OONI Project Plan

Goal

The goal of this project plan is to illustrate the state of the software project at beginning November 2012 and plot the next steps in reaching the milestones that are to come.

OONI is composed of two major software components: **OONIProbe** and **OONIB.**

Timeline

Alpha release

Implement in alpha quality the following OONIProbe tests:
DNS Probing, HTTP Requests, URL Lists, Captive Portal, Keyword filtering, HTTP Host

We should implement a basic implementation of test helper for HTTP Request, DNS Probing and HTTP Host.

Reports shall get generated to flat files on the file system, their markup must be valid.

Test templates for the HTTP Protocol and Scapy shall be implemented and documented.

Beta release

Implement the following tests: TTL Walking and Network Latency.

Improve and bugfix previous tests following hands-on experience.

Implement the backend reporting system that allows the submission of reports from OONIProbes.

Implement test helper subsystem for HTTP Host, DNS Tamper, Two way traceroute and Network latency.

Early M-Lab testing.

Reports that get generated on the client shall be pushed to the remote backend.

Stable release

Implement the user interface for OONIProbe and the OONIProbe API.

Full M-Lab deployment.

Do quality assurance work on OONIProbe to verify that everything is working as it should.

Have a client that people can install on Windows and OSX.

Start creating builds for Windows and OSX.

Project Status Summary

This section highlights current project status.

TODO: We must make a review of milestones and tickets, so that each activity is properly tracked with an assigned ticket.
https://trac.torproject.org/projects/tor/wiki/org/sponsors/SponsorH

OONIProbe

The test writing API and the reporting system have been prototyped and partially documented. HTTP and Scapy test templates are implemented. Most tests are implemented as a prototype or alpha quality.

sub-deliverable	status summary	status	next action
Reporting	prototype	The reporting engine creates YAML formatted reports	Verify that the generated reports are valid. Extend the reporting engine to support packet

			captures
Testing API	prototype	The test writing API is implemented and partially documented. We have test templates for HTTP and scapy based tests	Port legacy tests to the new API
HTTP Test template	alpha	This is implemented with hooks for processing headers and	Add hooks for sending a string in the request body
Scapy Test template	prototype	This is implemented with support for writing to a PCAP file and reading of test inputs	Parametrize the location of where the PCAP file is stored.
Logging subsystem	alpha	Messages printed via log.* are collected in a log file that is rotated daily	
Config	not		Add

file support	implemented		support for config file based test para meter specification
HTTP Requests via SOCKS5 client	not implemented		Implement a socks5 client and hooks in the HTTP API to allow requests over Tor
TTL Walking	early prototype	The test is implemented in twisted not using the OONI API	Implement such test using the new API
Keyword injection	prototype		Document the running of such test
DNS Probing	prototype	This test is implemented using the new API	Document the running of this test and verify it's functionality.
HTTP Requests	not implemented	This test is not implemented using the new API.	Implement such test using the HTTPT API

HTTP Host	alpha	This test is implemented using the new API	Verify and document the funcionality of this test
URL Lists	prototype	This test is implemented using the new API with support for censorship detection based on redirect	Document the running of this test.
Squid Proxy detection	alpha	This test is implemented	Document the running of the test
Network Latency	not implemented		Implement such test using the new API
Captive Portal	alpha	This test is implemented using the new API	Document the running of this test
RST Packet detection	early prototype	This test is implemented using the *legacy* API	Port this test to the new API
Netalyzr	prototype	This test is implemen	Add the content of a netalyzr

		ted using the new API	run to the report
daphn3	early prototype	This test is implemented using the *legacy* API	Port this test to the new API and verify it's functioning
BridgeT	prototype	This test is implemented using the *legacy* API	Port this test to the new API and verify it's functionality
TCP connect	early prototype	This test is implemented using the *legacy* API	Port this test to the new API and verify it's functioning
OONIProbe API	not specified		Specify how OONI Probes can be controlled remotely via an HTTP based API that runs on a Tor Hidden Service
Update system	not specified		Specify how the update of OONI should happen

Build system	not implemented		Build OONIProbe using APAF
User Interface	not implemented		Implement javascript based user interface

