ECDNA-2001: "Ethinic Conflict Dataset Network Analysis-2001"

"The mind is everything. What you think, you become....(*Buddha*)"

Ravindra

Date : 03-19-2021

Abstract

There are many news reports in media contain records of a wide range of socio-economic and political events. Using Ethinic Conflict-2001 dataset of news records, we study the network of ethnic conficts. Network analysis of this dataset we get important information about the related actors, that are pointing towards their long range efect over the world. We find power law decays in distributions of actor mentions, co-actor mentions and degrees and dominance of infuential actors and roups

Introduction

An ethnic conflict is a conflict between two or more contending ethnic groups. While the source of the conflict may be political, social, economic or religious, the individuals in conflict must expressly fight for their ethnic group's position within society. This final criterion differentiates ethnic conflict from other forms of struggle.The causes of ethnic conflict are debated by political scientists and sociologists.
The Ethnic Conflict -2001 dataset have information regarding actors and groups, their geographical location and the event event time. Here we will generate the network for the given dataset and analyse it to get the most active actor or group and try to find out the cause behind it. Here we are using Girvan-Newman algorithm to find out the communities and power law decays in degree distributions of actor mentions, co-actor mentions and degrees and dominance of infuential actors and groups. Most infuential actors or groups form a giant connected component, and is expected to encompass all actors globally in the long run.

Method

Before Constructing the network we perfrom some preliminary analysis on given dataset by plotting bar graphs for each column in dataset. after analysing all bargraphs we find that there are so many null values present in the dataset which can effect the network analysis , first we removed the null values and clean the data. Now data is ready, to construct the network we are using "Actor1Name" as "Source" and "Actor2Name" as "Target". And also using different color for Source (Blue) and for Target (Green) so that source and Target can be identified easily. And the size of the node is depened upon the degree of it. higher the degree larger the size. After creating Network graph we perform following task to analyse the network:

- Plot the degree distribution of the network and fit the line using power law and noting down the exponent and slope of the line.
- After this we find total number of organizations that are involved in ethnic conflict activities and also the number of attacks with some other statistic measure like number of nodes, edges, average in degree, average out degree, density or graph and also average node connectivity.
- After that we calculate the degree centrality, closeness centrality, and normalized betweenness centrality by plotting the bargraph and also find the same for node 15.
- After that we find the number of subgroups in the network that are 10.
- Along with the subgroups we also find out the number of different communities present in the graph and dawing them with having different node colors so that we can easily find out them. Also to find out the communities we use the Girvan Newman algorithm.
- After that we find out the strongly connected component, weakly connected component, average shortest path length, diameter,transitivity and average clustering coefficient
- For doing all the above task we use Python's jupyter notebook, Networkx, seaborn, numpy, library, matplotlib, pandas, math, community, collections etc. libraries and packages.

Data

The below "Ethnic Conflicts-2001" event records are stored in an expanded version of the dyadic CAMEO ('The Conflict and Mediation Events Observations' framework is a new event data coding scheme optimized for the study of third-party mediation in international disputes) format,capturing two actors and the action performed by Actor1 upon Actor2. And a unique array of georeferencing fields offer estimated landmark-centroid-level geographic positioning of both actors and the location of the action.

›r1Code	Actor1Name	Actor1Geo_ADM1Code	Actor2Code	Actor2Name	Actor2Geo_ADM1Code	ActionGeo_FullName	ActionGeo_ADM1Code	Year	SQLDATE
ALB	ALBANIAN	AL	MKD	MACEDONIAN	AL	Albania	AL	2001	20010301
CVL	POPULATION	AL	ALB	ALBANIAN	AL	Belgrade, Serbia (general),	RB00	2001	20010304
CAF	BANGUI	CT00	GOV	GOVERNMENT	CT18	Yakoma, , Central African Republic	CT00	2001	20010605
GOV	MINIST	AL	ALB	ALBANIAN	MK00	Skopje, Macedonia (general), Macedonia	MK00	2001	20010828
ARM	ARMENIA	AM	AZE	AZERBAIJAN	AJ00	Armenia	AM	2001	20011114

Description of each columns present in the dataset:

Actor1Code

The complete raw CAMEO code for Actor1 (includes geographic, class, ethnic, religious, and type classes). May be blank if the system was unable to dentify an Actor1.

Actor1Name

The actual name of the Actor 1. In the case of a political leader or organization, this will be the leader's formal name, for a geographic match it will be either the country or capital/major city name, and for ethnic, religious, and type matches it will reflect the root match class (KURD, CATHOLIC, POLICE OFFICER, etc).

Actor1Geo_ADM1Code

This is the 2-character FIPS10-4 country code followed by the 2-character FIPS10-4(Federal Information Processing Standard 10-4) administrative division 1 (ADM1) code for the administrative division housing the landmark.In the case of the United States, this is the 2-character shortform of the state's name.

Actor2Code

The complete raw CAMEO code for Actor2 (includes geographic, class, ethnic, religious, and type classes). May be blank if the system was unable to dentify an Actor2.

Actor2Name

The actual name of the Actor 2. In the case of a political leader or organization, this will be the leader's formal name, for a geographic match it will be either the country or capital/major city name, and for ethnic, religious, and type matches it will reflect the root match class (KURD, CATHOLIC, POLICE OFFICER, etc).

Actor2Geo_ADM1Code

This is the 2-character FIPS10-4 country code followed by the 2-character FIPS10-4 adaministrative division 1 (ADM1) code for the administrative division housing the landmark.In the case of the United States, this is the 2-character shortform of the state's name.

ActionGeo_FullName

This is the full human-readable name of the matched location. In the case of a country it is simply the country name. For US and World states it is in the format of State, Country Name", while for all other matches it is in the format of "City/Landmark, State, Country". This can be used to label locations when placing events on a map.

ActionGeo_ADM1Code

This is the 2-character FIPS10-4 country code followed by the 2-character FIPS10-4 administrative division 1 (ADM1) code for the administrative division housing the landmark. In the case of the United States, this is the 2-character shortform of the state's name.

Year

Alternative formatting of the event date, in YYYY format.

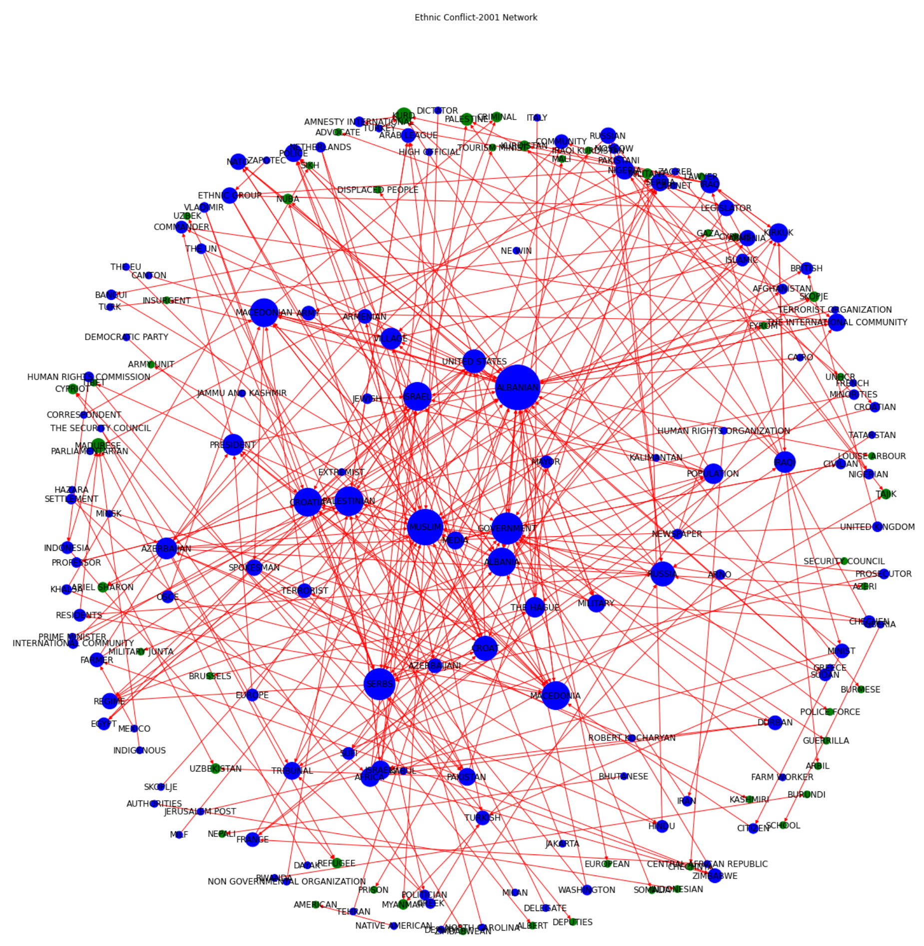
SQLDATE

Date the event took place in YYYYMMDD format.

Analysis and Result Interpretation

Network Graph Construction

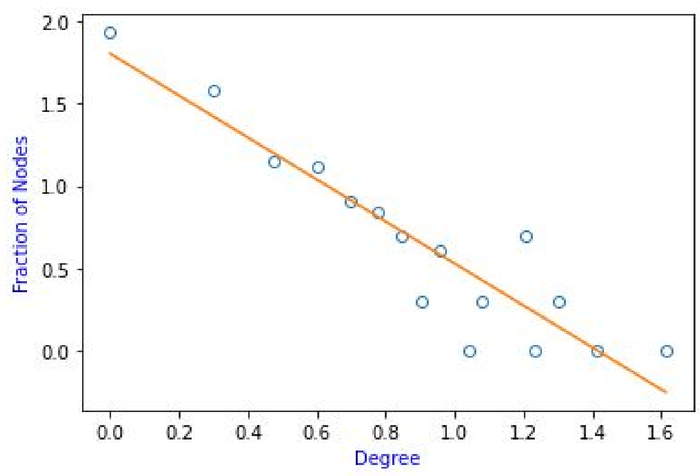
For graph construction we use "Actor1Name" as "Source" and "Actor2Name" as "Target". And also using different color for Source (Blue) and for Target (Green) so that source and Target can be identified easily. And the size of the node is depened upon the degree of it. higher the degree larger the size. By analysing the above network we can say that among the all node some having large size (having high degree) have more ethnic conflicts as compare to others for example ALBANIAN, GOVERNMENT, UNITED STATE are having large conflicts



The Ethnic Conflict-2001 Network

Degree Distribution

Using python's packages we plotted the following degree distribution with Degree as x-axis and Fraction of Nodes (cumulative value) as y-axis. This degree distribution is plotted over the logarithmic scale.




```
Values of exponents are Below

[1.93449845 1.5797836  1.14612804 1.11394335 0.90308999 0.84509804
 0.69897    0.30103    0.60205999 0.         0.30103    0.69897
 0.         0.30103    0.         0.        ]

slope of line :  -1.2789479459930169
```

After plotting the degree distribution we also fit a line on the basis of Power law. this line a have of -1.2789479459930169 and also the exponents (values) are shown below the figure of degree distribution. The Degree distribution is the probability distribution of the degrees over the whole network.

