Network Science Research Tool Project

Group Number: 44

GROUP MEMBERS:

Abhishek Bhat, 1211045365
Ashish Sirohi, 1211009485
Gurumurthy Raghurman, 1211150041

Mentor (TA): Arun Das

Site URL: 10.218.105.80:8000

Introduction:

Today there are many kind of disasters like Tsunami, Hurricane, Floods etc. Every disaster may affect the connectivity of networks. Thus, networks built need to be robust to faults. Also as network might be over a vast area it is difficult to test the robustness of the network. In this project, we build a tool that helps us estimate the impact of such disasters(fault).

In this project we consider a wired network as a undirected graph G(V,E) where V is the number of nodes and E is the no of Edges. We define the connectivity of a graph as the number of faults up to which the graph remains connected i.e. the number of connected components in G is one [1]. So, suppose the connectivity of a graph is k, then it can tolerate up to k-1 faults which implies that the graph remains connected even after k-1 faults. These faults can be confined in a region. Also, the faults do not give any information regarding the impact on the graph like the number of connected components into which it disintegrates, size of largest connected component, the locations where the impact of the fault is the highest etc. when the size of the fault is greater that the connectivity of the network.

To understand different types of faults and their impacts we introduce the concept of region based connectivity. In region based connectivity the fault is confined to a region. A region can be defined as a subgraph of diameter d (where diameter is the maximum of the shortest path distance taken between all pairs of nodes in the graph). A region can also be defined as a circle of radius 'r' [1]. We can see that there can be infinite number of circles of radius r in a network. But as stated in paper [2], we can say that considering only a finite number of regions is sufficient to analyze a network. This project consists of tools that build and analyze faults in a network. The report is divided into the following sections: System Architecture and Setup, Topology Manager, Generic Fault Analysis, Specified Fault Analysis, Assumptions, Results, Conclusion, and Future Work.

The project is reimplementation of the paper [1], but on a python Django Framework and PostgreSQL using the PostGIS features available on GeoDjango and PostgreSQL(Spatial Database). The tool in paper [1] uses open street maps API but our tool uses Google Maps API V3. Also, we have built a responsive UI and the application can be used on mobile devices as well.

System Architecture and Configuration

In this section, we will discuss about the underlying system architecture and steps followed to configure the system for our application.

A. System Architecture

Network Planning and Management Tool(NPMT) has been implemented as a web application, which allows users to remotely connect and operate from the web browser. Our web-application follows the standard three tier architecture i.e. client tier, server tier and database tier[1]. We have used Django (a free and open source web framework written in python[6]) which follows model-view-template architecture. NPMT being a GIS(Geographic Information System) application, we have exploited and used GeoDjango, it is an included contrib module for Django, that turns it into a world-class geographic Web framework. GeoDjango strives to make it as simple as possible to create geographic Web applications, like location-based services[7]. Below figure shows the high-level Django MVT architecture.

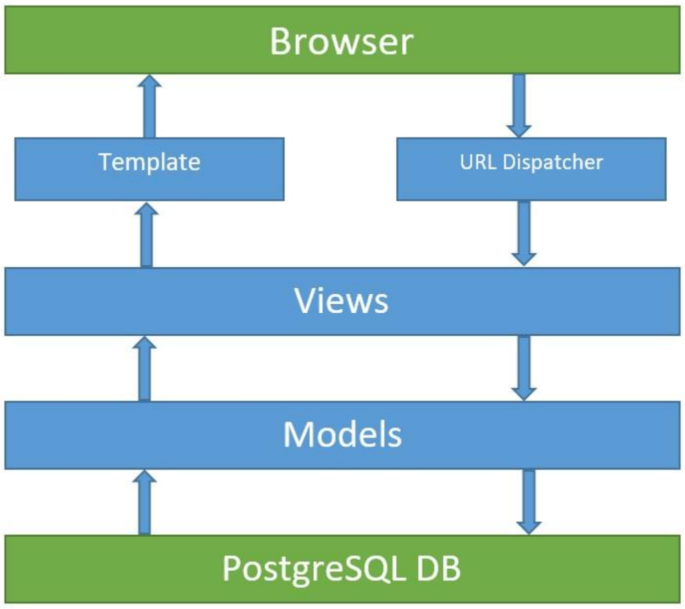


Fig 1 : Underlying MVT Architecture

The NPMT tool is currently deployed and accessible from Arizona State University's WAN, and runs from testbed server. The tool's web application is deployed on an Apache Tomcat 7 instance, and the repository used is PostgreSQL (with PostGIS extension). The application tier business logic for several operations on network topologies like Region base fault tolerance are implemented on python. Our testbed server is 64 bit Intel Core 2 Quad Core (2.66 GHz) system with 2 GB of RAM and 8 GB of disk space running on Ubuntu 16.10 instance.

B. Configuration Steps

I. Installed Python 3.5.2 using the following commands

```
$ sudo apt-get update
$ sudo apt-get install python3
$ apt-cache show python3
$ sudo apt-get install python3==3.5.2
```

II. Installed Django Framework with virtual environment using following commands

```
$ pip install virtualenv
$ virtualenv –python=python3 foaproject
$ source foaproject/bin/activate
$ pip install Django==1.10.2
$ Django-admin --version
```

III. Installing Geospatial libraries

GEOS : GEOS is a C++ library for performing geometric operations, and is the default internal geometry representation used by GeoDjango (it's behind the "lazy" geometries)[8].

