TOR PROJECT - GOOGLE SUMMER OF CODE

Arti API exploration to build example tools

Summary
self link: Tor GSoC Proposal

Create utilities using Arti to illustrate deficiencies in APIs and docs and to help instruct future developers on how to use the library in new and creative ways, as well as fulfill certain use cases

Owner: Saksham Mittal (gotlou)
Approver: Ian Jackson, Nick Mathewson
Status: **For Review**

Google
Summer of Code

Project Abstract

The main goal of this project is to help complete documentation and speed up development of Arti, the Rust rewrite of the Tor utility, by creating example utilities using the Arti-provided crates (which help abstract away how Tor connections work) which will help the Arti developers discover any gaps in documentation, code, undefined behavior, functionality etc.

Such utilities include, but are not limited to:

- A download manager in Rust which will get the Tor Browser Bundle through a bridge connection.

  It is not only a small project which will test the async capabilities of Arti, but also a useful tool in its own right; the utility could be built and used in situations where bootstrapping a Tor connection anonymously or securely is almost impossible (see countries where requests to torproject.org or associated mirrors could be monitored and flagged).

  While this will be the large project that will showcase (and also test) how well Arti can be used for real, tangible, use cases, some smaller projects may also be designed to easily explain to new users of Arti's APIs how they can be leveraged to build Tor-powered network applications.

- A non-HTTP related project would also be ideal since arti_client could be effectively utilized (ie, it makes sense to use arti_client).

For this purpose, a simple DNS lookup tool like dig could be built which uses Tor relay to resolve hostnames anonymously, reducing the need to trust a DNS server.

The DNS lookup tool could be built both as a small tutorial in docs to get developers acquainted with Arti and to showcase the experience of running different protocols over Tor.

- Another utility could just print out the current connection info on the circuit that had been chosen (like in Tor Browser) and to test relay and bridge connectivity at the user's end for diagnostic purposes. This will help illustrate how to connect to bridges and get connection info, which may be desirable for other projects.

- We should also have a small project that can also be completed if the need arises, which doesn't necessarily use Tor in its entirety, but rather some functionality that Arti has.

  For example, using pluggable transport protocols (obfs4, Snowflake, meek) to encapsulate network traffic of various sorts in order to make it harder for a passive network observer to find out not just the contents of the message, but also the message *type*, removing more metadata from an adversary's view (a simple client SOCKS5 proxy and a small SOCKS5 server would perhaps suffice)

- Finally, Roger Dingledine helped point me in the direction of a tool requested by Tor developers themselves to create a bridge reachability monitor using Arti. This tool would hold connections to many obfs4 bridges open and detect when one of them goes down. I believe such a tool could also be built during the work period, and that it will likely benefit Arti greatly by testing its concurrency capabilities at scale.

- Besides this, we can also focus on bolstering Arti's docs by providing examples for those functions which don't have any and may need some to better explain their utility for developers. The exact list to document will be partly worked out in this proposal, with the rest being open to suggestions from Arti developers based on the future of that method, current stability or scope of the project.

Background

1. Arti's APIs are experimental. While Arti has reached v1.0.0, the crates arti_client and arti_hyper still haven't. Arti itself still has yet to support onion services, for example, though this is being earnestly worked on

2. The docs (see arti-client and arti-hyper) are lacking a bit in terms of using the Arti bindings to build new projects, and there aren't many projects using Arti either (see reverse dependencies of arti-client and arti-hyper)

This presents us with a vicious cycle where if the APIs are not used in production workloads, we won't be able to find bugs and undefined behavior in the code. For this, we'd need to build projects/example code of our own to use the API in non-trivial ways to uncover issues.

Hence the importance of this project cannot be stressed enough to accelerate the adoption of Arti over C Tor in production, such as in the Tor Browser Bundle or in Tor relays.

3. There is still a huge amount of potential for building new tools/projects using Arti to take advantage of the safety and concurrency benefits that a Rust-based project inherently has, so we should endeavor to build using Arti today rather than write using C Tor and then port to a more stable Arti later down the line, even if Arti is not feature-compatible with C Tor yet.

4. While looking at Tor's distribution methods (directly via torproject.org, mirrors hosted by allied parties, a Telegram bot, emailing gettor@torproject.org) I found that there was a weak link for the solutions to get Tor in censored regions: they rely too much on a third party to directly distribute the Tor Browser Bundle.