Centrality Measures

For getting Centrality measures, we draw the bar graph for each node with Degree Centrality, Closeness Centrality and Betweenness Centrality by analysing these parameters it is found that the whole graph is not connected and it contain sum subgroups and communities. We noted down the value of Degree Centrality, Closeness Centrality and Betweenness Centrality for the 15th node in the graph which is below:

Degree of Centrality of Node 15 (BHUTANESE) is 0.005291005291005291

Closeness Centrality of Node 15 (BHUTANESE) is 0.0

Betweenness Centrality of Node 15 (BHUTANESE) is 0.0

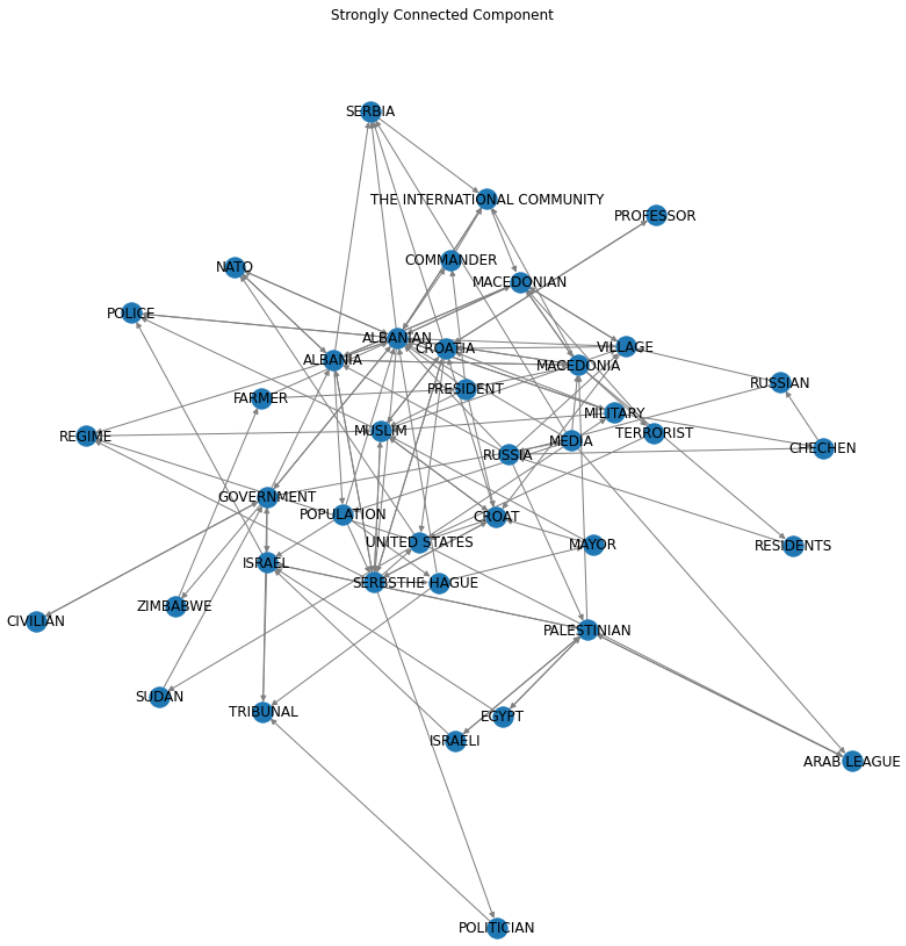
From the bar graph plot we can easily indicate that ALBANIAN have hightest value of all three parameters because it involves in more ethnic conflict events

Subgroups and Communities

As the graph is not connected so it has some subgroups and communities. the total number of groups that we have in the whole network is 10 and also we have plotted separately it inside the code below sum subgroups have large number of nodes and edges while more number of subgroups have less number of nodes and groups it means the subgroup having large number of nodes are highly involved in ethnic conflict while subgroups having less number of nodes not so much reponsible to participate in more number of ethnic conflict events. There are communities too present here and we draw each community with differnt color nodes so that each community can be identified easily . we use Girvan-Newman algorithm to identified the communities here.

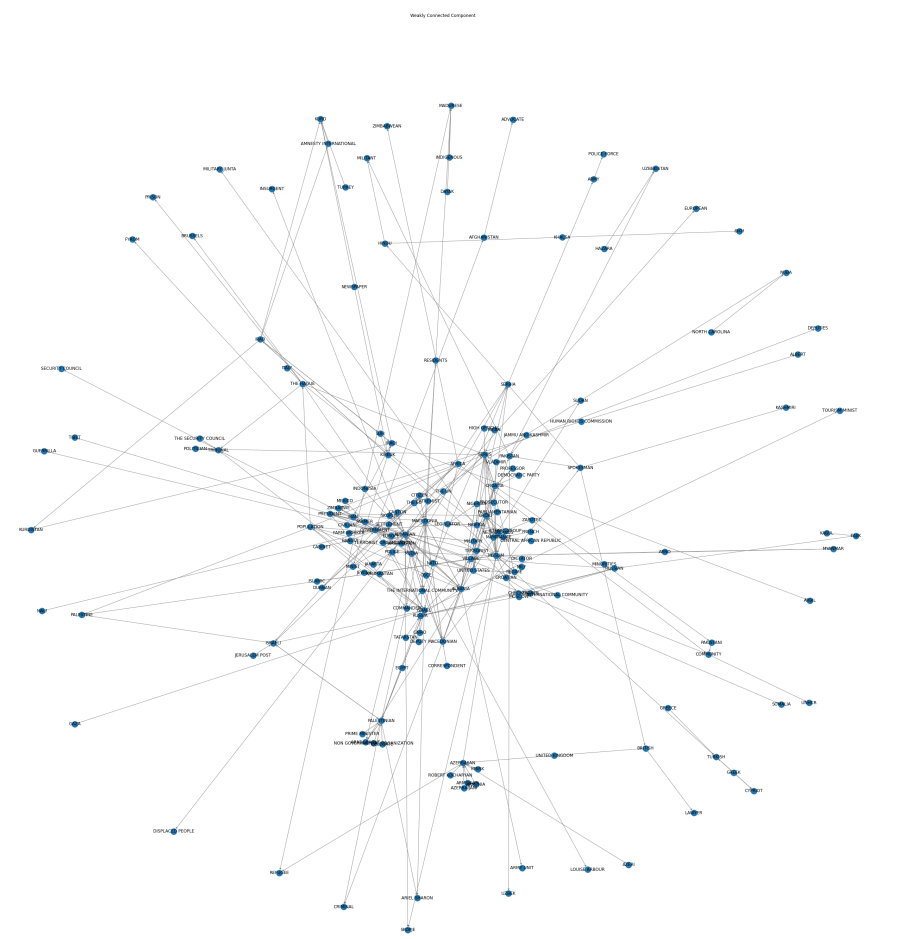
Strongly Connected Components

The total number of strongly connected Components in the whole graph are 145. and we have plotted the largest strongly connected component of the whole network that have 40 nodes.



Weakly Connected Components

The total number of weakly connected Components in the whole graph are 10. and we have plotted the Largest weakly connected component of the whole network that have 172 nodes.



Some other measures

Average Shortest Path for longest weakly connected graph is 1.1651026791785666

Diameter of largest Strongly connected graph is 6

Transitivity of Original graph is 0.13418803418803418

Average Clustering of Original graph is 0.11533424068646471

Critical summary

Before creating this report i was not so much aware about how to plot the network graphs and how to use networkx in python and now after creating this report now i can say that i have so much knowledge to draw network graphs easily. i had programming knowlege before but did not use it for that much to visualize the data and create a network graph and then compare the nodes on the basis of degrees, size and colors very easily and now i have a knowlege to draw degree distribution of a network graph by using power law and using Girvan-Newman algorithm to find out the different communities present in a graph and also finding out the subgroups existed inside the whole graph, now i can find out other parameters related to directed or undirected graphs like Average Shortest Path, Diameter, Transitivity, Average Clustering, order, Strongly Connected Components, Weakly Connected Components, Degree Centrality, Closeness Centrality and Betweenness Centrality of a graph Network.

The assignment is very good and knowledgeable for me, it helps me to improve my knowledge and skills now i feel better then before in understanding the network graphs and their programming work. i think assignment is well design and connected sequencially. Thank you so much sir for providing us this type of assignment.

