GSOC - 2018 Project Proposal

OpenWISP IPAM

(IP Address Management tool for OpenWISP 2)

Personal Information
Name - Anurag Sharma
Email - anssharma61gmail.com
Phone - +91 9741322480
Github/Gitter.im username - anurag-ks
IRC nickname - anurag-ks
Website - anurag-sharma.me

Preferred Method of Contact -

- Easy to reach out through Emails, Gitter and IRC.
- Between 9:00 am to 12:00 pm and 2:00pm to 1 or 2 am (in IST)

Skills -

- Proficient in Python and Django
- I also have a good amount of experience in JavaScript technologies(Angular, Ionic, NodeJS etc)
- Java and C++ most used in competitive programing.

Development Environment - Elementary OS (Ubuntu Xenial) 64-bit

Contributions to OpenWISP - I have already started contributing to django-freeradius project.

- https://github.com/openwisp/django-freeradius/pull/101 (merged)
- https://github.com/openwisp/django-freeradius/pull/100 (merged)
- https://github.com/openwisp/django-freeradius/pull/98 (merged)
- https://github.com/openwisp/django-freeradius/pull/107(merged)
- https://github.com/openwisp/django-freeradius/pull/111 (approved)

About Me

I am a 19 year old undergraduate, currently doing my Bachelors in Engineering in MIT, Manipal, India. I have a passion/love for programming and developing softwares since my school days(9th grade to be specific). At a school level, I started contributing to the open source world and Google Code In helped me a lot in this. I took part in two editions of **Google Code In 2013-14** and **2014-15**, during which I contributed to **Sahana Software Foundation**. I specifically worked on Sahana Eden which was a “disaster management system” web application written in Web2Py and with the help of some wonderful mentors I managed to become a **Grand Prize Winner** two times(could have gone again but then the rules changed).

Apart from GCI, I have made several personal projects, one of them was an online judging system. A basic django application to compile, execute and test computer programs on a django server. It was used in my school during a competitive programming event.

Currently I work as a developer at **Manipal The Talk Network**, it’s a student run media organization. Here I am developing a cross platform mobile application for them using the Ionic Framework. I also gave talk on Django framework in my first semester during MuPy(Manipal’s version of PyCON).

Apart from all the coding life, I love playing Chess and Soccer, and also I am a hobbyist Illustrator.

Motivation

I developed my interest in open source development since my school days and taking part in **Google Code In** helped me learn a lot. I see **Google Summer of Code** as a next step in my career and that’s why I am interested in taking part in GSoC 2018.

Networking has been one of the field that interest me a lot and I wanted to learn more about it. **OpenWISP** is the perfect platform for me to learn more and develop more open source software parallely. Having worked with Django and Python for quite some time now also motivated me in selecting OpenWISP as my GSoC organization.

Project Details

Aim - This project idea consists in creating a new **OpenWISP 2 IPAM** module with the basic features for IP Address Management.

Possible Mentors - Federico Capoano, Marco Giuntini

The project is divided into two phases -

- **Phase 1 - Developing django-ipam**
- **Phase 2 - Integration of django-ipam in OpenWISP 2**

Phase 1 -

- IPv4 and IPv6 IP address management
- Section / Subnet management with nested subnets
- Automatic free space display for all subnets
- Visual display for a specific subnet
- IP request module (similar to phpipam)
- RESTful API to for CRUD operations (using session authentication and django model permissions for authorization)
- Possibility to search for an IP or subnet
- CSV Import and Export of subnets and their IPs (using the same format of phpipam)

Additional features -

- abstract abase models
- reusable views for its API
- reusable admin classes
- reusable test classes (as in django-netjsongraph and django-freeradius)

Phase 2 -
In this phase we will integrate *django-ipam* with the rest of the OpenWISP 2 ecosystem
This will require creating a module called *openwisp-ipam* which will wrap and integrate django-ipam in the rest of the OpenWISP 2 ecosystem, in a very similar way as openwisp-controller integrates django-netjsonconfig, and openwisp-network-topology integrates django-netjsongraph.