Steps :

```
$ wget http://download.osgeo.org/geos/geos-3.4.2.tar.bz2
$ tar xjf geos-3.6.0.tar.bz2
$ cd geos-3.6.0
$ ./configure
$ make
$ sudo make install
$ cd ..
```

PROJ.4 : It is a library for converting geospatial data to different coordinate reference systems[8].

Steps :

```
$ wget http://download.osgeo.org/proj/proj-4.9.1.tar.gz
$ wget http://download.osgeo.org/proj/proj-datumgrid-1.5.tar.gz
$ tar xzf proj-4.9.1.tar.gz
$ cd proj-4.9.1/nad
$ tar xzf ../../proj-datumgrid-1.5.tar.gz
$ cd ..
$ ./configure
$ make
$ sudo make install
$ cd ..
```

GDAL : It is an excellent open source geospatial library that has support for reading most vector and raster spatial data formats[8].

Steps :

```
$ wget http://download.osgeo.org/gdal/1.11.2/gdal-1.11.2.tar.gz
$ tar xzf gdal-1.11.2.tar.gz
$ cd gdal-1.11.2
$ ./configure
$ make
$ sudo make install
$ cd ..
```

IV. Installed PostgreSQL with PostGIS extension

Steps :

```
$ sudo add-apt-repository "deb
http://apt.postgresql.org/pub/repos/apt/xenial-pgdg main"
$ wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc |
$ sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install postgresql-9.6
```

Adding PostGIS Extension to the PostgreSQL DB :

```
$ wget http://download.osgeo.org/postgis/source/postgis-2.1.8.tar.gz
$ tar xfz postgis-2.1.8.tar.gz
$ cd postgis-2.1.8
$ ./configure
$ make
$ sudo make install
$ sudo ldconfig
$ sudo make comments-install
$ sudo ln -sf /usr/share/postgresql-common/pg_wrapper
/usr/local/bin/shp2pgsql
$ sudo ln -sf /usr/share/postgresql-common/pg_wrapper
/usr/local/bin/pgsql2shp
$ sudo ln -sf /usr/share/postgresql-common/pg_wrapper
/usr/local/bin/raster2pgsql
$ psql -h localhost -U cse551foa nsrtp
nsrtp=# CREATE EXTENSION postgis;
nsrtp=# \q
```

System Capabilities

We have designed NPMT to be used for the following tasks :

1. Topology Management (Network Creation)
2. Fault Analysis (Network Analysis)

Each of the above features are accessible from a tabbed interface of the tool and can be navigated to from any part of the web-application[1]. In the following subsections we discuss each of the features and provide a brief functional overview.

Topology Management

Network creation is the foremost step for analyzing or performing any operations on it. Our first tab is the Topology Manager which allows the user to create, edit, save and delete network topologies. Our Topology manager has the following feature :

- Google Map API has been used for topology creation
- Used Mercator projection for efficient distance calculation for analysis in later stages
- Used PostGIS spatial database for efficiently storing and retrieving network data
- Duplicate markers, polylines case has been handled
- Markers and polylines can be deleted and updated accordingly, using map interaction UI

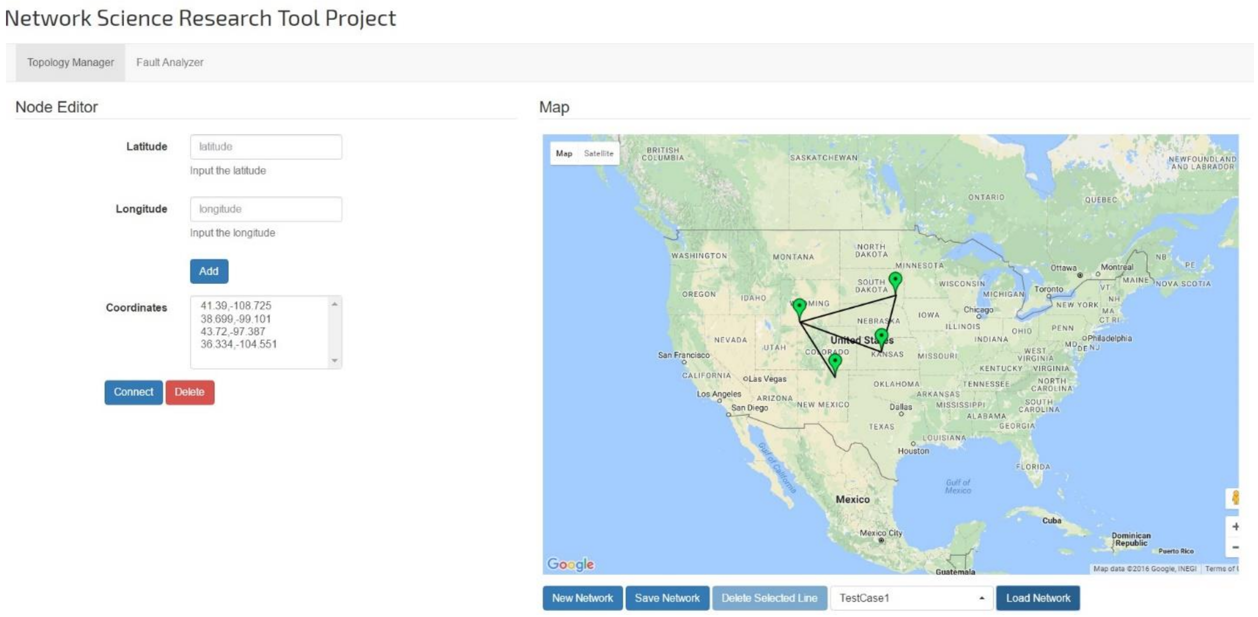
To create the network topology, user must place markers on the map by entering the marker's latitude and longitude in the provided fields and then click add which will create the marker on the map. Using the same step, user can add as many markers as needed for the network. To connect the required markers, user can either click on the two markers which needs to be connected and click connect or user can select upto 2 markers from the stored markers' list and click connect to create polyline between them. By repeating the above process user can create the required network topology. This tool has given user the capability to delete any of the marker and associated polylines if needed. This can be done by clicking on the marker and then click the delete button. Similarly, polylines can be deleted by selecting the desired polyline and click "Delete Selected Polyline".