Reference

- A complex network analysis of ethnic conflicts and human rights violations
- Distress propagation on production networks: Coarse-graining and modularity of linkages
- Networkx Documentation

In [1]:
```
# Importing required library packages
import networkx as nx
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from numpy import random
import pandas as pd
import math
import community
import collections
from random import randint
from colour import Color
```

Method

Uploading Data From "2001-EC.csv" file and creating Dataframe

In [2]:
```
df = pd.read_csv("2001-EC.csv") #Uploading the csv file having data related to Ethnic Conflicts-2001
df.head(5) # Printing first five rows of the dataset
```

Out[2]:

	Actor1Code	Actor1Name	Actor1Geo_ADM1Code	Actor2Code	Actor2Name	Actor2Geo_ADM1Code	ActionGeo_FullName	ActionGeo_ADM1Code	Yea
0	ALB	ALBANIAN	AL	MKD	MACEDONIAN	AL	Albania	AL	200
1	RWA	RWANDA	RW	NaN	NaN	NaN	Rwanda	RW	200
2	MIL	PARAMILITARY	US	NaN	NaN	NaN	United States	US	200
3	CVL	VILLAGE	AL46	NaN	NaN	NaN	Varvare, Qarku Korce, Albania	AL46	200
4	CVL	POPULATION	AL	ALB	ALBANIAN	AL	Belgrade, Serbia (general),	RB00	200

Some preliminary analysis on the data

In [3]:
```
df.columns # Displaying all columns used in the dataframe
```

Out[3]:
```
Index(['Actor1Code', 'Actor1Name', 'Actor1Geo_ADM1Code', 'Actor2Code',
       'Actor2Name', 'Actor2Geo_ADM1Code', 'ActionGeo_FullName',
       'ActionGeo_ADM1Code', 'Year', 'SQLDATE'],
      dtype='object')
```

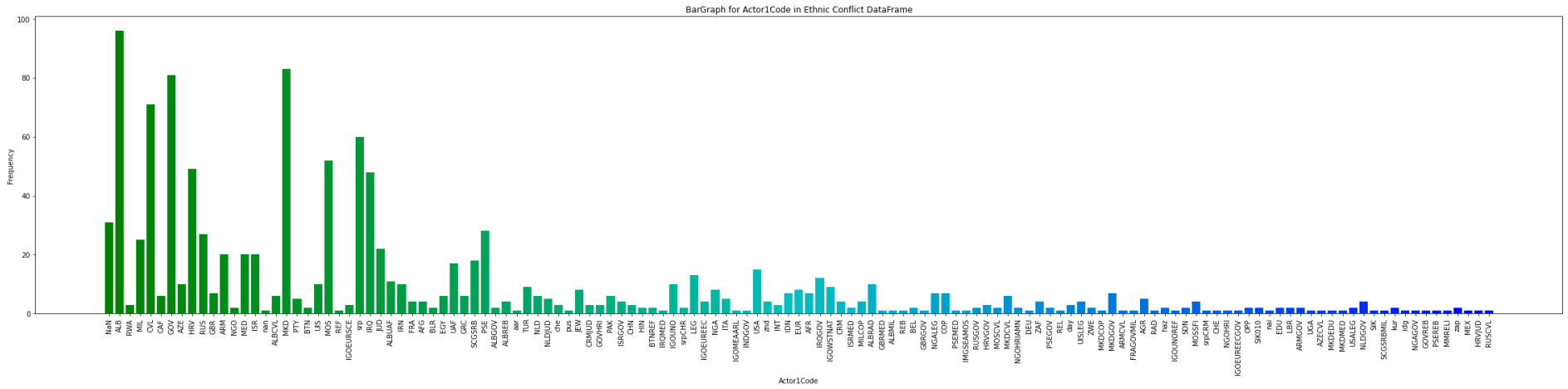
In [4]:
```
def cm_to_inch(value): # method to convert centimeter to inch.
    return value/2.54
```

Dataset Description: Knowing the data first

The dataset is in the form of a .CSV file. This "Ethnic Conflicts 2001" event records are stored in an expanded version of the dyadic CAMEO ('The Conflict and Mediation Events Observations' framework is a new event data coding scheme optimized for the study of third-party mediation in international disputes) format,capturing two actors and the action performed by Actor1 upon Actor2. And a unique array of georeferencing fields offer estimated landmark-centroid-level geographic positioning of both actors and the location of the action.

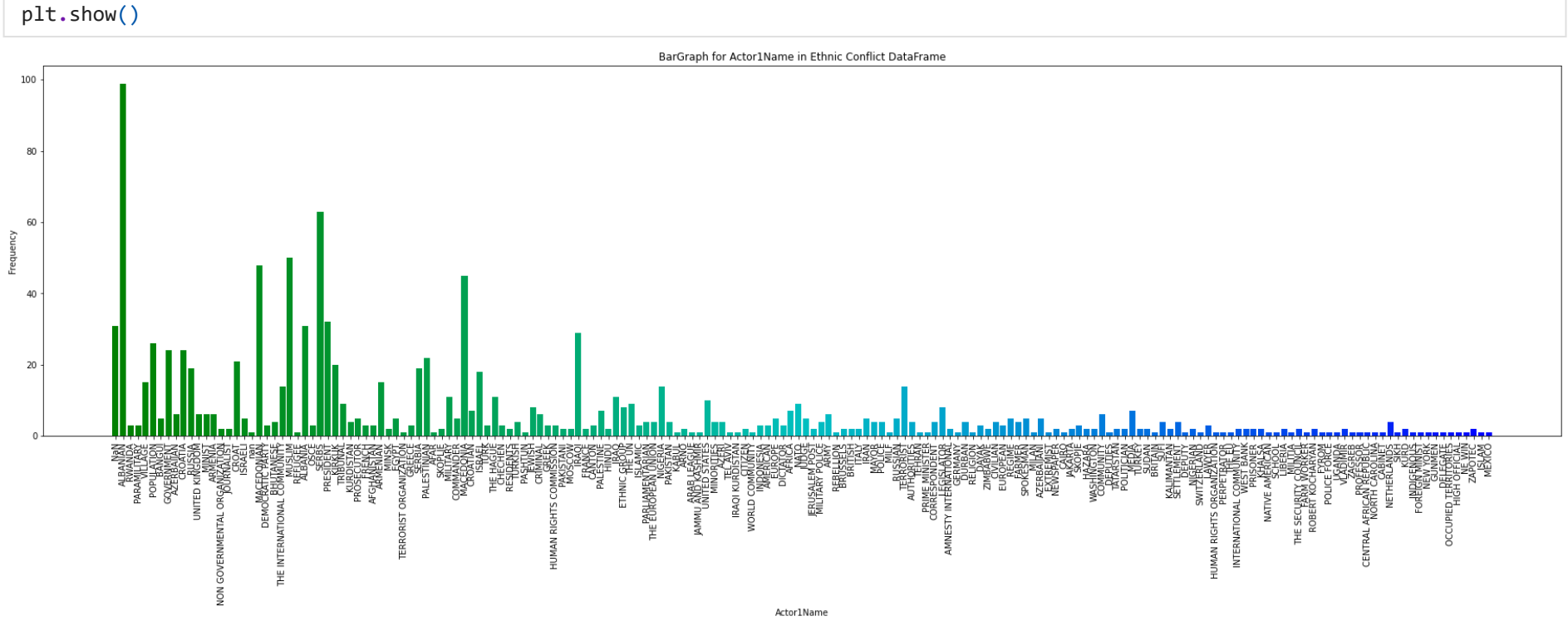
Plotting Bar Graph for "Actor1Code" attribute.

In [5]:
```
dctac1 = {}
for i in df['Actor1Code']:
    if "NaN" not in dctac1:
        dctac1["NaN"] = df['Actor1Code'].isnull().sum()
    if i in dctac1.keys():
        dctac1[str(i)] = dctac1[i] +1
    else:
        dctac1[str(i)] = 1
green = Color("green")
colors_dctac1 = list(green.range_to(Color("blue"),len(dctac1.keys())))
colors_dctac1 = [color.rgb for color in colors_dctac1]

plt.figure(figsize=(cm_to_inch(80),cm_to_inch(20)))
plt.xlabel('Actor1Code')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for Actor1Code in Ethnic Conflict DataFrame")
plt.bar(dctac1.keys(),dctac1.values(),color=colors_dctac1)
plt.tight_layout()
plt.show()
```



Plotting Bar Graph for "Actor1Name" attribute.

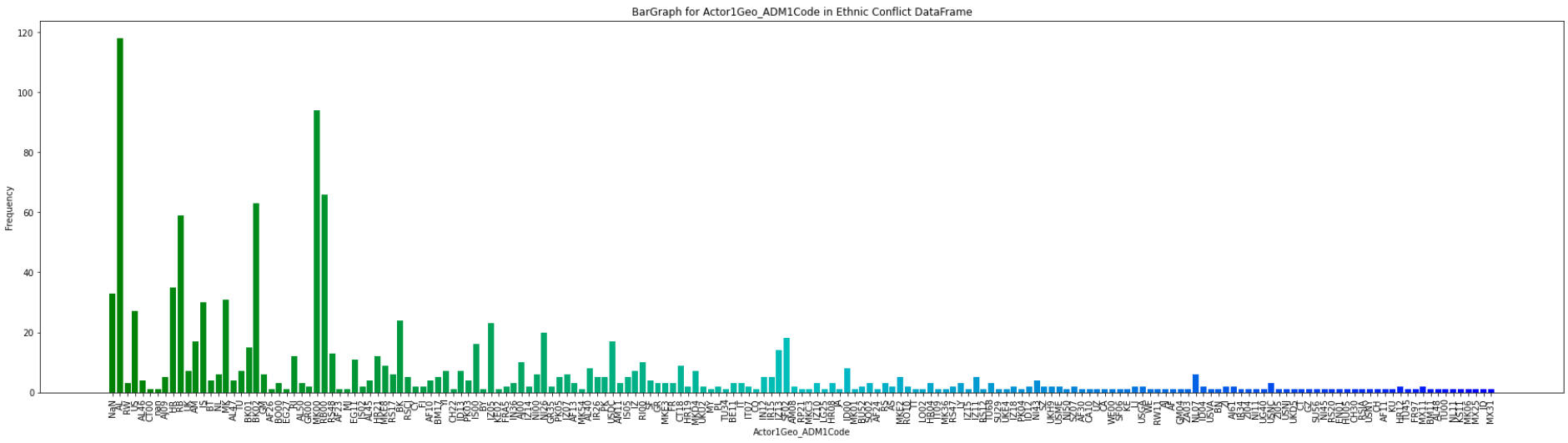
In [6]:
```
dctacn1 = {}
for i in df['Actor1Name']:
    if "NaN" not in dctacn1:
        dctacn1["NaN"] = df['Actor1Name'].isnull().sum()
    if i in dctacn1.keys():
        dctacn1[str(i)] = dctacn1[i] + 1
    else:
        dctacn1[str(i)] = 1

green = Color("green")
colors_dctacn1 = list(green.range_to(Color("blue"),len(dctacn1.keys())))
colors_dctacn1 = [color.rgb for color in colors_dctacn1]

plt.figure(figsize=(cm_to_inch(80),cm_to_inch(20)))
plt.xlabel('Actor1Name')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for Actor1Name in Ethnic Conflict DataFrame")
plt.bar(dctacn1.keys(),dctacn1.values(),color=colors_dctacn1)
plt.show()
```



Plotting Bar Graph for "Actor1Geo_ADM1Code" attribute.