Measurable Outcomes

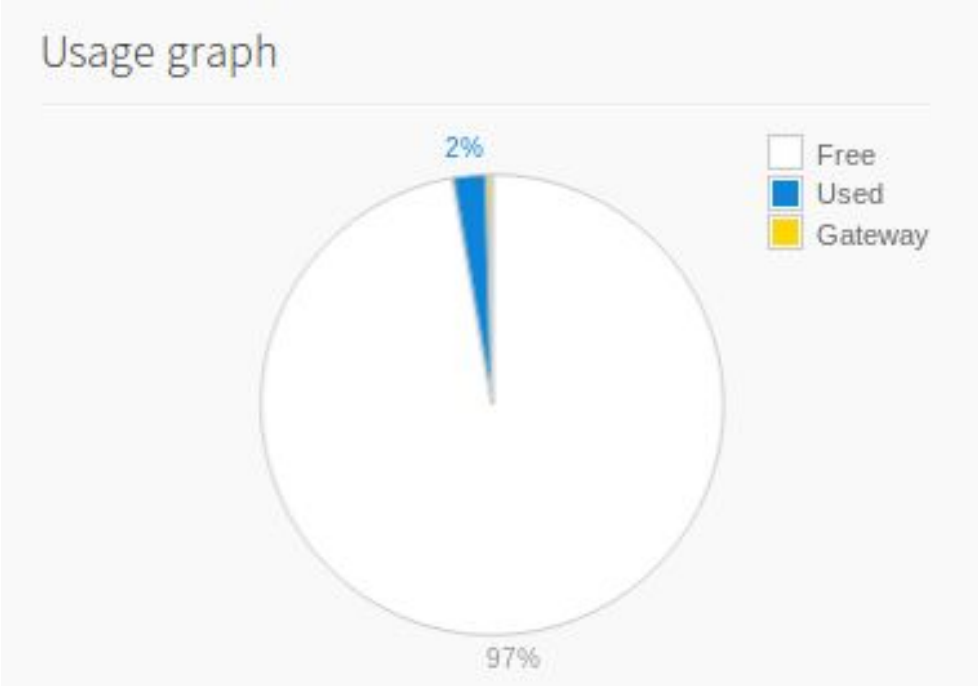
- complete items described in phase 1
- complete items described in phase 2
- provide good documentation for django-ipam in form of a README.rst (using ResStructuredText), explaining how to install, run tests, configuration details, screenshots and explaination of the main features
- provide basic developer documentation for openwisp-ipam in form of a README.rst (using ResStructuredText), explaining how to install and run tests
- achieve a test coverage higher than 90% in both modules

Achieving Project Goals

Phase 1 -

- **IPv4** and **IPv6** IP address management
    - This would involve creation of base model classes for IPv6 and IPv4 IP addresses.
    - Register models in the admin app for basic **CRUD** operations through the admin portal.
    - Following could be a tentative structure of the models -
        - IP address field
        - Short Description
        - Owner
        - Gateway
        - Host
        - Netmask
        - Device
        - Subnet (ORM relationship to be used over here)
        - Used - (to check if the ip address has been taken or not, will have to figure out how to make a check for this)
- **Subnet Management -**
    - Creation of base model for subnets.
    - Register models in the admin app for basic **CRUD** operations
    - Following could be a tentative structure of the models -
        - Subnet name
        - Master subnet (If we want to create nested subnets)
        - Subnet description (A short description about the subnet)
        - Subnet Usage - (Total space available, used etc.)