To save the above created network, user can click save button which will then ask the user to input the graph name, and it will be saved to the database with the given graph name. A message is returned to the user confirming the successful saving of the graph to the database.

To load the previously saved graphs, user can select the desired graph from the provided dropdown list and click on the load graph button. This will load the already saved graph. User can edit and save it as a new graph again by clicking the save button.

New graph button will refresh the map and old data provided by the user will be cleared and will give a fresh map to build the new network topology. Clear map button will clear the map along with data.

Fig. 2 shows the network topology with the above discussed features.


Network Science Research Tool Project

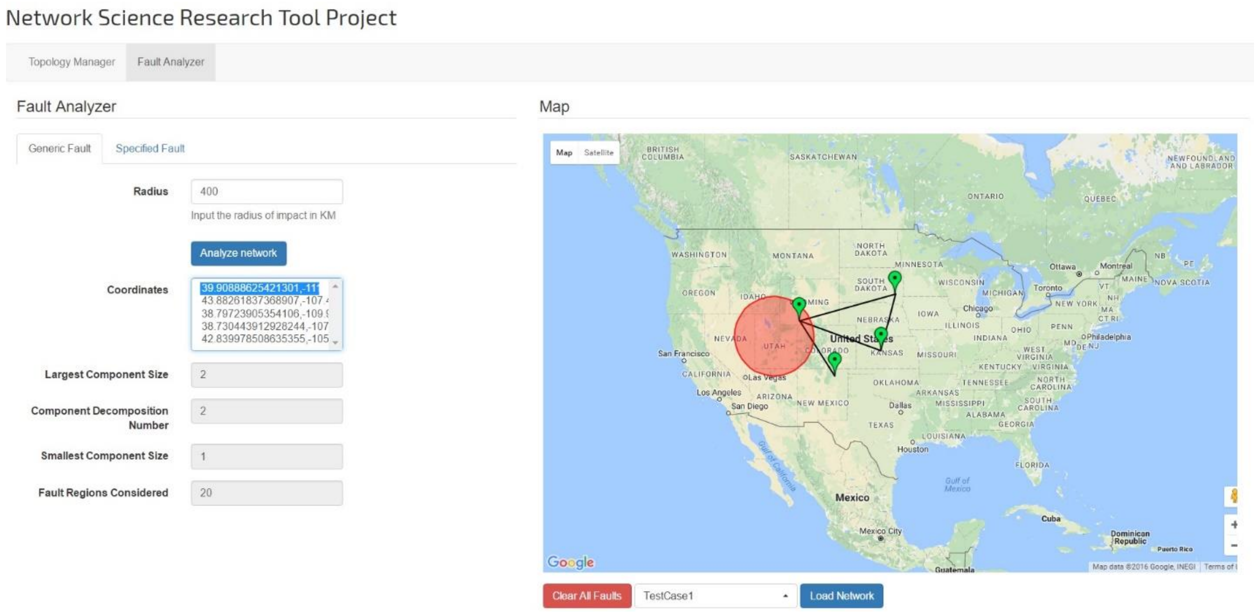
Node Editor

Latitude: latitude — Input the latitude

Longitude: longitude — Input the longitude

Coordinates:
41.39,-108.725
38.699,-99.101
43.72,-97.387
36.334,-104.551

Fig 2 : Network Topology Manager


Network Science Research Tool Project

Fault Analyzer

Generic Fault | Specified Fault

Radius: 400 — Input the radius of impact in KM

Coordinates:
39.90888625421301,-11
43.88261837368907,-107.
38.79723905354106,-109.
38.730443912928244,-107
42.839978508635355,-105