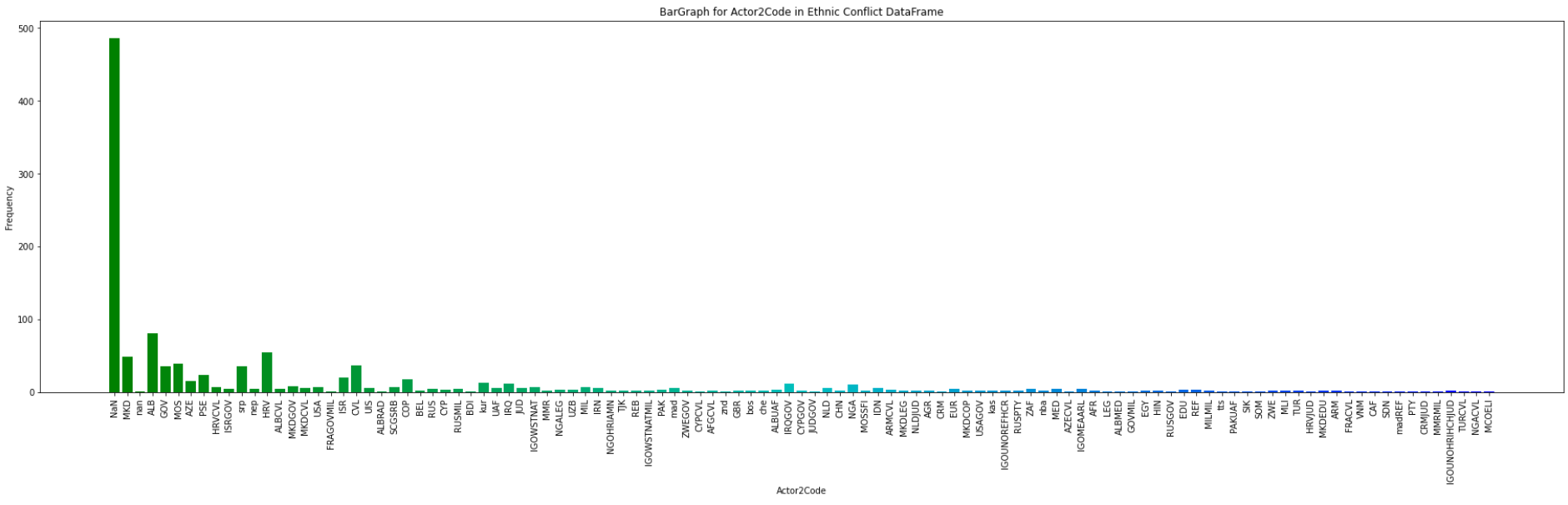
In [7]:
```
dctacg1 = {}
for i in df['Actor1Geo_ADM1Code']:
    if "NaN" not in dctacg1:
        dctacg1["NaN"] = df['Actor1Geo_ADM1Code'].isnull().sum()
    if i in dctacg1.keys():
```

```
        dctacg1[str(i)] = dctacg1[i] +1
    else:
        dctacg1[str(i)] = 1

green = Color("green")
colors_dctacg1 = list(green.range_to(Color("blue"),len(dctacg1.keys())))
colors_dctacg1 = [color.rgb for color in colors_dctacg1]

plt.figure(figsize=(cm_to_inch(80),cm_to_inch(20)))
plt.xlabel('Actor1Geo_ADM1Code')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for Actor1Geo_ADM1Code in Ethnic Conflict DataFrame")
plt.bar(dctacg1.keys(),dctacg1.values(),color = colors_dctacg1)
plt.show()
```



Plotting Bar Graph for "Actor2Code" attribute.

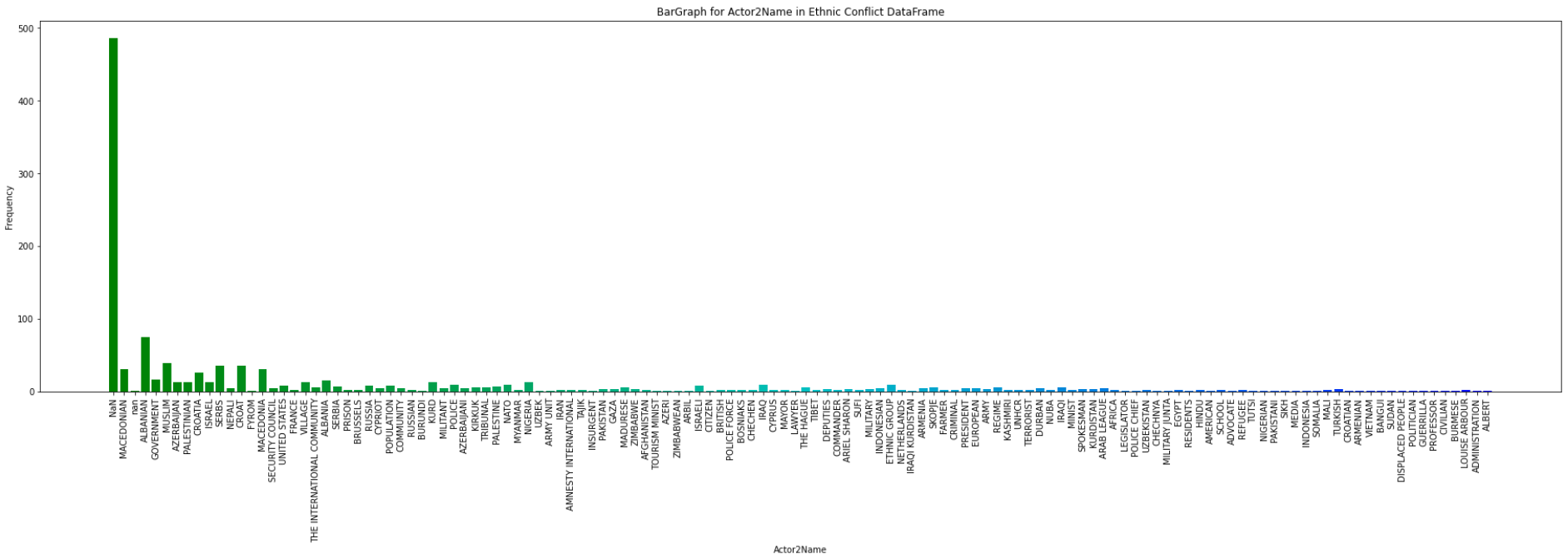
In [8]:
```
dctac2 = {}
for i in df['Actor2Code']:
    if "NaN" not in dctac2:
        dctac2["NaN"] = df['Actor2Code'].isnull().sum()
    if i in dctac2.keys():
        dctac2[str(i)] = dctac2[i] +1
    else:
        dctac2[str(i)] = 1

green = Color("green")
colors_dctac2 = list(green.range_to(Color("blue"),len(dctac2.keys())))
colors_dctac2 = [color.rgb for color in colors_dctac2]

plt.figure(figsize=(cm_to_inch(80),cm_to_inch(20)))
plt.xlabel('Actor2Code')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for Actor2Code in Ethnic Conflict DataFrame")
plt.bar(dctac2.keys(),dctac2.values(), color = colors_dctac2)
plt.show()
```



Plotting Bar Graph for "Actor2Name" attribute.

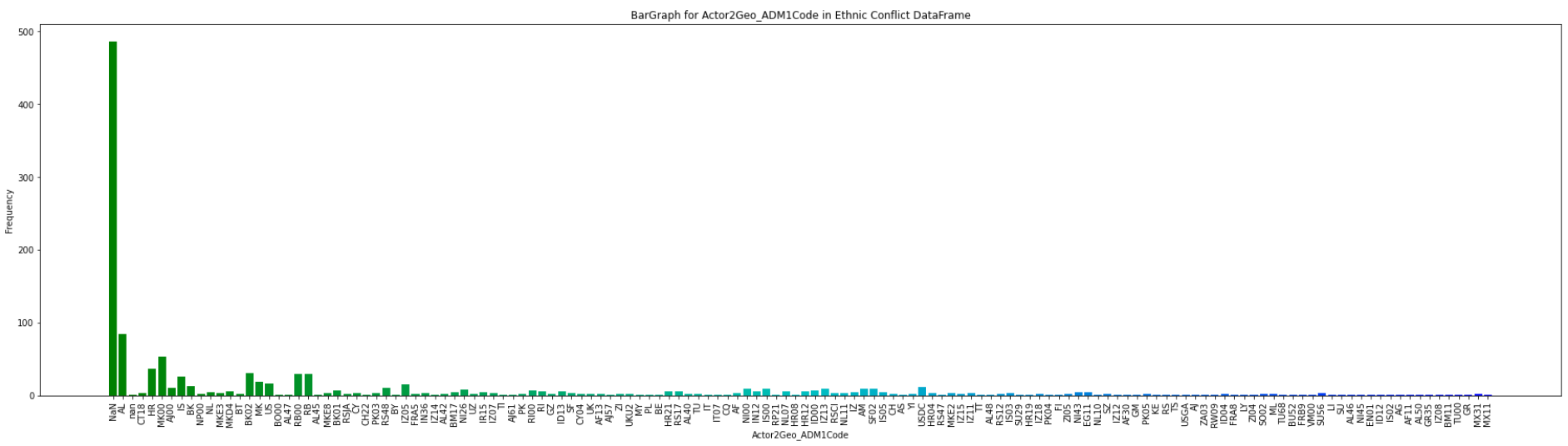
In [9]:
```
dctacn2 = {}
for i in df['Actor2Name']:
    if "NaN" not in dctacn2:
        dctacn2["NaN"] = df['Actor2Name'].isnull().sum()
    if i in dctacn2.keys():
        dctacn2[str(i)] = dctacn2[i] + 1
    else:
        dctacn2[str(i)] = 1

green = Color("green")
colors_dctacn2 = list(green.range_to(Color("blue"),len(dctacn2.keys())))
colors_dctacn2 = [color.rgb for color in colors_dctacn2]

plt.figure(figsize=(cm_to_inch(80),cm_to_inch(20)))
plt.xlabel('Actor2Name')
```

```
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for Actor2Name in Ethnic Conflict DataFrame")
plt.bar(dctacn2.keys(),dctacn2.values(),color = colors_dctacn2)
plt.show()
```