OONIB

The API for the reporting system has been specified and some basic test helpers have been prototyped.

sub-deliverable	status summary	status	next action
Reporting collector	specified	The report collector HTTP API has been defined, the technological selection has been done.	Implement collector that writes submitted reports to a database
Test Helper - collector hooks	not implemented		When a report containing the test_helper field, the backend must be able to link this to the client that is performing the request
Tor Hidden Service support	not implemented		The HTTP API should be exposed as a Tor Hidden Service
DNS Test Helper	early prototype	A DNS proxy is implemented as a service and is started	Hook this DNS proxy to the test helper - collector hook
HTTP Test Helper	early prototype	A HTTP server that contains the request	Hook the HTTP Test Helper to the reporting collector

		headers it received is implemented. A HTTP Server that returns a random web page.	
Two way traceroute	not implemented		Implement such test
Daphn3	prototype	Given a pcap file it is possible to recreate the server-side message exchanges	Hook this to the reporting API allowing to submit when creating a report the pcap of the server side packets to be sent
SSL Support	prototype	It is possible to run the HTTP based services wrapped in SSL given a server PEM certificate	Parametrize the cipher suites that are advertised to the client

OONIProbe

This is the software component that people interested in performing censorship related measurements will end up running.

Testing Framework

This is the interface that a developer must use when developing an OONI test. When developing a test using such API they get the advantage of having the test result be structured using the standard YAML format adopted by OONI-probe.

Reporting

The reporting system must generates reports using the YAML markup format. They must include the IP Address of the client performing the test, the ASN number of the user performing the test and the country code.

Current status:
Reports are written to filesystem. The filename contains the timestamp of when the test was executed. For every test timestamps are recorded of when the tests started and when the test ended.

Next steps:

- Verify that the markup is valid and passes parsing

    - priority: high
    - effort**:** 3h

- Have a full PCAP of the sent and received packets from the probes network POV
    - priority: medium
    - effort: 10h

- Write unittests for the reporting API
    - priority: medium
    - effort: 10h

- Fully document the report format
    - priority: medium
    - effort: 5h

- Make reports be sent to the collector backend
    - priority: medium
    - effort: 15h

Testing API

The testing API is the interface that is exposed to developers interested in implementing OONI Tests.

Current status:
The test API is implemented and documented here: http://ooni.readthedocs.org/en/latest/writing_tests.html
Through the test API it is possible to specify which command line arguments are supported by the test, what filename the report should be written to and where the log should be written to.

Based on the Testing API there are a set of Test Templates that bootstrap the process of implementing a test.
Some unittests for such API are implemented here:
https://gitweb.torproject.org/ooni-probe.git/blob/HEAD:/tests/test_inputunit.py

Next steps:

- Add checks to verify that required command line arguments are being passed to tests
    - priority: high
    - effort: 8h

- Port all tests written using the *legacy* API to the new API
    - priority: high

    - effort: 20h

- Remove all the old code that is of support to the *legacy* API
    - priority: medium
    - effort: 6h

- Write unittests for the new API
    - priority: medium
    - effort: 10h

- Write documentation on how to implement a simple test using our Testing API
    - priority: medium
    - effort: 5h

- Design and implement command line argument for specifying what test backend to use
    - priority: medium
    - effort: 5h

- Test specifications should be built from source
    - priority: medium
    - effort: 10h
    - https://trac.torproject.org/projects/tor/ticket/7223

- Detect if the client performing the test is behind NAT
    - priority: medium
    - effort: 15h
    - https://trac.torproject.org/projects/tor/ticket/7228

HTTP Test template

Such test template is implemented. There are hooks for allowing the processing of HTTP headers and of the body of the response with the backend. There is also a hook for processing redirects.

Next steps:

- Document how to write a test using such template
    - priority: medium
    - effort: 10h

Scapy Test template

Such test template is implemented in alpha quality status.
It takes as input scapy packets and spawns threads for handling the sending and receiving of packets.