   One way that more technically-minded users can use is to build the download manager from source from a code hosting website like Github (since they are typically less restricted due to their sheer usefulness) and use that to download Tor Browser as a last resort in case these services go away. (Telegram for example has a tendency to be vilified in certain regions of the world and may be blocked, emails can be accessed by governments, allied parties may also end up blocked or monitored).

   The Tor Browser so downloaded can be checked cryptographically for correctness and then be distributed using a sneakernet, as is often the case in many countries. This may help limit the amount of digital evidence left behind trying to get Tor safely and can improve the chances of spreading Tor further.

   While the outcome of this GSoC project won't really be anything more than a CLI tool, there's no reason why the user experience can't be improved in the future to allow the average person to be able to use this tool to get connected to the Tor network.

Design Ideas

The download manager proposed will do the following:

1. Connect to the Tor network via a Snowflake proxy (this should be configurable but default behavior should be via Snowflake since Snowflake is the least likely to have been blocked in a censored region due to its ephemeral nature and the legitimacy of WebRTC connections compared to the apparent gibberish that is obfs4)

2. Create X different Tor circuits (the value of X could be configurable, but by default may be assumed to be six)

3. Find the Tor Browser Bundle's latest version and download it via multiple interfaces to speed up the process (using HTTP Range headers)

4. Implement support to resume and stop downloading at any particular time (or to handle network changes easily). For further help, the download manager can take some inspiration from aria2.

For the DNS lookup tool, here are the proposed features:

1. Connect (optionally via Snowflake or other pluggable transports, but this is a nice-to-have) to the Tor network

2. Fire a DNS request and print the response, either through DNS over TLS (avoid using DNS over HTTPS to justify using arti_client) or by just attempting to create a TorAddr and printing the result.

   This will require learning how to craft a DNS request, parsing the response and handling it suitably. (We will do this the hard way)

   This is supposed to be a smaller project, so I don't expect to add too many features to this tool other than just being a way to get a DNS response manually using Tor. It will serve as a good example of a protocol that can be routed over Tor (DNS over TCP at least) and can help other developers do the same, illustrating the ease of use when Tor is built into their application instead of requiring a proxy being run all the time.

The status checker is also a smaller project that will require just the basic knowledge of how Arti stores connection info and a simple way to print that information to the user in an easy-to-understand manner.

The design of the status checker would be as follows:

1. Try to connect directly to the Tor Network and check if a certain set of Web resources are accessible

2. Report the circuit info to the user if this is successful, if not print error and proceed to step 3

3. Test connections via all pluggable transports and print their circuit info if successful, and an error if not.

The obfs4 connectivity checker is the other production level program that will be built during this period, and will be of great use to the Tor Project.

A design I thought of for such a tool would go like this:

1. Find the maximum number of simultaneous connections that Arti can sustain to bridges. (Let's say it is X)

2. Always maintain X connections, but in order to cover ~2k bridges, we randomly choose a fraction of connections to close and then open for other bridges. So, say the fraction is ½, so that will mean we close half of the open connections and pick X/2 other bridges to connect to.

3. Randomly picking which connections to close means that we can test long-lived connections to certain bridges and also whether a bridge is firewalled or not from making new connections.

4. If X is sufficiently large, we can balance testing new connections with the benefits of holding open connections to bridges and get near real-time updates on when a bridge goes down. (For actual real-time updates, X would likely have to equal the total number of bridges, and so many open connections could make the tool expensive to run)

5. Overall, we'd need to invest time experimenting with the values of X and the fraction of connections to close in each pass, but the tool itself can be built relatively easily thanks to async Rust.

For the pluggable transport SOCKS proxy system, the design could be as such:

1. Get more information about how pluggable transports are implemented in Arti. This may require pre-emptively improving documentation on pluggable transports

2. Use the pluggable transport APIs to create a SOCKS5 server and client, which can encapsulate TCP and UDP traffic. Some talk on the Matrix/IRC group has revealed that the PTs do indeed implement SOCKS forwarding, so this will be a trivial task.

As for the partial list of improvements to documentation in arti_client, here it is:

1. BridgesConfig could use more examples detailing how to use different pluggable transports automatically; this would be explored in the example projects but a dedicated, small example should be added to the docs for ease of use by programmers.