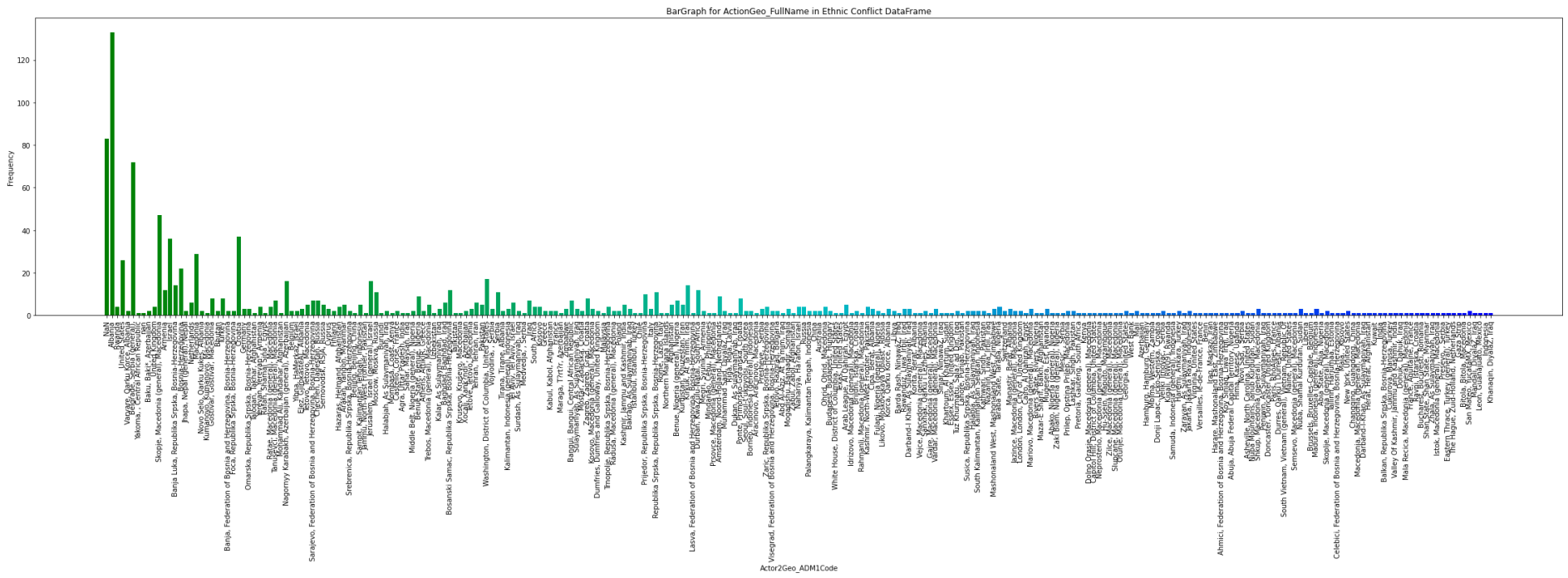
Plotting Bar Graph for "Actor2Geo_ADM1Code" attribute.

In [10]:
```
dctacg2 = {}
for i in df['Actor2Geo_ADM1Code']:
    if "NaN" not in dctacg2:
        dctacg2["NaN"] = df['Actor2Geo_ADM1Code'].isnull().sum()
    if i in dctacg2.keys():
        dctacg2[str(i)] = dctacg2[i] +1
    else:
        dctacg2[str(i)] = 1

green = Color("green")
colors_dctacg2 = list(green.range_to(Color("blue"),len(dctacg2.keys())))
colors_dctacg2 = [color.rgb for color in colors_dctacg2]

plt.figure(figsize=(cm_to_inch(80),cm_to_inch(20)))
plt.xlabel('Actor2Geo_ADM1Code')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for Actor2Geo_ADM1Code in Ethnic Conflict DataFrame")
plt.bar(dctacg2.keys(),dctacg2.values(),color = colors_dctacg2)
plt.show()
```



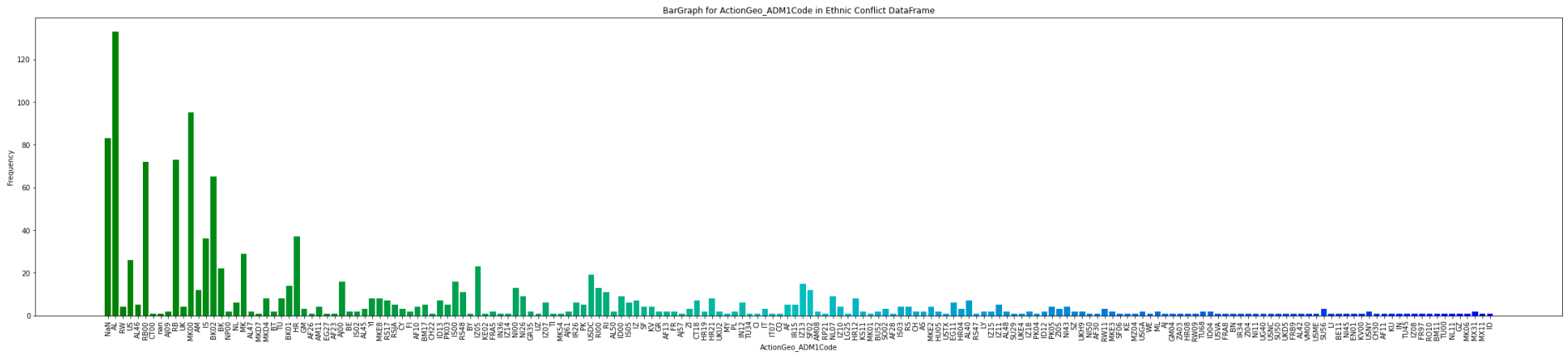
Plotting Bar Graph for "ActionGeo_FullName" attribute.

In [11]:
```
dctagf = {}
for i in df['ActionGeo_FullName']:
    if "NaN" not in dctagf:
        dctagf["NaN"] = df['ActionGeo_FullName'].isnull().sum()
    if i in dctagf.keys():
        dctagf[str(i)] = dctagf[i] +1
    else:
        dctagf[str(i)] = 1

green = Color("green")
colors_dctagf = list(green.range_to(Color("blue"),len(dctagf.keys())))
colors_dctagf = [color.rgb for color in colors_dctagf]

plt.figure(figsize=(cm_to_inch(100),cm_to_inch(20)))
plt.xlabel('Actor2Geo_ADM1Code')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for ActionGeo_FullName in Ethnic Conflict DataFrame")
plt.bar(dctagf.keys(),dctagf.values(), color = colors_dctagf)
plt.show()
```



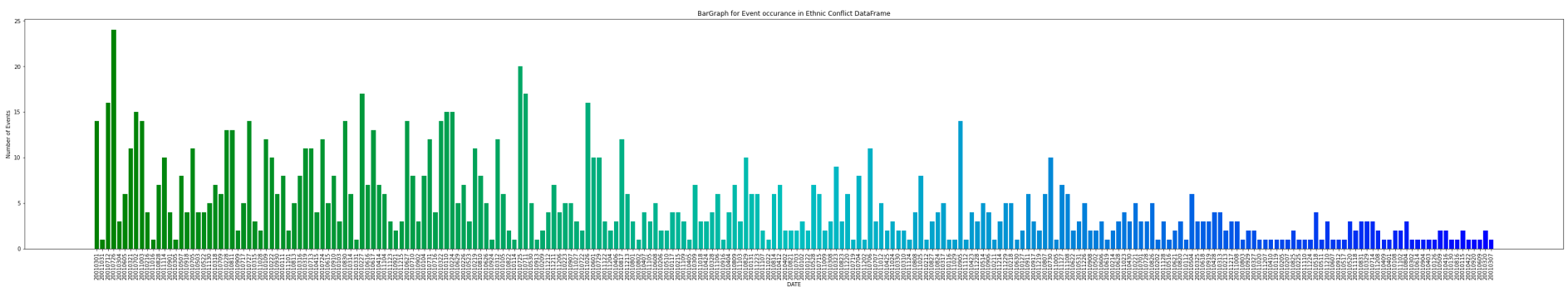
Plotting Bar Graph for "ActionGeo_ADM1Code" attribute.

In [12]:
```
dctagc = {}
for i in df['ActionGeo_ADM1Code']:
    if "NaN" not in dctagc:
        dctagc["NaN"] = df['ActionGeo_FullName'].isnull().sum()
    if i in dctagc.keys():
        dctagc[str(i)] = dctagc[i] +1
    else:
        dctagc[str(i)] = 1

green = Color("green")
colors_dctagc = list(green.range_to(Color("blue"),len(dctagc.keys())))
colors_dctagc = [color.rgb for color in colors_dctagc]

plt.figure(figsize=(cm_to_inch(100),cm_to_inch(20)))
plt.xlabel('ActionGeo_ADM1Code')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.title("BarGraph for ActionGeo_ADM1Code in Ethnic Conflict DataFrame")
plt.bar(dctagc.keys(),dctagc.values(), color = colors_dctagc)
plt.show()
```



Plotting Bar Graph for "SQLDATE" attribute.

In [13]:
```
dctdate = {}
for i in df['SQLDATE']:
    if str(i) in dctdate.keys():
        dctdate[str(i)] = dctdate[str(i)] +1
    else:
        dctdate[str(i)] = 1

green = Color("green")
colors_dctdate = list(green.range_to(Color("blue"),len(dctdate.keys())))
colors_dctdate = [color.rgb for color in colors_dctdate]

plt.figure(figsize=(cm_to_inch(130),cm_to_inch(20)))
plt.xlabel('DATE')
plt.ylabel('Number of Events')
plt.xticks(rotation=90)
plt.title("BarGraph for Event occurance in Ethnic Conflict DataFrame")
plt.bar(dctdate.keys(),dctdate.values(), color = colors_dctdate)
plt.show()
```



By analysing the above graph it is clear that so many Null (NaN) values are present in the given dataset. for getting a better result null values should be dropped from it.

Cleaning data

Ethnic Conflicts 2001 dataset comprises of 1176 rows and 10 columns originally but for the purposes of exploratory analysis we have to drop rows having "NaN" value. Now after droping "NaN", dataset have 622 rows and 10 columns.