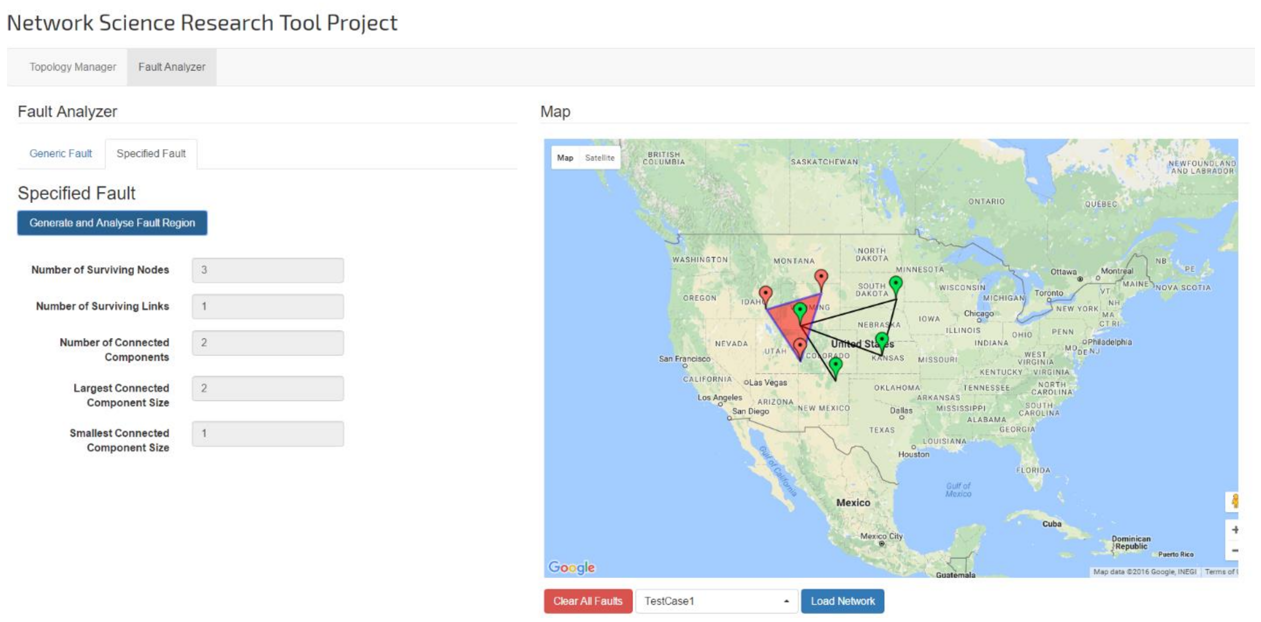
Largest Component Size: 2

Component Decomposition Number: 2

Smallest Component Size: 1

Fault Regions Considered: 20

Fig 3: Generic Fault Analyzer


Network Science Research Tool Project

Topology Manager | Fault Analyzer

Fault Analyzer

Generic Fault | Specified Fault

Specified Fault

Generate and Analyse Fault Region

Number of Surviving Nodes	3
Number of Surviving Links	1
Number of Connected Components	2
Largest Connected Component Size	2
Smallest Connected Component Size	1

Map

Clear All Faults | TestCase1 | Load Network

Fig 4: Specific Fault Analyzer

Generic Fault:

In this section, we give an overview of the Generic Fault Analysis. In generic fault a region of radius r is given. As we know there can be infinite number of regions in a graph (If we move the center of the circle by a minute distance we get another circle).