- **Automatic free space display -**



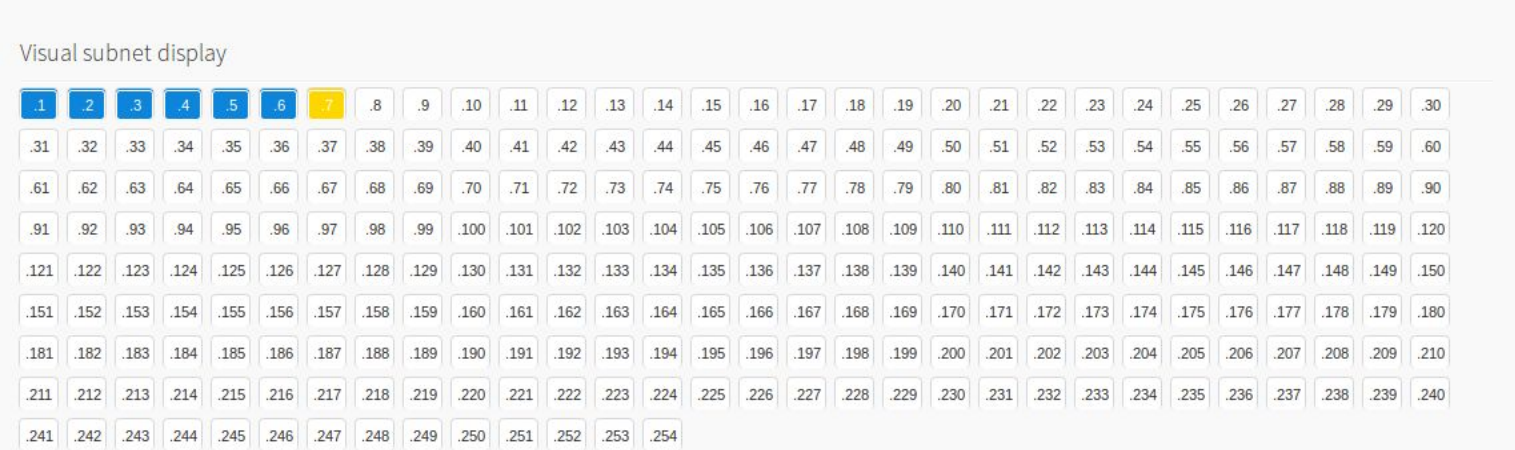
  - We will make something like this for the visual display of a selected subnet.

  - The following example was taken from phpipam, which displays a pie chart of free and used space in a particular subnet.

  - For this particular task, we can make use of **django-chartit** module to create charts on the admin webpage.

    - Github link - https://github.com/chartit/django-chartit

- **Visual display for a specific subnet -**
    - This would be visual representation of list of IP addresses available inside a specific subnet.
    - For example -



    - This would involve editing the base admin template with some html and css with the help of an ORM query.

- **IP request module -**
    - This feature would allow users to request an IP address from a specific subnet.
    - For this we would query for the first unused IP address belonging to our selected subnet with the help of **Django's QuerySet API.**

```
Subnet.objects.filter(ip_address__used=false)[:1].get()
```

- **RESTful API to for CRUD operations -**

    - We would build an api around django-ipam for CRUD operations related to our models.
    - We can make use of **django-rest-framework** as it has been used previously in django-freeradius to build our API.
    - The generic views provided by REST framework allows us to quickly build API views that map closely to our database models.

    - Example -

```
class UserList(generics.ListCreateAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = (IsAdminUser,)

    def list(self, request):
        # Note the use of `get_queryset()` instead of `self.queryset`
        queryset = self.get_queryset()
        serializer = UserSerializer(queryset, many=True)
        return Response(serializer.data)
```

    - We would also implement features like **pagination** and **filtering** like we have done in django-freeradius.