In [14]:
```
len(df) #total data before removing "NaN" Data
```

Out[14]: 1176

In [15]:
```
df = df.dropna() #Droping "Nan" Data from dataframe
```

In [16]:
```
len(df) #total data After removing "NaN" Data
```

Out[16]: 622

In [17]:
```
df.head(5) #Displaying the top five rows After removing "NaN" Data
```

Out[17]:

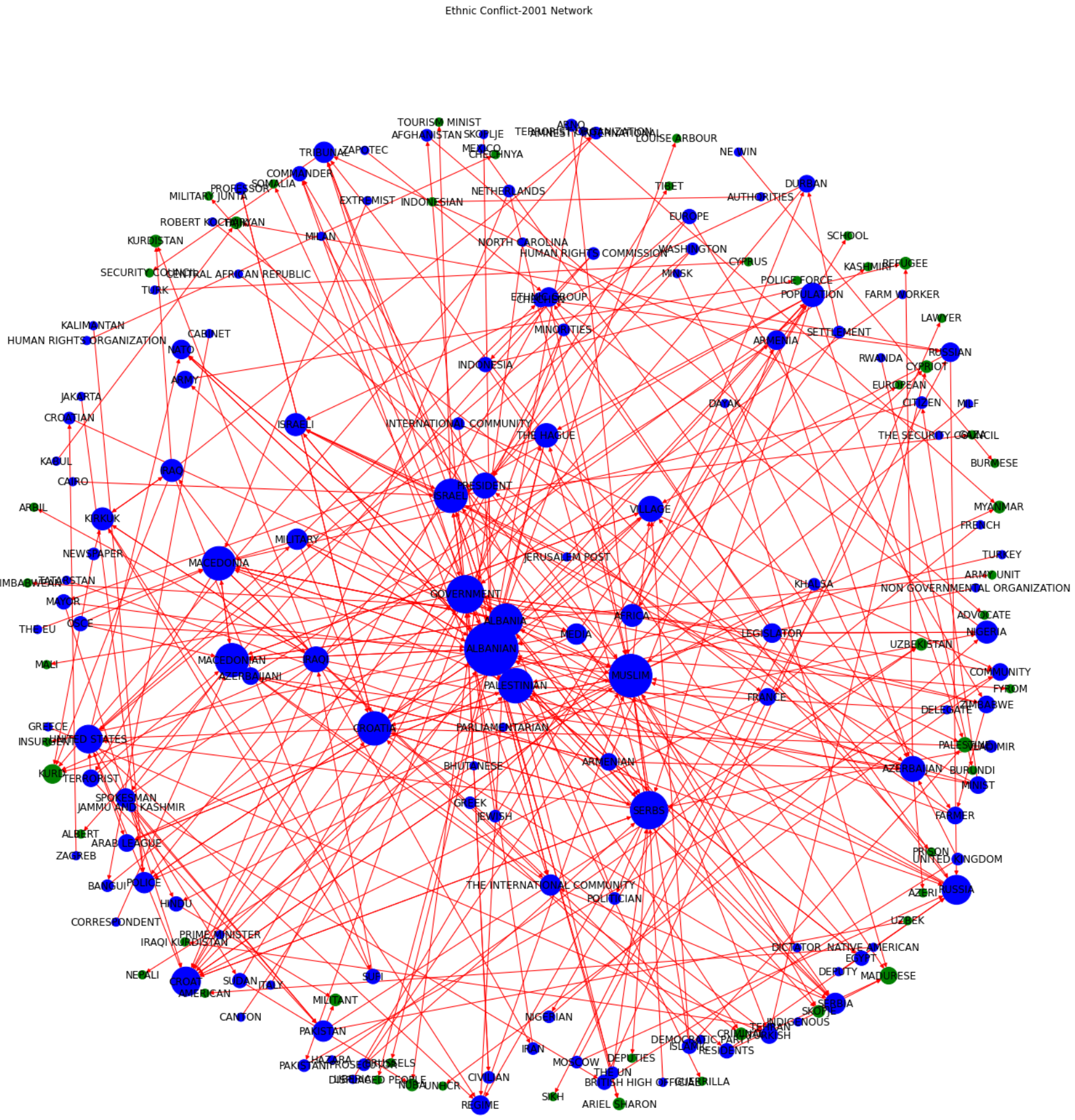
	Actor1Code	Actor1Name	Actor1Geo_ADM1Code	Actor2Code	Actor2Name	Actor2Geo_ADM1Code	ActionGeo_FullName	ActionGeo_ADM1Code	Yea
0	ALB	ALBANIAN	AL	MKD	MACEDONIAN	AL	Albania	AL	200
4	CVL	POPULATION	AL	ALB	ALBANIAN	AL	Belgrade, Serbia (general),	RB00	200
5	CAF	BANGUI	CT00	GOV	GOVERNMENT	CT18	Yakoma, , Central African Republic	CT00	200
11	GOV	MINIST	AL	ALB	ALBANIAN	MK00	Skopje, Macedonia (general), Macedonia	MK00	200
12	ARM	ARMENIA	AM	AZE	AZERBAIJAN	AJ00	Armenia	AM	200

Creating The Network

Describing the set of actors; source and targets.

Here to construct the network we are using ***"Actor1Name"*** as **"Source"** and ***"Actor2Name"*** as **"Target"**. And also using different color for **Source (Blue)** and for **Target (Green)** so that source and Target can be identified easily. The size of the node is depened upon the degree of it. higher the degree larger the size.

In [18]:
```
df_new= pd.DataFrame({'from':df['Actor1Name'], 'to':df['Actor2Name']})
G = nx.from_pandas_edgelist(df_new, 'from', 'to',create_using=nx.DiGraph())
plt.figure(figsize=(cm_to_inch(50),cm_to_inch(50)))

colors = []
for node in G:
    if node in df_new["from"].values:
        colors.append("blue")
    else: colors.append("green")

pos = nx.spring_layout(G,.9)
deg = dict(G.degree)
nx.draw(G, pos, cmap=plt.get_cmap('jet'), with_labels=True, node_color=colors, edge_color="red",node_size=[v * 100 for v in deg.va
plt.title("Ethnic Conflict-2001 Network")
plt.show()
```



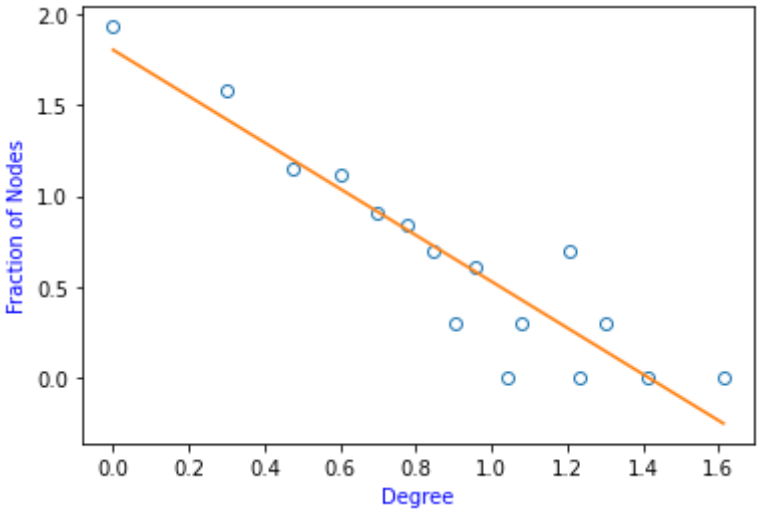
By analysing the above network we can say that among the all node some having large size (having high degree) have more ethnic conflicts as compare to others for example ALBANIAN, GOVERNMENT, UNITED STATE are having large conflicts.

Plotting the degree distribution of this network.

And fitting a line (power law) on the degree distribution. And printing the values(exponent)

In [20]:
```
# generating the degree distribution
k = []
Pk = []

for node in list(G.nodes()):
    degree = G.degree(nbunch=node)
    try:
        pos = k.index(degree)
    except ValueError as e:
        k.append(degree)
        Pk.append(1)
    else:
        Pk[pos] += 1

logk = []
logPk = []

for i in range(len(k)):
    logk.append(math.log10(k[i]))
    logPk.append(math.log10(Pk[i]))
order = np.argsort(logk)
logk_array = np.array(logk)[order]
logPk_array = np.array(logPk)[order]

plt.xlabel('Degree').set_color('blue')
```

```
plt.ylabel('Fraction of Nodes').set_color('blue')

plt.plot(logk_array, logPk_array, "o",markerfacecolor="none") # plotting the degree distribution

m, c = np.polyfit(logk_array, logPk_array, 1) # fitting a line to the degree distribution
plt.plot(logk_array, m*logk_array + c, "-")

plt.show()
print("Values of exponents are Below\n")
print(logPk_array)

print("\nslope of line : ",m)
```




```
Values of exponents are Below

[1.93449845 1.5797836  1.14612804 1.11394335 0.90308999 0.84509804
 0.69897    0.30103    0.60205999 0.         0.30103    0.69897
 0.         0.30103    0.         0.        ]

slope of line :  -1.2789479459930169
```

Finding the number of organizations and attacks using network properties, and other statistical measures