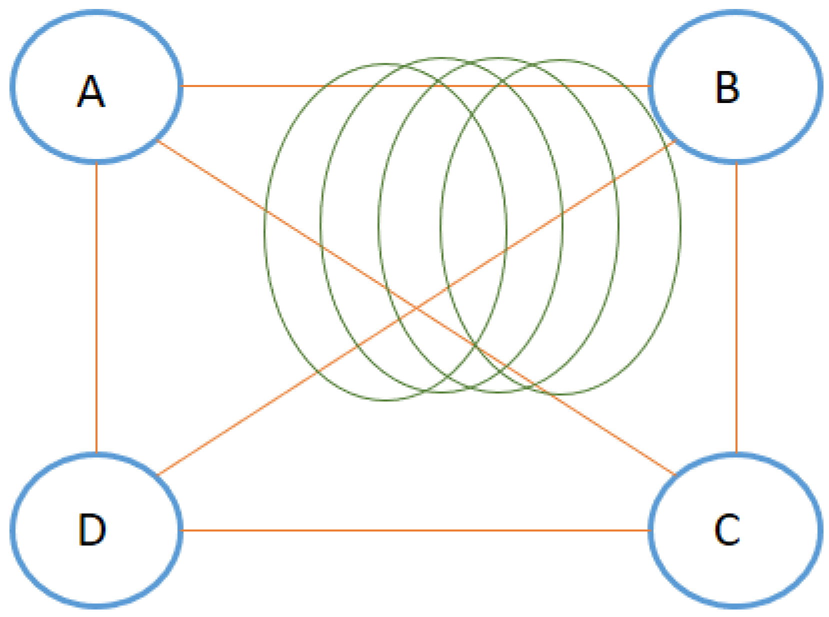
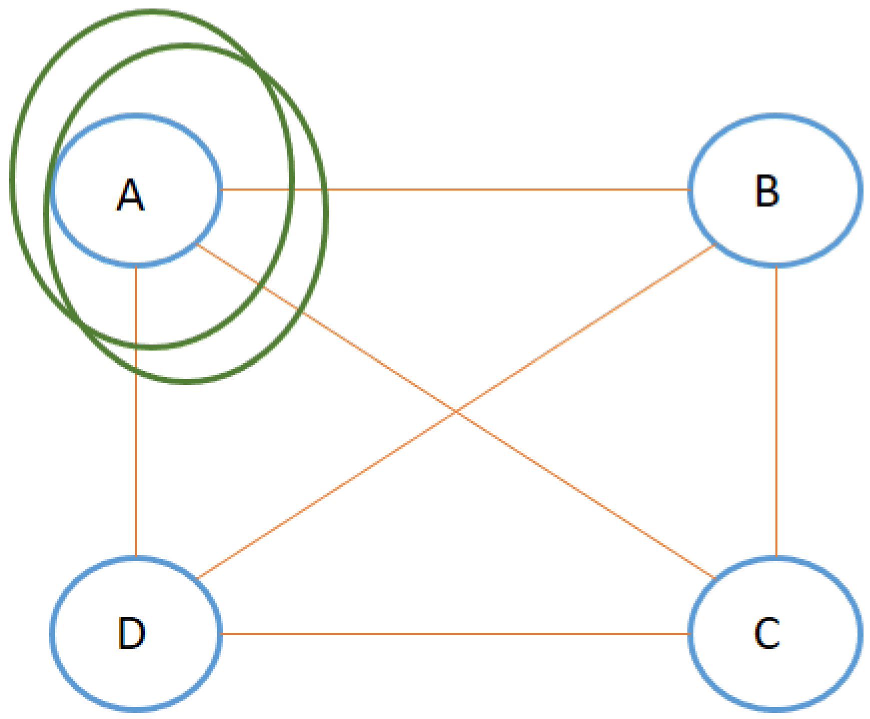


Fig 5: Showing there can be infinite number of Regions in a graph.

From paper [2], we can say that we do not need to compute all the infinitely small regions, but only the principal regions. Suppose there is a region which intersects edge AD, AB and AC and completely covers node A of as shown in Fig 6



.

Fig 6: A sample graph showing 2 regions intersecting the same set of nodes and edges.

Now as we can see in figure 6, both the regions cover/intersect the same set of nodes and edges. So, the effect of the fault in both the cases remain the same. Thus, it is sufficient to examine only one of the regions. Thus, a graph has a set of finite regions which cover/intersect a unique set of nodes and edges. These finite set of regions covering a unique set of nodes and edges is called the principal regions[2].

Thus, we can compute the effect of a region based fault by computing the effect of the faults at the principal regions. As mentioned in paper [2], we can say that there are a finite set of principal regions and the set of principal regions can be computed in time O(n^4) (Polynomial Time Algorithm).

Computation of Distinct regions:

For computing the distinct regions in the graph we introduce the concept of a NVZ(Node Vulnerability Zone) and LVZ(Link Vulnerability Zone)[3]. Consider a graph with 2 nodes and 1 edges as shown in Figure 7.

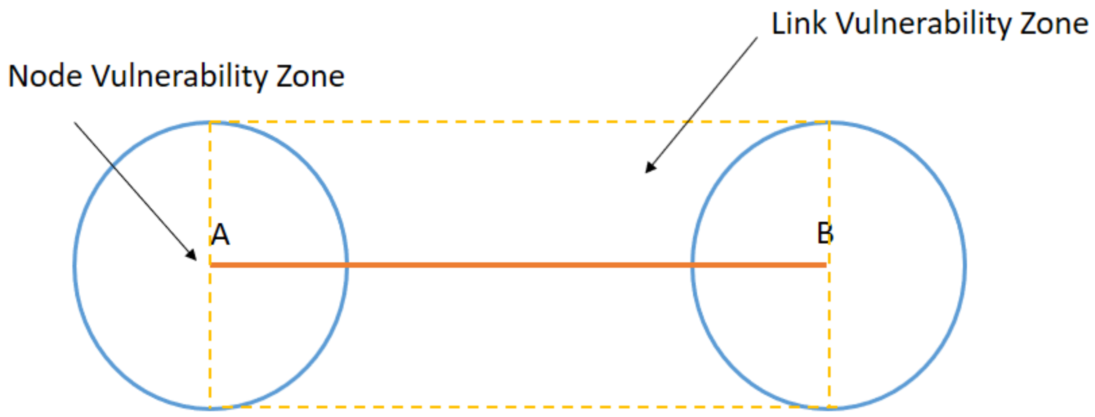


Figure 7: Link Vulnerability and Node Vulnerability Zone of a graph

As shown in figure 7 the region confined to the dotted line is the link Vulnerability zone. If a fault occurs in that zone, then the edge A to B will fail. The circle around the nodes A and B is called the Node Vulnerability Zone(NVZ). If a fault in the NVZ then nodes A or B will fail. We use the concept of node vulnerability zone and Link Vulnerability zone to define the concept of I points.