        - https://github.com/openwisp/django-freeradius/pull/101
        - https://github.com/openwisp/django-freeradius/pull/100
    - Our api could be restricted using token authentication api provided by DRF.
    - Add permissions to the class based api views, for example -

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView

class ExampleView(APIView):
```

```
    permission_classes = (IsAuthenticated,)
```

- More information about DRF's authentication can be found over here - http://www.django-rest-framework.org/api-guide/authentication/

- **Possibility to search for an IP or subnet** -
  - Make use of the search_fields attribute of the ModelAdmin.
  - An example usage of that attribute -

```
class SubnetAdmin(admin.ModelAdmin):
    ...
    search_fields = ('name', 'description', 'ip_address', )
```

  - We can also go for an elasticsearch or django haystack for more efficient and advance searching methods.

```
from django.contrib import admin
from haystack.admin import SearchModelAdminMixin
from .models import MockModel


class MyCustomModelAdmin(admin.ModelAdmin):
    pass


class MockModelAdmin(SearchModelAdminMixin, MyCustomModelAdmin):
    haystack_connection = 'solr'
    date_hierarchy = 'pub_date'
    list_display = ('author', 'pub_date')


admin.site.register(MockModel, MockModelAdmin)
```

  - http://django-haystack.readthedocs.io/en/v2.8.0/admin.html#django-admin-search

- **Import and Export of IP addresses and subnets** -
    - Taking inspiration from phpipam we would follow the following template for export and import -

Template

The import XLS/CSV should have the following fields and a **header row** for a succesful import:

section *	IP address *	Hostname	description	vrf	subnet *	mac	owner	device	note	tag	Gateway	Location	Ticketnummer	Site	Inuse	Valid until
varchar(128)	varchar(100)	varchar(100)	varchar(64)	varchar(32)	varchar(255)	varchar(20)	varchar(32)	varchar(100)	text	varchar(32)	tinyint(1)	varchar(256)	varchar(128)	set('site1','site2')	tinyint(1)	date

    - We can make use of django-import-export module - http://django-import-export.readthedocs.io/en/latest/
    - **django-import-export** is a Django application and library for importing and exporting data with included admin integration.
    - A very basic implementation for admin import-export would be something like this -