2. Error handling could be better explained in examples (for example, circuit failures); right now the examples just use ? to propagate the errors up the call stack, which, while understandable for examples, is not suitable for developers wishing to use Arti in production. Some examples/explanations on the error handling would be appreciated.

3. Various structs, such as config::pt::ManagedTransportConfig, config::dir::NetworkConfig, config::circ::PathConfig etc, are not very explicit in what they do or why they are important enough to be exported publicly. These can be prime candidates for writing example code..

During the development process, it is extremely likely that bugs will be uncovered in Arti. I would be working alongside the Arti developers (including Nick and Ian) to help debug and fix these issues for the betterment of the project.

Code Affected

arti_client and arti_hyper will be most affected, but it is possible that lower level crates will be altered or added to in the process of building example projects.

It will be difficult to ascertain as various parts of the code may need to be changed throughout development depending on the issues that pop up, as well as new design proposals that may crop up if, say, it is necessary to expose a certain functionality to external developers.

Documentation of Arti will obviously be bolstered through this project, especially arti_client.

Related Work

Right now, there is little documentation on how to use Arti. Arti_client is a very basic interface to the Tor network in Rust, and arti_hyper provides just the HTTP client interface to access web services using what is mostly just vanilla hyper, just with some Tor-isms added.

This is mostly, as stated before, due to Arti's experimental nature. This project will be the bootstrap project to get other developers using Tor in their applications in interesting and novel ways.

Pre-proposal Work

Here are my prior contributions to Arti/Tor:

- On my first day of building software using arti_client and arti_hyper, I found a bug related to how Arti works with SQLite

  It has been addressed by the Arti devs in a Merge Request

- I experimented with Arti in a repo to help me learn more about the APIs and to build a basic downloader to get Tor. This will serve as the basic building block for building example utilities.

- Arti!1042: Remove unnecessary warning from arti-hyper/README

- Arti!1037: Use humantime in tor-checkable and tor-guardmgr

- Arti!1048 (unmerged): Handle \r in tor-netdoc

- Tor!703 (unmerged): Add test for \r in directory parsing

Other Commitments

Before continuing, it is important that I highlight the other commitments that I have during the GSoC work period clearly in order to better explain the rationale behind the timeline that I have prepared.

1. I have end semester practicals from May onwards as well as end semester exams from mid-May to June 14. Till this time I will not be able to deliver many contributions, however since most of this time is spent in the Community Bonding Period and the beginning stages of the coding period, I feel like this will not affect me as much, since even the work selected for the first two weeks is relatively lightweight, and the Community Bonding Period's work has also been determined.

   Besides, I am already familiar to an extent with Arti thanks to my pre-proposal work, so this shouldn't pose a huge obstacle.

   I also think that I can manage the end semester exams with my GSoC work fairly well. My college's exams are somewhat easy (mostly similar to previous years' papers so we get a good idea of what's to come) and in the past I was able to do a variety of activities other than exam prep during the preparatory leaves between exams.

2. In my college, we have the 6th semester for internship, so from July onwards companies will start their internship selection processes, due to which I may have less time to devote to the GSoC projects.

   This eventuality has also been planned for, since the week-wise workloads are such that I do not have to work all seven days of the week on the GSoC project and can take the rest of the time to focus on such commitments as well.

   Note: this also helps with managing any unforeseen obstacles while building these projects like any program-breaking bugs that may be discovered or code refactorings.

3. My college will start from August onwards, so naturally I will not be as free to work on the project as in the summer, however by that time the majority of the work will be done and I will only have improvements to make. I will still have weekends free to work on the project as well. This has been reflected accordingly in my timeline.

Ultimately, it is my responsibility to manage my time and to effectively deliver the project on time as per the commitment that I am making to the Tor Project and to Google Summer of Code.

I feel like my timeline and my design ideas help illustrate my level of confidence and expertise in this project and that I am still a worthy candidate for this project idea, despite these commitments.

Additionally, if a week's goals are met (which I do expect to happen, especially with buffer weeks), there is nothing stopping me from getting a head start on next week's work either, or to concurrently work on two different things at once.