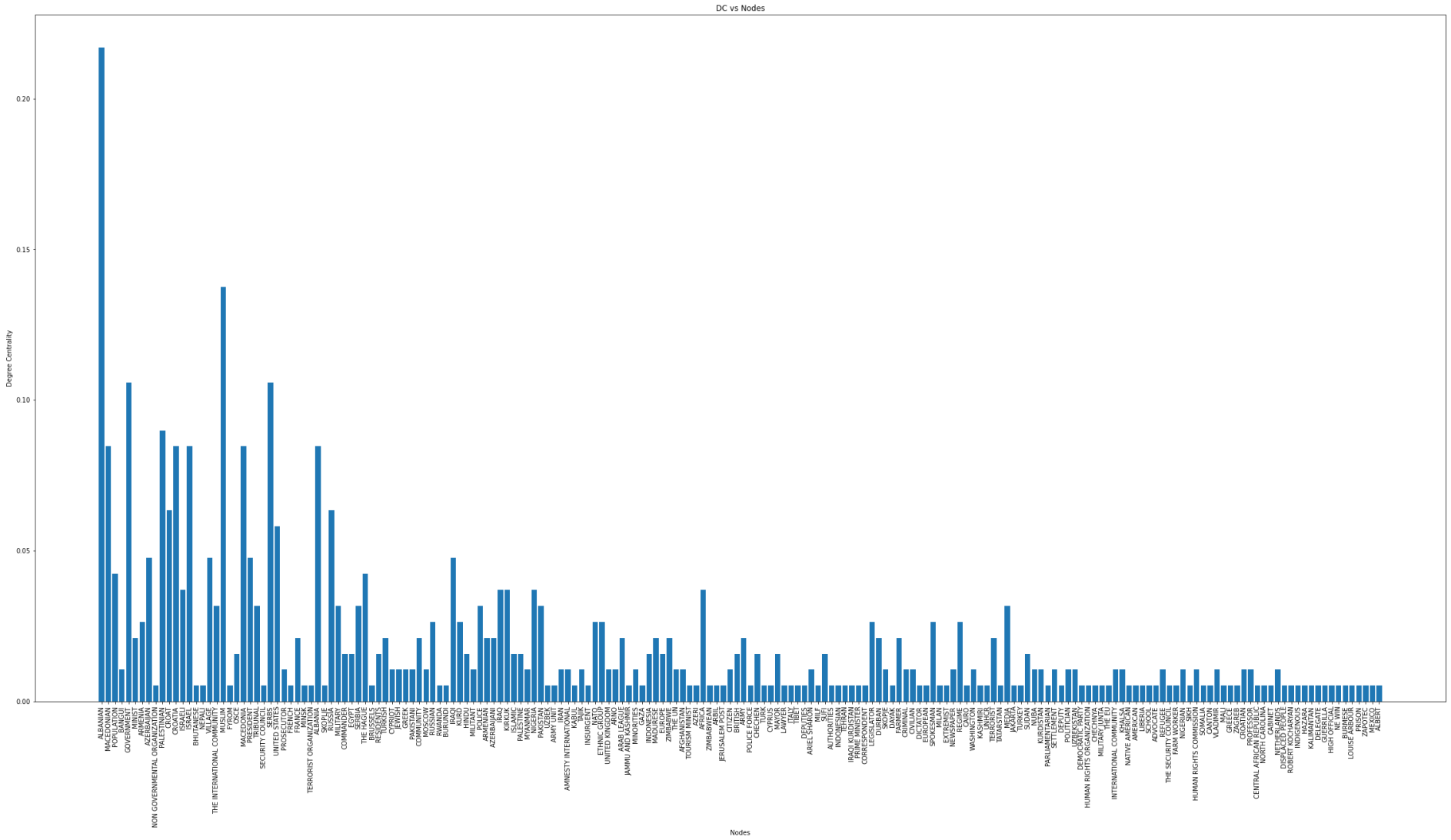
In [21]:
```
print("the number of organizations are : {}".format(len(list(G.nodes()))))
print("the number of attacks are : {}".format(len(list(G.edges()))))
print("Other statistical measures")
print(nx.info(G))
print("Density of graph is {}".format(nx.density(G)))
print("Average node connectivity of graph is {}".format(nx.average_node_connectivity(G)))
```

```
the number of organizations are : 190
the number of attacks are : 332
Other statistical measures
Name:
Type: DiGraph
Number of nodes: 190
Number of edges: 332
Average in degree:   1.7474
Average out degree:   1.7474
Density of graph is 0.009245335561125034
Average node connectivity of graph is 0.3093010303536619
```

The degree centrality, closeness centrality, and normalized betweenness centrality (excluding endpoints) of node 15

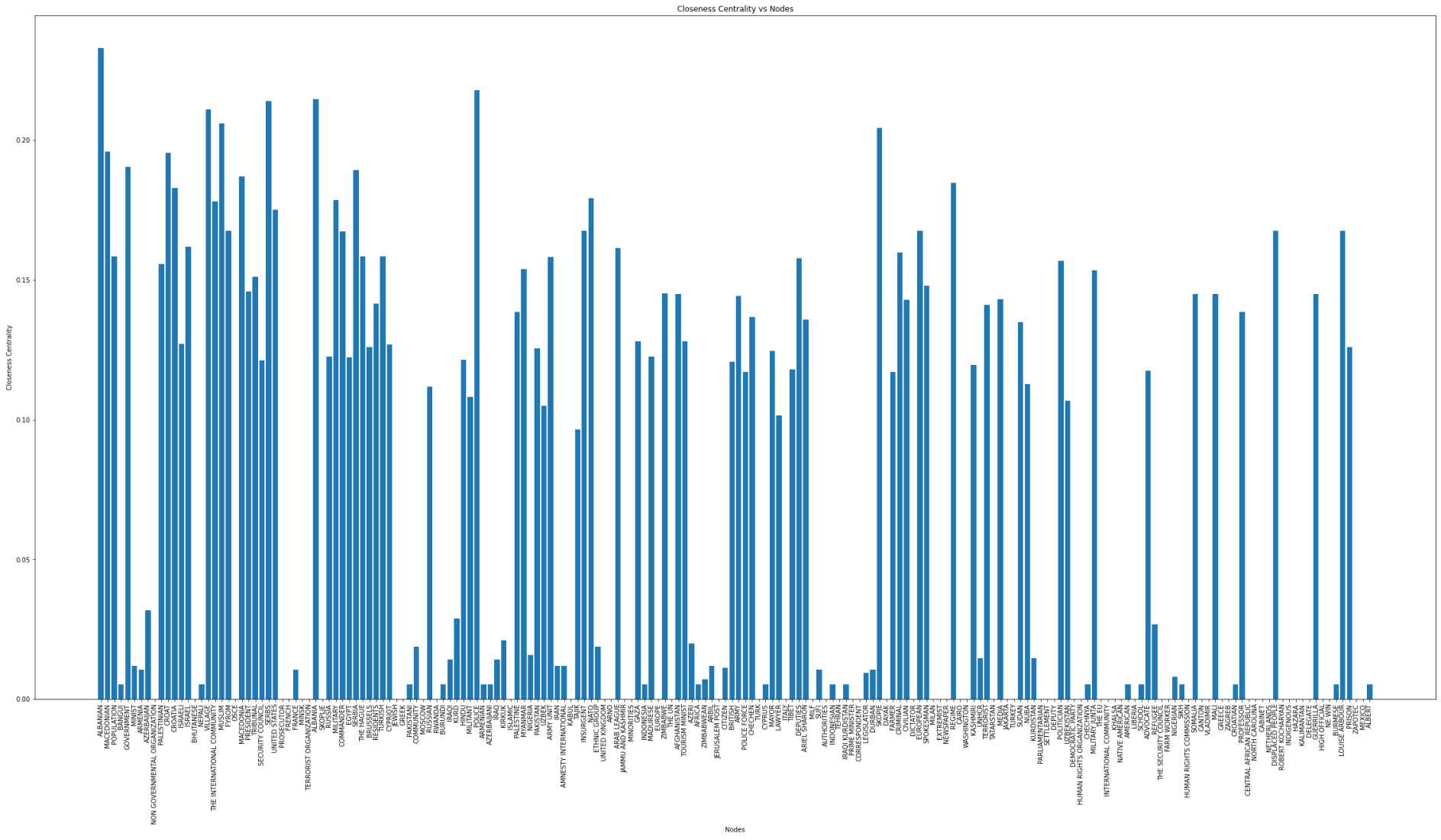
Creating bargraph for Degree Centrality

In [22]:
```
dct_dc = dict(nx.degree_centrality(G))
n_dc = list(dct_dc.keys())
dc = list(dct_dc.values())
plt.figure(figsize=(cm_to_inch(100),cm_to_inch(50)))
plt.xlabel('Nodes')
plt.ylabel('Degree Centrality')
plt.xticks(rotation=90)
plt.title("DC vs Nodes")
plt.bar(n_dc,dc)
plt.show()
```



Creating bargraph for Closeness Centrality

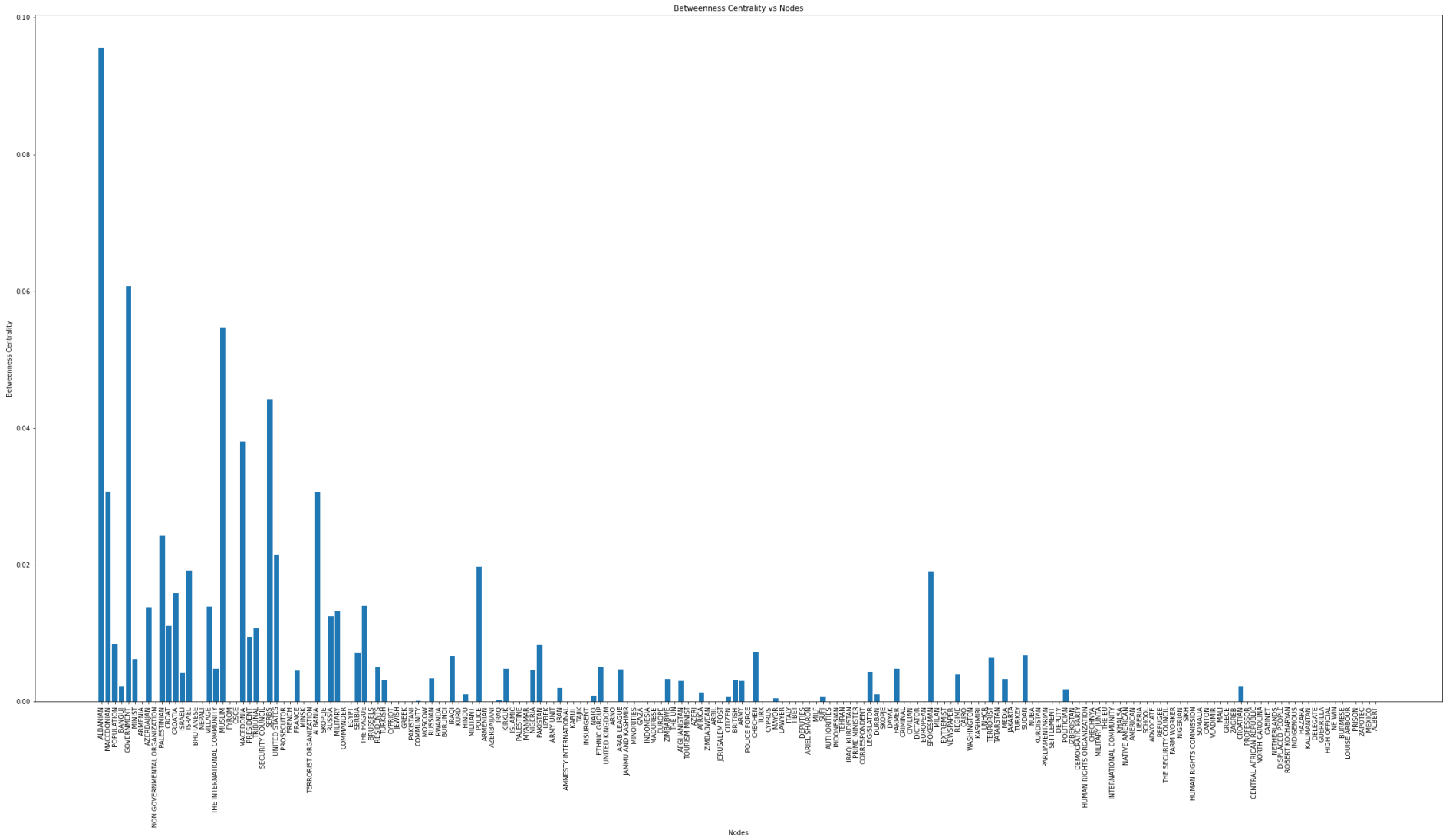
In [23]:
```
dct_cc = nx.closeness_centrality(G)
n_cc = list(dct_cc.keys())
cc = list(dct_cc.values())
plt.figure(figsize=(cm_to_inch(100),cm_to_inch(50)))
plt.xlabel('Nodes')
plt.ylabel('Closeness Centrality')
plt.xticks(rotation=90)
plt.title("Closeness Centrality vs Nodes")
plt.bar(n_cc,cc)
plt.show()
```