<u>I -Point:</u>

An I Point can be defined as the intersection of a link vulnerability zone and the node vulnerability zone. These intersection points are called the I Points[3]. We can take these I Points as the centers and draw the circles. The I Points are highlighted for a 4 node graph as follows:

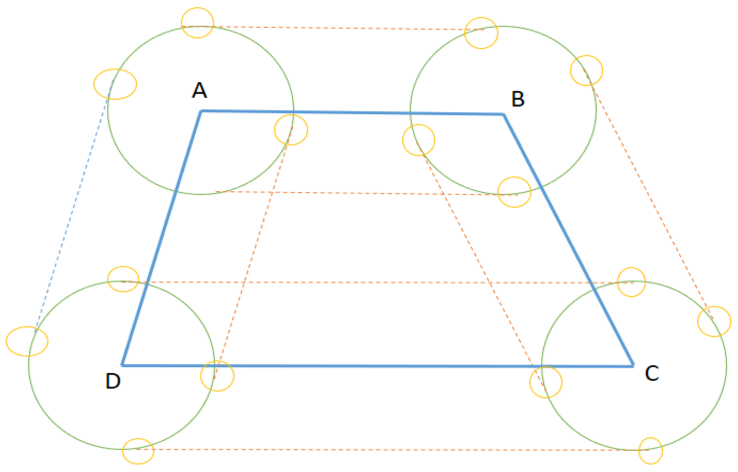


Figure 8: Showing the I Points of a Graph with 4 nodes and 4 edges.

The number of I-Points is of the power of O(n^6) where n is the number of nodes [1].

We now go on to define the terms Region Based Component Decomposition Number(RBCDN), Region Based Largest Component Size(RBLCS) and Region Based Shortest Component Size(RBSCS)[1][3].

<u>Region Based Decomposition Number:</u>

As we pointed out earlier, we need metrics to compute the effect of the fault. For every region let us remove the intersecting nodes and edges of the graph. Let us call this graph as G'. Now for G', compute the number of connected components. For all distinct regions, compute G' and number of connected components. Add the number of connected components to a list CN. Once this is done for all the regions the max of CN is the RBCDN of the network[3]. Mathematically it can be defined as follows:

RBCDN of Graph G = max CN[i] where 1<=i<=k

Where k is the number of distinct regions.

<u>Region Based Largest Component Size:</u>

For computing the Region Based Largest Decomposition Number, we identify the minimum size of largest connected component in the graph. So, for k distinct regions we will have k different values of the largest connected component. The smallest of these values is the Region Based Largest Component Size[3].

RBLCS of Graph G=min LCS[i] where 1<=i<=k

<u>Region Based Smallest Component Size:</u>

For computing the Region Based smallest component size we compute the smallest component size for each of the k distinct regions. Let us assume that this list is name SCS. The minimum value in the list SCS is defined as the Region Based Smallest Component Size[3].

RBSCS of a Graph G = min SCS[i] where 1<=i<=k

<u>Pseudo Code:</u>

1. Input: Radius r in Kilometers
2. Output: Region Based Decomposition Number, Region Based Largest Component Size and Region Bases Shortest Component Size.
3. Define the Node Vulnerability and the Link Vulnerability Zones.
4. Find the points of intersection of the Link Vulnerability Zone and the node vulnerability zone (these are the set of I points).
5. From each I Point draw a circle of radius r.

6. Check if the nodes and edges intersected by the I point has already been intersected by another I Point.
7. If result of step 6 is no, remove the intersecting nodes and edges from the graph.
8. Using DFS Compute the RBCDN, RBSCS and RBLCS of the graph.
9. Return the maximum of RBCDN, Minimum of RBSCS and Minimum of RBLCS.

Specified Fault:

In specified fault the user inputs the co-ordinates of the fault and then the fault metrics are calculated. After the points of the fault is given we draw a convex hull around it as shown in the figure below.