Based on the class attributes it is possible to tell the test template to have the test wait for it to receive all the responses or to go in timeout after a certain amount of time. The packets that are sent by scapy and the ones that it received are stored inside of a PCAP file.
The path to such PCAP file is specified via a class attribute.

Next steps:

- Name the PCAP file using the same convention of the test report and include such file path in the report
    - priority: high
    - effort: 5h

- Document writing tests using such API
    - priority: medium
    - effort: 5h

Logging subsystem

This allows the collection of logs on the probe running the test for debugging purposes.

The log file is parametrized and can be set from the configuration file.

Config file support

This should allow parameters related to the running of an OONIProbe to be configured via config file.
The categories of configuration parameters are:

- General
- Debug
- Test specific

The test specific parameters should allow the configuration of all the parameters that can be passed as command line arguments to tests.

Current status:
Not implemented

Next Steps:

- Implement test specific config file support
    - priority: low
    - effort: 12h

HTTP Requests via SOCKS5 client

The HTTP Test template should provide a simple way of performing HTTP request with the support of a SOCK5 compliant server. This will allow us to perform measurements over the Tor network.

Current status:
Not implemented

Next steps:

- Implement SOCKS5 Client
    - priority: high
    - effort: 5h
    - notes: https://github.com/globaleaks/Tor2web-3.0/blob/master/socksclient.py

- Add hooks in the HTTP API to allow requests via a SOCKS5 client
    - priority: high
    - effort: 10h

Tests

TTL Walking

Priority: high
These is an implementation of this here:
https://gitweb.torproject.org/ooni-probe.git/blob/HEAD:/old-to-be-ported-code/very-old/traceroute.py
Needs to be re-written, based on scapyt.
This tests should do a multi-protocol multi port traceroute as is detailed here: [insert link]

Work to be done:

- Implement such test
    - priority: high
    - effort: 10h

- Document such test
    - priority: medium
    - effort: 3h

Keyword injection

Priority: high
We want to have the ability to take a list of keywords and place them inside of any kind of request.

The basic that will be part of the alpha release is should be one that places the keyword in the body of a HTTP POST request and as parameters in a GET request.
Such tests should be implemented based on the HTTPT API and testing such test based on the HTTPT.
https://trac.torproject.org/projects/tor/wiki/doc/OONI/Tests/KeywordFiltering
Current status:
Such test is implemented in alpha quality

Work to be done:

- Document such test
    - priority: medium
    - estimated effort: 3h

DNS Probing

Priority: high
Current status:
This test is implemented at an alpha level quality.

Work to be done:

- Verify functionality of such test
    - priority: high
    - estimated effort: 5h

- Document the running of such test
    - priority: medium
    - estimated effort: 5h

- Write unittests for such test
    - priority: low
    - estimated effort: 3h

HTTP Requests

Priority: high
This test involves performing HTTP requests to a set of hostnames specified as input.
This test is also known as Header Field manipulation.

The kinds of HTTP requests we will be performing are:

- All standard HTTP methods with variations on capitalization
- The payload of the HTTP request being sent should be customizable

Current status
This test is not implemented.

Work to be done

- Clean up the trac page for test renaming Header Field Manipulation to HTTP Requests. Remove the HTTP Scan test as it is redundant.
    - Priority: high
    - Effort: 3h

- Implement such test based on HTTP test template
    - Priority: high
    - Effort: 10h

- Verify functionality of such test
    - Priority: high
    - Effort: 5h

- Document the running of such test
    - Priority: medium
    - Effort: 5h

URL Lists

Priority: high
This involves performing HTTP GET request towards the set of inputs to the test.

Current status:
Alpha quality.
This test performs HTTP requests to a given as input list of URLs and logs all the requests and responses to the OONI Report.
This test is implemented using the new API with support for censorship detection based on redirect.

Work to be done:

- Verify functionality of such test
    - priority: high
    - effort: 5h

- Document the running of such test
    - priority: medium
    - effort: 5h

Squid proxy detection

This test involves doing network measurements aimed at detecting the presence of a squid transparent HTTP proxy.
Current status:
Implemented

Network Latency

Priority: high
This involves computing the RTT of raw IP frames that are sent to the specified host.
This test requires a backend.

