const node_t *node, tor_addr_port_t *ap_out)
{
    tor_assert(ap_out);

    if (node_ipv6_or_preferred(node)) {
        node_get_pref_ipv6_orport(node, ap_out);
        if (!tor_addr_port_is_valid_ap(ap_out, 0)) {
            node_get_prim_orport(node, ap_out);
        }
    } else {
        /* the primary ORPort is always on IPv4 */
        node_get_prim_orport(node, ap_out);
    }
}

```

The same thing with the function node_get_pref_dirport() that chooses the dirport address and port (either ipv6 or ipv4) :

1599	1602	{
1600	1603	tor_assert(ap_out);
1601	1604	
1602		- if (node_ipv6_dir_preferred(node)) {
	1605	+ /* the primary Dirport is always on IPv4 */
	1606	+ node_get_prim_dirport(node, ap_out);
	1607	+ if (node_ipv6_dir_preferred(node) !tor_addr_port_is_valid_ap(ap_out, 0)) {
1603	1608	node_get_pref_ipv6_dirport(node, ap_out);
1604		- } else {
1605		- /* the primary DirPort is always on IPv4 */
1606		- node_get_prim_dirport(node, ap_out);
1607	1609	}
1608	1610	}
1609	1611	

First there was no checking for the null address that may result from node_get_prim_dirport(), this function that sets the ipv4 dirport as primary choice. So the idea was to call it and then check if there's i) a preference to ipv6 dirport

Or

ii) the result ipv4 dirport address pointer is null(!tor_addr_port_is_valid_ap()) in order to switch to ipv6 with node_get_pref_ipv6_dirport() .

4.2 Other contributions

4.2.1 shadow-plugin-tor, issue 44

Shadow-plugin-tor is a plugin that can be used to run private Tor networks of clients and relays on a single machine using the Shadow discrete-event network simulator.

When testing this tool, we found a bug that we reported on its github:

<https://github.com/shadow/shadow-plugin-tor/issues/44>

valentecaio commented on 21 Jan



I'm trying to build shadow-plugin-tor using the newest version of Tor (commit ID 48a51c5f8b80a359da31bc5aaac8ecd25890fe0d) but it seems that a Tor global variable called `tor_git_version` is being redeclared.

When trying make, I have the following output:

```
[ 98%] Linking C shared library libshadow-plugin-tor.so
CMakeFiles/shadow-plugin-tor.dir/___/___/tor/src/or/git_revision.c.o:(.rodata+0x0): multiple definition of `tor_git_version'
CMakeFiles/shadow-plugin-tor.dir/shadowtor-main.c.o:(.rodata+0x0): first defined here
collect2: error: ld returned 1 exit status
src/tor/CMakeFiles/shadow-plugin-tor.dir/build.make:4839: recipe for target 'src/tor/libshadow-plugin-tor.so' failed
make[2]: *** [src/tor/libshadow-plugin-tor.so] Error 1
CMakeFiles/Makefile2:147: recipe for target 'src/tor/CMakeFiles/shadow-plugin-tor.dir/all' failed
make[1]: *** [src/tor/CMakeFiles/shadow-plugin-tor.dir/all] Error 2
Makefile:127: recipe for target 'all' failed
make: *** [all] Error 2
2018-01-21 16:10:05,847 INFO make returned 2
[2018-01-21 16:10:05.847483] Shadow Setup: ERROR! Non-zero return code from make.
2018-01-21 16:10:05,847 INFO returning code '2'
```

valentecaio commented on 21 Jan • edited ▼



`tor_git_version` is already declared in Tor core (at `tor/src/or/git_revision.h`) and should not be redeclared (accordint to Tor [ticket23845](#)).

I removed its redeclaration from `src/tor/shadowtor-main.c` and the build works, but I don't know if this change could break older versions of Tor.

We also found the source of the bug, helping to solve it.

A recent change in Tor's code had added a global variable **tor_git_revision**, and the `src/tor/shadowtor-main` module of the plugin was redefining this variable, breaking its compilation.

We removed its redeclaration from `src/tor/shadowtor-main.c` and the build worked, but this would break the compatibility with previous versions of Tor, so the developers of shadow-plugin-tor proposed a different fix that kept the compatibility.

V - Conclusion

This project was a good introduction to the world of open source projects, allowing us to actively contribute to an existing big project that is still in progress.

It was an opportunity that helped us experience a new work process, different from the traditional way of implementing algorithms for local projects. As our code has been reviewed and evaluated by associated developers of The Tor Project, who are specialised in onion routing and experienced in C development, we did our best on each written patch.

We got to notice the importance of a good infrastructure. When dealing with a big project that involves the network layer, the proper setup of a development environment is crucial for a good experience, but this task sometimes can require a remarkable time that may be foreseen to avoid delays.

Tor is a big open source project, with many complex modules whose documentation can sometimes be imprecise, but we were able to contribute to it by understanding the bugs, reproducing them and understanding the modules where these bugs come from. This way we could help to solve smaller bugs while we learned bit by bit about the whole project, instead of trying to understand it completely before working.

After being able to build and test the application, there is a great variety of tickets to choose from, each one involves different areas, with a different level of complexity, and this project gave us a great freedom to choose the tickets that we felt more comfortable to work on. This way, we could start our contributions with simple tickets, that we could quickly find a solution, and then we could eventually evolve to more complex ones.

The project also allowed us to discover how the management and documentation of a big open source project are done, and how important good tests and good communication can be for a project. The contact with both associated developers and tutors was essential for the good development of this project: many of our issues were solved with the help of our tutors or in discussions with the developers of The Tor Project, who always answered our questions quickly and accurately.

We reached the end of the project having two patches already integrated into Tor's official code and some others waiting for approval, which means that we have been able to do more contributions than we had planned.

VI - Bibliography & References

Our forks of Tor, used for sending patches

<https://github.com/valentecaio/torproject-tor>

<https://github.com/mary-em/torprojet-tor>

Chutney on Tor gitweb

<https://gitweb.torproject.org/chutney.git>

Tor Stem documentation and how-to-use

<https://stem.torproject.org/index.html>

<https://www.antitree.com/2014/04/15/private-tor-network-chutney/>

Shadow-plugin-tor repository on Github

<https://github.com/shadow/shadow-plugin-tor>

Tor Bug Tracker

<https://trac.torproject.org/projects/tor>

IPv6 Roadmap on Bug Tracker

<https://trac.torproject.org/projects/tor/wiki/org/roadmaps/Tor/IPv6>

Starter guide for submission of patches

<https://gitweb.torproject.org/tor.git/tree/doc/HACKING/GettingStarted.md>

Coding and Git usage standarts adopted by Tor developers

<https://gitweb.torproject.org/tor.git/tree/doc/HACKING/CodingStandards.md>

List of useful tools for testing patches

<https://gitweb.torproject.org/tor.git/tree/doc/HACKING/HelpfulTools.md>