For example, if I am using a certain method in my project to accomplish a certain goal, I can always write a small bit of example code in the docs and try to get it merged in order to fulfill a goal of this project, which is to improve documentation for Arti's APIs. While this won't always be a feasible technique, it is just one optimization that I am willing to make in order to deliver this project in a timely, organized and efficient manner for all parties involved.

Schedule of Deliverables (timeline)

This is a **175 hour work period**, so here is the weekly timeline to achieve the goals of this proposal in an organized manner:

May 4 - 28 (Community Bonding Period)

- This period of time will be spent better getting to know mentors and delving into arti's crates. This will include trying to understand arti_client, which seems to be the less documented crate (at least arti_hyper can be used using hyper docs, but arti_client, if documented properly, could lead to building diverse types of applications on top of Tor safely)

- I'll also be researching how DNS over TLS works and how to talk correctly to a server for making a correct DNS request over it, as well as how to print the connection info for a given circuit.

- I will also be delving into the available Arti crates with the help of my mentors to get an idea of how I would actually implement certain features and to get a better idea of Arti's architecture as well (I've already gone through https://gitlab.torproject.org/tpo/core/arti/-/blob/main/doc/dev/Architecture.md)

- Some details regarding the obfs4 status checker would also be ironed out during this period thanks to a discussion with fellow Tor developers. This includes learning about the relevant APIs, getting some pseudocode together, and existing solutions that exist and how they work etc.

- Also, extending or curating the list of methods/structs etc to document/illuminate through examples will be done in a similar way, ie, discussion.

May 29- July 3 (Phase I)

During the exams, I expect I can only devote 1-2 hours per day to the work on average. After the papers, I will be able to work 3-6 hours per day and will also write blogs on my website regarding my progress.

Note: the agenda for each week will be the main task to be worked upon, if the agenda has been fulfilled the leftover time would be devoted to writing a blog post or to prepare for next week.

- **May 29:** Write the user-facing Tor connection checker utility (it is not a very big project so I expect a week should be enough thanks to the legwork done in the community bonding period) and improve the required docs, starting with pluggable transport examples which would be used in other projects.

- **June 5:** Start working on the bridge connector; first write basic code to connect to a particular bridge and use that to find the upper limit on simultaneous bridge connections

- **June 12:** Take existing download manager code and modify it to create a fixed number of connections and preserve them.

- **June 19:** Distribute portions of the Tor Browser Bundle to each connection made and download them concurrently (using HTTP range headers)

- **June 26:** Add resume downloads feature to handle any sudden interruptions, and support downloading TBB for multiple platforms.

- **July 3:** Buffer week in case any task(s) are left over. If buffer week is not needed, then proceed to one or more of the following tasks: writing docs, writing tests, general code cleanup

  Along with buffer tasks, also write pluggable transport SOCKS proxy if needed (depending on if this project is deemed to be relevant enough to also be completed during the GSoC work period)

July 10 - August 21 (Phase II)

I will be able to work for 3-6 hours per day till the end of July and then on average 2-3 hours per day on weekdays and 3-6 hours on the weekends from August onwards.

- **July 10:** Using the maximum simultaneous connection value, test closing a fraction of connections at a time and opening new connections to find an optimal value (can start from 1/10 and increment by 1/10 for each test round, for example, though this would require more discussion)

  Submit the work done for midterm evaluation by July 14 as highlighted in the timeline

- **July 17:** Write the manual DNS lookup tool. This shouldn't take more than a week thanks to the fact that the scope of this project is to simply illustrate better how non-HTTP protocols can be routed through the Tor network, and DNS is a fairly simple protocol.

- **July 24:** Route connections of the download manager through a pluggable transport (target Snowflake as a priority, but if possible allow obfs4 and meek-azure as well with a flag to use bridges or not or to use a specific type).

- **July 31:** Buffer week in case any task(s) are left over. If buffer week is not needed, then proceed to one or more of the following tasks: writing docs, writing tests, general code cleanup.

- **August 7:** Handle a situation where a certain circuit dies due to an unexpected network error and create a new connection to replace the same in the download manager, ensuring it doesn't crash the program with it.

- **August 14:** Get reviews from the Tor developers on any and all improvements that can be made to the projects and address their concerns as best as possible, while keeping in mind the looming final deadline and the scope of the project (so while not every improvement may be addressed, I'll do my very best to implement those changes that are able to be implemented).