Work to be done:

- Implement such test
    - priority: high
    - estimated effort: 10h

- Verify functionality of such test
    - priority: high
    - estimated effort: 3h

- Document such test
    - priority: medium
    - estimated effort: 3h

- Write unittests for such test
    - priority: low
    - estimated effort: 3h

Captive portal

Priority: medium
This involves emulating what standard captive portal detection software would do to detect the presence of a captive portal.

Current status:
* Fully implemented

Work to be done:

- Verify functionality of such test
    - priority: high
    - effort: 3h
- Document running of such test
    - priority: medium
    - effort: 5h
- Write unittests for test

    - priority: low
    - effort: 2h

RST Packet detection

Priority: medium
This involves taking as input a certain packet and measuring if what we get back is a RST packet.
Current status:

Work to be done:

- Verify functionality of such test
    - priority: high
    - effort: 3h

- Document the running of such test
    - priority: medium
    - effort: 3h

- Write unittest for such test
    - priority: low
    - effort: 2h

Netalyzr

Priority: low
This involves running the netalyzr tool and collecting the result of an execution of such test.

Current status:
Such test is implemented

Work to be done:

- Collect the result from a netalyzr run and add it to the report
    - priority: high
    - effort: 5h
- Verify the functionality of such test
    - priority: high
    - effort: 3h
- Document the running of such test
    - priority: medium
    - effort: 3h

BridgeT

Priority: low
This involves performing a series of measurements to verify the functioning or not of Tor bridges.

Current Status:
Prototype

Work to be done:

- Verify the functioning of such test
    - priority: high
    - effort: 3h
- Document the running of such test
    - priority: medium
    - effort: 3h
- Write unittests for test
    - priority: low
    - effort: 2h

High priority effort: 3h
Medium priority effort: 3h
Low priority effort: 2h

Daphn3

Priority: low
This involves using the bisection method on a known censored client server packet exchange to understand where the censor is triggering censorship based on DPI.

Current Status:
Implemented in alpha status

Next steps:

- Verify the functioning of such test
    - priority: high
    - effort: 3h
- Document the running of such test
    - priority: medium
    - effort: 1h
- Write unittest for test
    - priority: low
    - effort: 2h

OONIProbe API

This is the API that will allow to remotely control the execution of a test that is installed on the machine. Every OONIProbe will expose a Tor Hidden Service that will allow tests to be run with an input set of choice.

Next steps:

- Specify the HTTP interface for instructing a probe to run a certain test with a specific input set
    - priority: high
    - effort: 10h
- Implement this HTTP API as an APAF service
    - priority: medium
    - effort: 30h

Build system

Tests written using OONI must be shipped to clients in compiled form. They must support Windows, OSX and Linux. For these requirements the ideal choice is to use the APAF since OONI is twisted driven and uses the same technological stack as applications that are to be built using APAF. This will require work on APAF.

Next steps:

- Setup build environment for Windows
    - priority: high
    - effort: 20h

- Setup build environment for Mac OS X
    - priority: medium
    - effort: 20h

- Setup build environment for Linux
    - priority: low
    - effort: 20h

- Build OONIProbe using APAF for Windows
    - priority: high
    - effort: 20h

- Build OONIProbe using APAF for Mac OS X
    - priority: medium
    - effort: 30h

User interface

The user interface for OONI will be written as a javascript web application. This will allow us to not have to worry about issues related to the GUI interface of the the various platforms we plan to support. In particular all major GUI libraries have their own event loop that does not cooperate well with the event loop of twisted.

Next steps:

- Make mockups of the User Interface for OONI
    - priority: medium
    - effort: 20h

- Implement in HTML5/JS the mockups
    - priority: medium
    - effort: 25h

- Implement client side logic for interacting with the OONIProbe API
    - priority: medium
    - effort: 30h

OONIB

This is the backend component of OONI. It is responsible for receiving the reports sent by OONIProbe and exposing the *Test Helper* system.

Reporting collector

The reporting system will expose an HTTP API that collects reports coming in from clients.
The API is specified here: https://trac.torproject.org/projects/tor/wiki/doc/OONI/Backend
This is being implemented using Storm and cyclone.