```
from import_export.admin import ImportExportMixin
class SubnetAdmin(ImportExportMixin, admin.ModelAdmin):
       pass
```

- Additional features like base models, reusable API classes and reusable test cases will be implemented by taking inspiration from **django-freeradius** and **django-netjsongraph.**

Phase 2 -

This phase mostly involves integration of **django-ipam** in the openwisp environment. This will require creating a module called ***openwisp-ipam*** which will wrap and integrate django-ipam in the rest of the **OpenWISP 2** ecosystem, in a very similar way as **openwisp-controller** integrates **django-netjsonconfig**, and **openwisp-network-topology** integrates **django-netjsongraph**.

Approach

- Create a new django reusable app - **openwisp-ipam**.
- This will be wrapper around django-ipam, so we will be subclassing already created classes for models, API, management commands etc. from **django-ipam.**
- Also implement multi-tenancy using **openwisp-user** and **openwisp-utils**.
- Create **setup.py** for installation of this newly created openwisp-radius module.
- Writing tests and keeping the coverage at 100%.

Testing and Documentation

Achieve a test coverage higher than 90% in both modules -
This can be done by constant writing tests for every new module/feature implemented in out app.

1. Create a reusable test suite for django-ipam.
2. Write all possible test cases in the test suite.
3. Create a **runtest.py** script to run the test suite.

Documentation -

1. Create a good quality README file in **rst** format for django-ipam explaining how to install, run tests, configuration details, screenshots and explanation of the main features.
2. We can also take inspiration from here - https://github.com/matiassingers/awesome-readme in order to create an “awesome” readme file.
3. Create developer docs in order to help new developers get started with contributions with the help of proper contributing guidelines.

Proposed Project Timeline

Currently (March, 2018) I am doing my second semester, which will be over by April 30th, 2018. After that I head back home, till July/August.
My End sem exams will be going on between 16th April to 30th April and then including the traveling time, I might be inactive between 16th April and 3rd May.
After that I will be able to devote my time primarily to GSoC weekly.

- Community Bonding - April 23, 2018 - May 14, 2018
    - During this period, my focus would primarily be towards integrating myself into the OpenWISP community and learning more about it.
    - Focus on learning more about IP address and Subnet masking methods.
    - Also discuss the above mentioned solution with my mentors and finalize on it.
    - Creating a wireframe of our plans, and setup directories or repositories for project work.

Coding Phase

(May 14, 2018 - August 6, 2018)

PHASE 1 -

- Week 1 - May 14, 2018 - May 21, 2018
  - Setup development environment. (Basically updating to latest version)
  - Create scaffolding app and other base scripts.
  - Build the basic model structure and test it out using the django's admin interface.
  - Also include theme from openwisp-util for the admin interface.

- Week 2 - May 21, 2018 - May 28, 2018
  - Create models for subnets.
  - Implement a ORM relationship between subnets and IP address
  - Implement nested subnet relationship as well.
  - Test the newly create models and document.

- Week 3 & Week 4 - May 28th, 2018 - 11th June, 2018
  - Create a details page for subnets.
  - Create a mock and final design for automatic free space display for all subnets, using the tech listed above.
  - Create a visual display in tabular form for a specific subnet in the details page.

- Week 5 - 15th June, 2018 - 21st June, 2018
  - Implement **IP request module**.

- Week 6 - 22nd June to 29th June, 2018
  - Finalize on APIs to be created.

    - Create API classes
    - Add filters and paginations to the api
    - Write down the documentation for the API.

- Week 7 - 29th June, 2018 - 6th July, 2018
    - Implement a search module in the app.
    - Implement CSV import-export modules.

PHASE 2 -

- Week 8 - 6th July, 2018 - 13th July, 2018
    - Summarize and present the work for second mid evaluations (9th July - 13th July)
    - Start with openwisp-ipam implementation.
    - Create basic models and other modules that are required.

- Week 9 - 13th July, 2018 - 21st July, 2018
    - Add multitenancy features using openwisp-utils and openwisp-user
    - Implement a test suite for openwisp-ipam and as well as for django-ipam, as we have done previously in django-freeradius and django-netjsonconfig.
    - Write test cases for all the features and ensure coverage of <90%.

- Week 10 & 11 - 21st July, 2018 - 4th August, 2018
    - Wrapping up work and preparation for final submission.
    - Improvements related to UI/UX if needed.
    - Ensure all the new features are documented properly
    - Additionally improve the existing documentation.
    - Clean up code and improve readability, wherever possible.

    - Make sure we achieve a test coverage of 100% in django-ipam and openwisp-ipam.

- Week 11 - August 6th, 2018 - August, 14th 2018
    - Submit work for final evaluation
    - Conclusion of GSOC

Availability

From May to July I can easily devote 40-50 hours per week, as I will be at home and my semester would have ended. Currently I do not have any other obligations and my primary focus would be on the project during that period of time.
Somewhere in mid-end July my college would reopen and I will have to readjust my daily schedule according to my class time-table which will be provided once I reach by the end of July.
Incase of our project most of the work would be over by July end and August start so I don't think college lectures would hinder my work.

After GSOC

Are you interested in working with OpenWISP after the GSOC ends?

- ❏ Yes, I am interested in working with OpenWISP even after GSOC ends. As of now I have been contributing to OpenWISP radius and I really felt that this community is very good and is helping me develop my skills as well. I got to learn a lot in the past few weeks and I would love to continue on the same track.

Will you maintain your implementation for a while?

- ❏ Yes, I can maintain my implementation for a while. I might be a bit inactive during my college exam time and other non-coding activities but overall I feel I'll be able devote my time towards the project even after GSOC.

If we get new business opportunities to build new features, would you be interested in occasional freelance paid work?

- ❏ Yes, I am interested in doing occasional freelance work. In the past I have worked with a few people for freelance paid work and I guess that experience might as well help me over here.