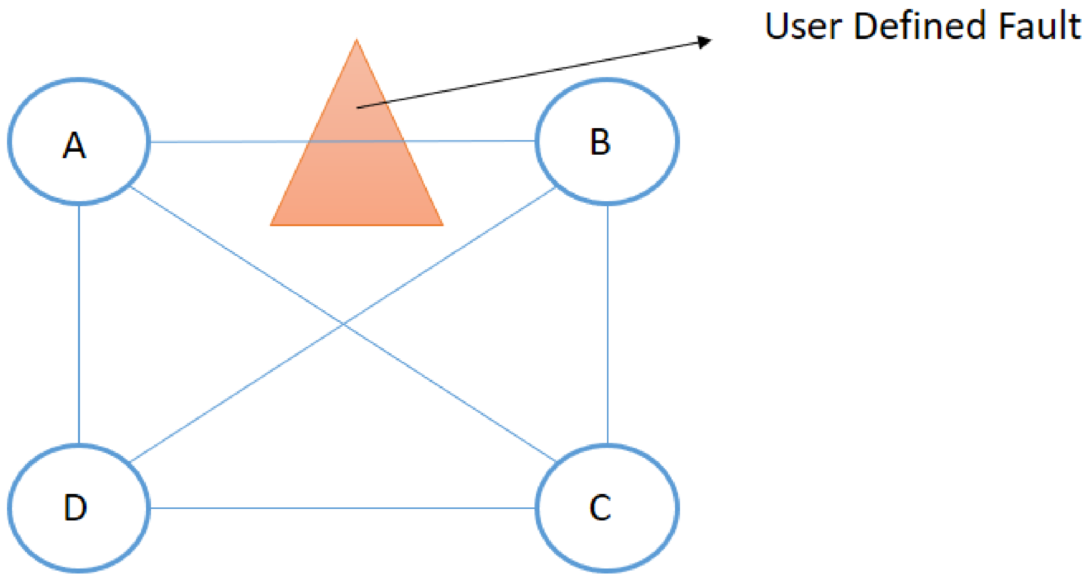


Fig 9 : Showing User Defined Fault Region.

In our implementation, we can extend the specified fault by adding new fault markers and analyzing the network again.

Algorithm for Computing RBCDN, RBLCS, RBSCS for a specified Fault:

1. Input: User Defines Fault Co-ordinates.
2. Output: RBCDN, RBLCS and RBSCS of the graph.
3. From the user, defined co-ordinates define the polygon.
4. Check whether a node is within the fault region. If yes remove it and its corresponding edges from the graph.
5. Now check whether an edge intersects with the fault region, if yes remove it from the graph.

6. Using DFS compute RBCDN, RBSCS and RBLCS of this remaining graph and output it along with the fault region to the user.

The DFS does take a computation time of O(n^2) and as mentioned earlier the computation of principal regions is of the order of O(n^6) and thus the time complexity of the algorithm above is O(n^8).

User Interface:

The user interface for the tool is developed using the Bootstrap libraries. The tool has a responsive design working seamlessly on both desktop and mobile devices.