Current status:
The API for submission has been specified. The technological selection has been done.
The backend takes a configuration file that allows to specify on which ports the various test helpers should be listening on.

Next steps:

- Prototype HTTP based collector for submission of reports created by OONIProbes and third party tests
    - priority: high
    - effort: 15h

- Add database support to OONIB
    - priority: high
    - effort: 30h

- Write submitted reports to database
    - priority: high
    - effort: 15h

- Write submitted reports to flat files
    - priority: high
    - effort: 20h

- Attach to the reports the data that is collected from the collector point of view
    - priority: high
    - effort: 20h

- Consolidate logging infrastructure of OONIB
    - priority: medium
    - effort: 10h

- Add support for configuration parameter to be set on test helpers
    - priority: low
    - effort: 10h

Test Helper - collector hooks

When a report for a Test that requires a test helper is created a temporary entry in the test helper database is created containing a reference of what is the test helper that the client is interested in using and their IP address. This is necessary for linking the report from the collector point of view to that of the client point of view.

Current status:
Not implemented

Next steps:

- Create prototype of test helper - collector mapping
    - priority: medium
    - effort: 30h

- Document how to interact with a collector that supports test helpers
    - priority: medium
    - effort: 5h

Tor Hidden Service support

The collector will expose a Tor Hidden Service.

Next steps:

- Using APAF make the OONIB collector start a Tor Hidden Service
    - priority: medium
    - effort: 10h

Build system

Since we are basing the Test Helper on APAF we can get for “free” the building of such reporting backend for cross platforms.

Next steps:

- Build OONIB using APAF for Windows
    - priority: medium
    - effort: 20h

- Build OONIB using APAF for Mac OS X
    - priority: medium
    - effort: 20h

- Build OONIB using APAF for Linux
    - priority: medium
    - effort: 20h

Test Helpers

Test helpers are the backend component that supports the running of tests that require such infrastructure. Test helpers must log all the requests that they receive from probes and add them to the final report that is stored on the OONI backend.

DNS Test Helper

Current status:
There is a prototype that proxies DNS queries that are incoming to a DNS resolver that is hardcoded.

Next steps:

- Implement basic DNS Test helper that logs to disk the DNS requests it receives from clients.
    - priority: medium
    - effort: 10h

- Parametrize the DNS server that the test helper uses
    - priority: medium
    - effort: 10h

HTTP Test Helper

The HTTP Test helper is to provide support to HTTP based tests that require a HTTP backend infrastructure.

Current status:
basic prototype.
A HTTP server that contains the request headers it received is implemented.
A HTTP Server that returns a random web page.

Next steps:

- Implement basic implementation for HTTP Request test helper that returns to the client all the requests it has received.
    - priority: medium
    - effort: 10h

- Implement test helper for HTTP Requests Test
    - priority: medium
    - effort: 10h

Two way traceroute

When a client requests a two way traceroute test helper it a traceroute from the collector to the client should be instantiated.

Current status:
Not implemented

Next steps:

- Implement such test
    - priority: high
    - effort: 20h

- Document the running of such test
    - priority: high
    - effort: 10h

Daphn3

Daphn3 allows to detect censorship using the bisection method. It takes as input the pcap file of a known censored client server packet exchange.

Current status:
Prototype
The pcap is specified via a config file and the test helper will recreate the server-side message exchanges.

Next steps:

- Verify the functioning of such test helper backend
    - priority: low
    - effort: 20h
- Allow the pcap file to be sent via the OONIB collector API
    - priority: low
    - effort: 20h

SSL Support

On request it should be possible to start on the backend an SSL server that will log the incoming SSL requests.

Current status:
basic Prototype

Next steps:

- Parametrize the possible SSL parameters of the server
    - priority: medium
    - effort: 10h
- Allow the specification of which parameters the SSL server should have via the collector API
    - priority: medium
    - effort: 10h

Resources

This enumerates the currently allocated resources on the project:

Person	Hours worked
Arturo	XXX
Isis	XXX
Jacob	XXX