  While I'll be getting feedback from my mentors throughout the work period, this week would specifically solicit advice from the general Tor dev community.

- **August 21:** Complete any documentation gaps in the projects, and suggest documentation improvements in Arti itself (eg. code examples for certain functions that may have been a bit hard to understand when using them in the projects).

  Submit the work done for the final evaluation

Post GSoC

- Help maintain the utilities that have been created from this work period
- Contribute to Arti by addressing TODOs in the code or helping to solve issues or feature requests
- Guide others to build tools with Arti in case the docs and examples don't sufficiently answer their questions
- Keep docs up-to-date with the API changes that may occur since we haven't reached v1 for arti_client crate

Communications

Here are my details on when and how to reach me:

- Timezone: Indian Standard Time (UTC+5:30)
- Phone: <redacted>
- Email: gotlouemail@gmail.com
- Matrix: @gotlou/matrix:matrix.org (I bridge to IRC using Matrix)

In order to complete this project, it is essential to work with my mentors. To this end, I have been active on the Tor IRC and ask my queries regarding a specific detail of how Arti works there, and I get good advice on what problem I'm facing.

While the plan outlined above is fairly realistic, there may be some delays arising due to unforeseen complications. In this case, I will communicate to my mentors about the same and adjust the work needed to be done in a suitable manner. Buffer time has also been allocated for this purpose.

About Me

I am a sophomore undergraduate student pursuing Information Technology from National Institute of Technology, Kurukshetra in Kurukshetra, Haryana, India. I have always loved to learn more about computers from an early age and have been doing so for many years.

One of my favorite areas to learn about is networking. I have a special passion to learn about how networked computer systems work, the Internet, and all the stakeholders and protocols that work hand-in-hand to provide such an amazing tool to billions of people on this planet.

I've also been writing code in Rust for added safety and ease-of-use and am fairly proficient in it. Thanks to this and the fact that I have written networking applications before would make me an ideal candidate for this project, and that I've had Tor around on my machine for years and sometimes use it to mask others' activities on Tor as well as my own.

I also have experience in C, C++, Python, JavaScript, Go, utilizing Linux effectively, version control and other small skills that would help me succeed in achieving milestones and creating deliverables.

Here are some links to learn more about me:

- Website: https://gotlou.srht.site
- Tor Project Gitlab: https://gitlab.torproject.org/gotlou
- Github: https://github.com/gotlougit
- SourceHut: https://sr.ht/~gotlou (Thanks Drew DeVault for helping with financial aid for my SouceHut account!)
- LinkedIn: https://linkedin.com/in/saksham–mittal

Prior Experience with open source

Most of my experience has been with my own personal projects, although I have made contributions to other projects as well:

- Personal project: https://github.com/gotlougit/p2p-file-transfer: this is a Rust tool to transfer a file from one peer to another, even if both are behind stateful firewalls that prevent incoming connections without corresponding outgoing ones.

- Personal project: https://gitlab.torproject.org/gotlou/arti-sample-project: a Rust project using Arti to build a small download manager to get the latest copy of Tor Browser Bundle in a secure fashion; this may serve as a basic skeleton of the download manager proposed in this document.

- Contribution to https://github.com/intel/cve-bin-tool: Added support to check for vulnerabilities in Apache Web Server, one of the most widely deployed pieces of software on Earth

- Contributions to https://github.com/nit-kkr-student-support-system/breadboard: This is a web service intended for use by NIT Kurukshetra students. I helped implement JWT support in the backend and also to cache college announcements in database to provide redundancy against the relatively unreliable official website

- Contribution to https://github.com/tldr-pages/tldr-python-client: Added a search function in order to help locate the appropriate subcommand for the situation in order to reduce someone reaching for the web browser and context switching in general

Why Tor?

The Tor Project has been helping people all around the world become anonymous users of the Internet to access Internet resources in a free, safe and secure manner (including myself). Anonymity is a valuable tool for those who need free access to information, and for those wishing to disclose information without severe repercussions to them or their loved ones. To that end, any improvements that I can make towards that goal is a noble pursuit and should be done.

Rewriting the Tor daemon in Rust would go a long way towards providing a safer experience for Tor users, as the project has already identified, and accelerating this process by dogfooding APIs and providing valuable example code is a welcome improvement I am more than willing to make.