Scrum iteration I

Rolf Jagerman, Laurens Versluis and Martijn de Vos

May 15, 2014

1 Introduction

We decided to do sprints of two weeks, where every day we perform our daily scrum. At the end of each week we evaluate the current goals and where we stand. Based on our current progress we decide the feasibility of the goals and adjust these goals if needed. In this chapter we describe our first sprint, starting in week 4.3 (5^{th} of may) and ending in week 4.5 (16^{th} of may).
First, we describe our goals and milestones we had initially set for this sprint. Then we evaluate the first week of the sprint and describe the adjustments, if any. After that, we talk about our productivity, where we explain what we have done, how we did it, if it's dependent on anything and what part of the final product it fulfills.

2 Goals and milestones

In this section we describe which goals were set and the milestones that were created on the Version Control System Github. The goals are often represented in different subgoals that are translated into milestones. We use milestones to separate different subgoals such as creating a small GUI from updating existing code. This sprint, we had the following goals: port the packages we need to Python for Android and understand, read and play with the current code. We translated this goal into two milestones for this sprint:

1. Get all packages that are needed to run the anonymous tunnels working on Android.

2. Implement a basic GUI to test with. This GUI should be created with Kivy.

Each issue we identified, was inserted in Github and assigned to one of our milestones. This allowed us to see the progress throughout the sprint. These issues, as well as the milestones on the Github page, allowed us to easily track, discuss and check off issues.

3 Our work

The Python for Android framework allows developers to add existing Python packages by creating so called recipes. As most of the packages were not available or were custom made, we had to build a lot of these recipes ourselves.

A recipe downloads a package, extracts the contents and applies operations on them when required, written by the developer himself.

Python for Android allows to build these packages under the ARM architecture so they won't compile for the desktop architecture which is not suitable for Android development. By simply using the 'push arm' and 'pop arm' commands in the recipe files this can be achieved. Because of this easy integration and the fact that Tribler and the code for the anonymous tunnels was written in Python, this is a good environment to develop our application.

Below are the packages described we use in our application and in most cases had to create a recipe for. We describe the functionality of the package in our application and which dependencies this package has.

- Kivy
  We use Kivy as framework for creating the GUI. Kivy is an open source software library for creating Natural User Interface (NUI) applications. It's easy to use and cross-platform, allowing users to create a GUI on their PC and then integrate it in their products (we integrate it in our Android application). As Kivy is integrated in Python for Android, it's a natural choice to use it for create the GUI. Kivy is dependent on Python, as it's a Python package.

- OpenSSL
  OpenSSL is the world's most famous open source cryptography toolkit, also available for Python. As our application makes use of PyCrypto and M2Crypto which are both dependent on openSSL, we have to include it in our app.

- M2crypto
  Dispersy and Tribler are dependent on M2Crypto as it has some security features which M2Crypto implements such as elliptic curves cryptography. As the anonymous tunnels, our core function of the application, are dependent on both Dispersy and Tribler we also need M2Crypto. M2Crypto itself is dependent on Python and OpenSSL as it's a Python package using the OpenSSL implementation.

- PyCrypto
  PyCrypto is a Python library which implements certain cryptography functions used by the tribler_core_minimal package. PyCrypto itself depends on functionality of the OpenSSL package.

- Boost
  The Libtorrent library is used to download torrent files with the Tor protocol. To compile Libtorrent, the boost package is required. Boost is a huge C++ library which adds code for multithreading, regex, math and asynchronous operations. We have to compile Libboost with Python bindings enabled so we can invoke the library from Python code.

- Netifaces
  Netifaces allows to easily get the address(es) of the machine's network

interfaces from Python. It's therefore dependent on Python and used by the Dispersy package.

- Zope
  Zope is a open source web framework for object-oriented web application servers. The Twisted package makes use of this framework for asynchronous networking.

- Twisted
  Twisted is an extensible framework for asynchronous networking written in Python. The framework has special focus on event-based network programming and multi-protocol integration. It has dependencies on the Zope framework.

- Anontunnels
  This package is the core of our application and contains the code needed for the anonymous tunnels. This package actually contains another package: the support for the Socks5 proxies. The files for this package come from pull request 525 on the Tribler Github.

  We made some minor changes to this code: the Main.py file has been removed from the package and the class definition of AnonTunnel has been moved to it's own file (atunnel.py). We import this file in our main.py so we can use the AnonTunnel class. We also adjusted the master key in community.py so we have our own community to test communication between devices.

- Tribler_core_minimal
  The anonymous tunnels are using some parts of the core of Tribler. These files have been bundled in the tribler_core_minimal package. We've looked closely to the various imports the code for the anonymous tunnels is using and when a script imports a script from the Tribler core, it is added to the tribler_core_minimal package. For example, this package contains the RawServer and the SocketHandler classes. It also contains the code needed for the cryptography such as support for elliptic curve cryptography and ELGamal.

- Dispersy
  Since the code of the anonymous tunnels are using Dispersy for node discovery and data synchronization, we've created a Python package with all the code that's needed for Dispersy. This package does not depend on other custom packages we made and can be used standalone.
  The files we have bundled are from pull request 525 on the Tribler Github. We didn't use the files from the official Dispersy Github because this build was missing some classes we needed (for example, the decorator.py). Besides that, some changes have been made to the Dispersy core to add support for the anonymous tunnels.

3.1 Porting the anonymous tunnels to Android

In deze subsectie iets over de anontunnels die we hebben geport naar android, wat over Python for Android en de packages. Misschien de packages in subsubsections bespreken?

3.2 Creating a GUI with Kivy

The first version of the anonymous tunnels used the standard output for printing information about what's going on. This required the phone to be connected to a computer so we can examine the log with the adb logcat tool. That's why we decided to create a graphical user interface for our application. The purpose of this application is to provide a button to start the tunneling and a log to see on the screen what's happening.

Creating a GUI was a small stap for us: we already included the Kivy package in our Python for Android distribution. Since Kivy is a GUI framework for desktop (but has been ported to Python for Android), we first tried to create a desktop interface. Creating interfaces in Kivy is quite straightforward and is related to creating user interfaces in Android: you specify your layout elements in Kivy files which have the kv extension. In the main Python file, you load this interface file and you can access properties of the UI elements.

3.3 Attempt (?) to compile Libtorrent for Android

Hier iets over onze uitdaging om Libtorrent werkend te krijgen in Python for Android

4 Conclusion

Hier komen onze conclusies over deze sprint (wat ging er goed/fout, wat willen we anders doen, wat gaan we volgende sprint doen etc.)