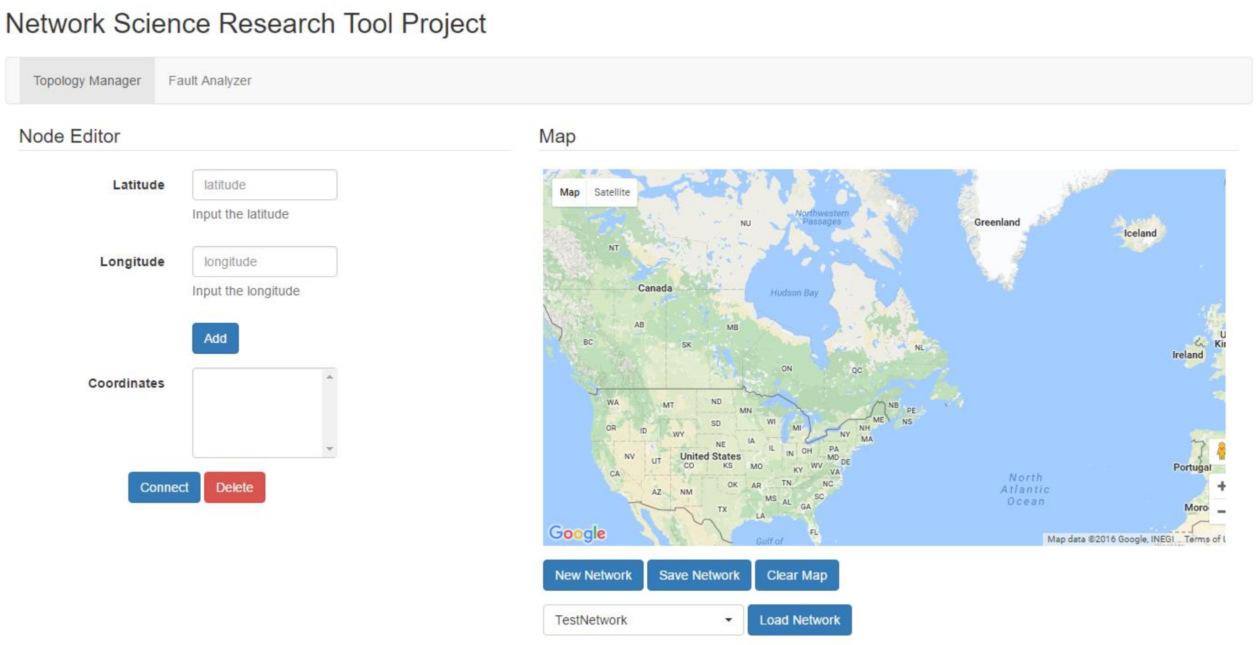


Fig 10: Desktop View

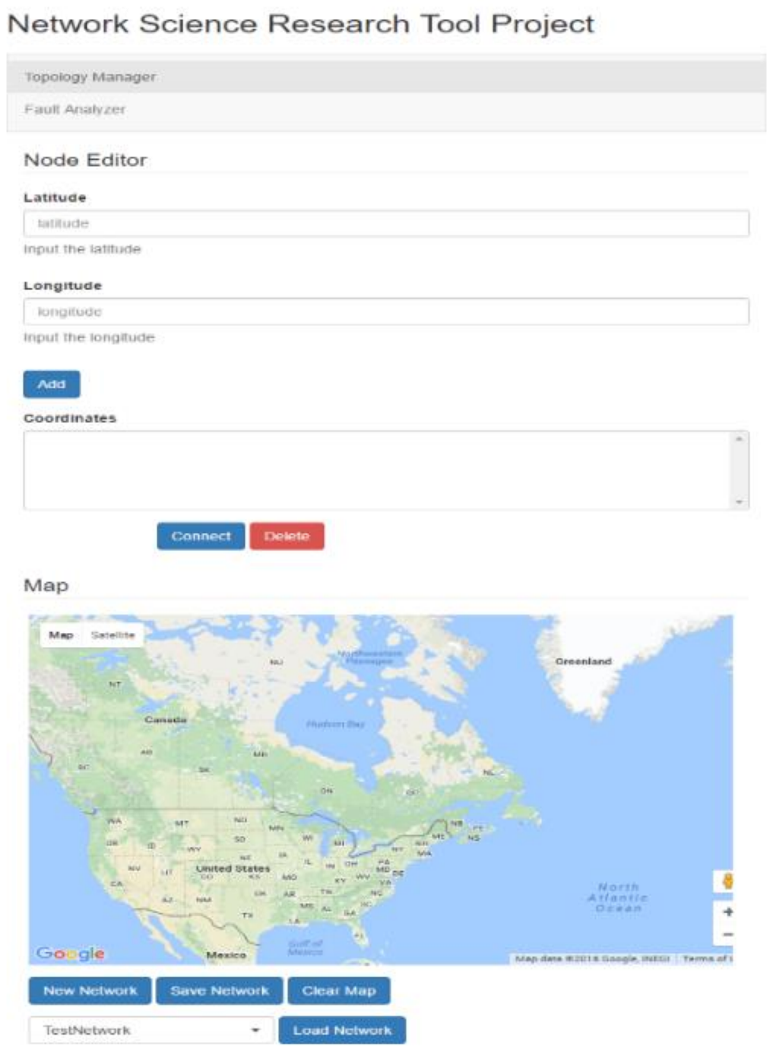


Fig 11: Mobile View

Assumptions:

1. GeoDjango replicates a circle as a closed polygon. This does not give the exact intersection points with the tangent of a circle.(As a polygon is smaller in size). So we add a pseudo distance of 0.0001 to the radius of the circle). Also geoDjango gives an intersection region(polygon) for the intersection of 2 circles. Two of the points on the polygon are the intersection points. The intersection points are at slightly greater than r from two circles. Because of these issues our answers may differ on certain test cases taken from the existing tool.
2. Internet Connection is a necessary to load google maps and bootstrap libraries.
3. For specified faults, map needs to be loaded first and then fault co-ordinates needs to be defined.

Conclusion and Future Work:

In this project, we have built a tool that allows to create a wired network and analyze the effects of region based faults. There is already an existing version of this tool as mentioned in [1], but this tool is built on a python Django Framework connected to PostGreSQL with PostGis Database. This will allow us specified faults as a geometric collection of points. As python is a scripting language the amount of effort needed to build new features on top of this is also less. With the help of Geo Django Library geometric computation may be easier in a lot of cases. Also, we have used the Google Maps API to build the network. Google Maps API also gives a lot of inbuilt features to draw circle, polygons. The geometry library of google maps can be used to compute and show results on the UI front. UI has a responsive design with can be accessed even from mobile devices. As a part of future work we can work on building the already existing Path Analyzer module in paper [1] in the python Django Framework. All in all, though this tool was a reimplementation of the existing tool in paper[1], it is built in a new framework which makes it more scalable in the near future.

Bibliography

1. Arun Das, Arunabha Sen, Chumming Qiao, Nasir Ghani, Nathalie Mitton "A Network Planning and Management Tool for Mitigating the Impact of Spatially Correlated Failures in Infrastructure Networks" 12th International Conference on the Design of Reliable Communication Networks (DRCN). IEEE 15-17 March, 2016.
2. A. Sen, B. H. Shen, L. Zhou, and B. Hao, "Fault-tolerance in Sensor Networks: A New Evaluation Metric," in Proceedings of IEEE Infocom, Barcelona, Spain, April 2006, pp. 1–12.
3. S. Banerjee, S. Shirazipourazad, and A. Sen, "Design and analysis of networks with large components in presence of region-based faults," in International Conference on Communications (ICC). IEEE, 2011.
4. S. Banerjee, S. Shirazipourazad, P. Ghosh, and A. Sen, "Beyond connectivity-new metrics to evaluate robustness of networks," in High Performance Switching and Routing (HPSR), 2011 IEEE 12th International Conference on. IEEE, 2011, pp. 171–177.
5. http://www.inf.ed.ac.uk/teaching/courses/inf2b/algnotes/note10.pdf Add Connected Components Reference.
6. https://en.wikipedia.org/wiki/Django_(web_framework)
7. https://docs.djangoproject.com/en/1.10/ref/contrib/gis/tutorial/#introduction
8. https://docs.djangoproject.com/en/1.10/ref/contrib/gis/install/