Creating bargraph for Betweenness Centrality

In [24]:
```
dct_bc = dict(nx.betweenness_centrality(G,normalized=True,endpoints=False))
n_bc = list(dct_bc.keys())
bc = list(dct_bc.values())
plt.figure(figsize=(cm_to_inch(100),cm_to_inch(50)))
plt.xlabel('Nodes')
plt.ylabel('Betweenness Centrality')
plt.xticks(rotation=90)
plt.title("Betweenness Centrality vs Nodes")
```

```
plt.bar(n_bc,bc)
plt.show()
```



Printing the Degree Centrality, Closeness Centrality and Betweenness Centrality of 15th node

In [25]:
```
print("Degree of Centrality of Node 15 ({}) is {}".format(list(dct_dc.items())[14][0],list(dct_dc.items())[14][1]))
print("Closeness Centrality of Node 15 ({}) is {}".format(list(dct_cc.items())[14][0],list(dct_cc.items())[14][1]))
print("Betweenness Centrality of Node 15 ({}) is {}".format(list(dct_bc.items())[14][0],list(dct_bc.items())[14][1]))
```

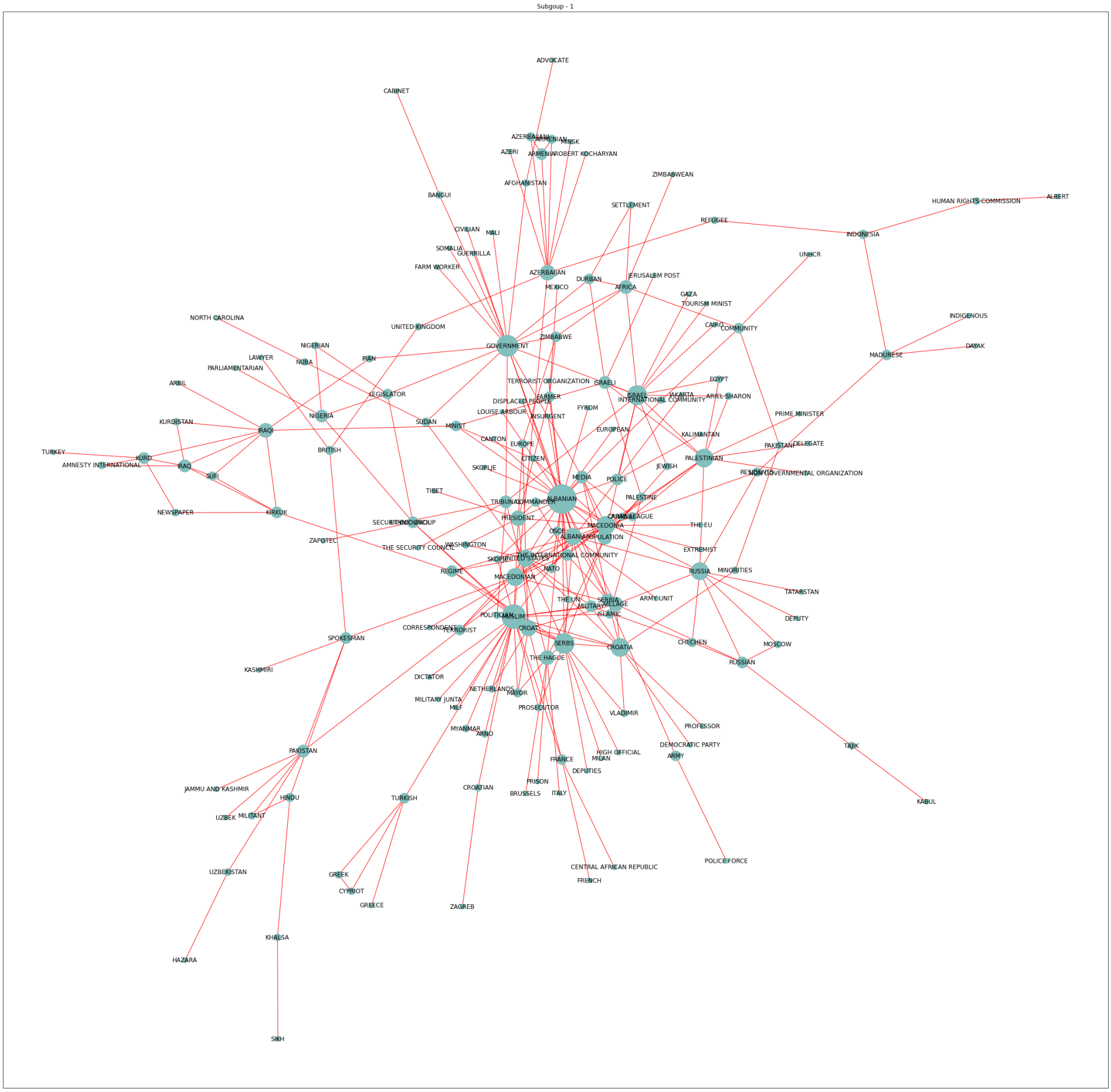
```
Degree of Centrality of Node 15 (BHUTANESE) is 0.005291005291005291
Closeness Centrality of Node 15 (BHUTANESE) is 0.0
Betweenness Centrality of Node 15 (BHUTANESE) is 0.0
```

Finding the subgroups for the given network

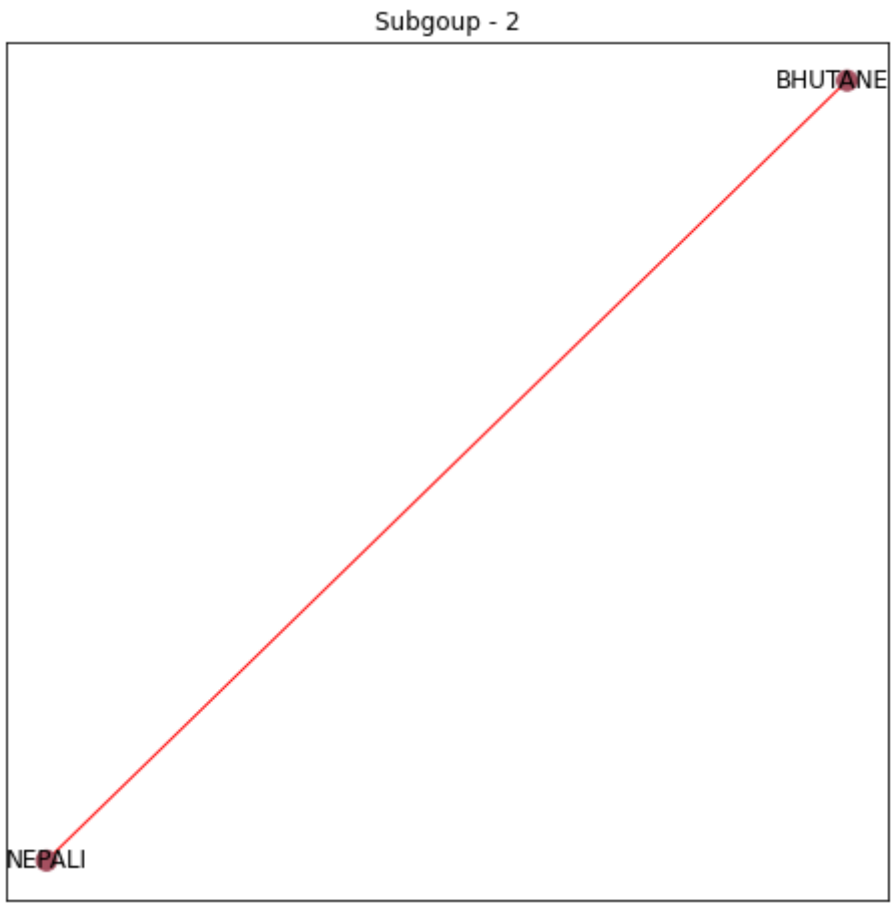
In [26]:
```
G1 = nx.from_pandas_edgelist(df_new, 'from', 'to')
S = [G1.subgraph(c).copy() for c in nx.connected_components(G1)]
print("Total number of Subgoups are  {}".format(len(S))) #printing the total number of subgraphs
color_subgroup = []

for i in range(len(S)):
    color_subgroup.append('#%06X' % randint(0, 0xFFFFFF))
j = 0

for i in S: #adjusting the size of figure according to the number of nodes in a graph
    if(i.number_of_nodes() > 5):
        nodes_num = 100
    else:
        nodes_num = 20
    plt.figure(figsize=(cm_to_inch(nodes_num),cm_to_inch(nodes_num)))
    dg = dict(i.degree)
    nx.draw_networkx(i, with_labels=True,edge_color = "red", node_color = color_subgroup[j],node_size=[v * 100 for v in dg.values(
    j = j + 1
    ttl= "Subgoup - "+str(j)
    plt.title(ttl)
    plt.show()
```

```
Total number of Subgoups are  10
```

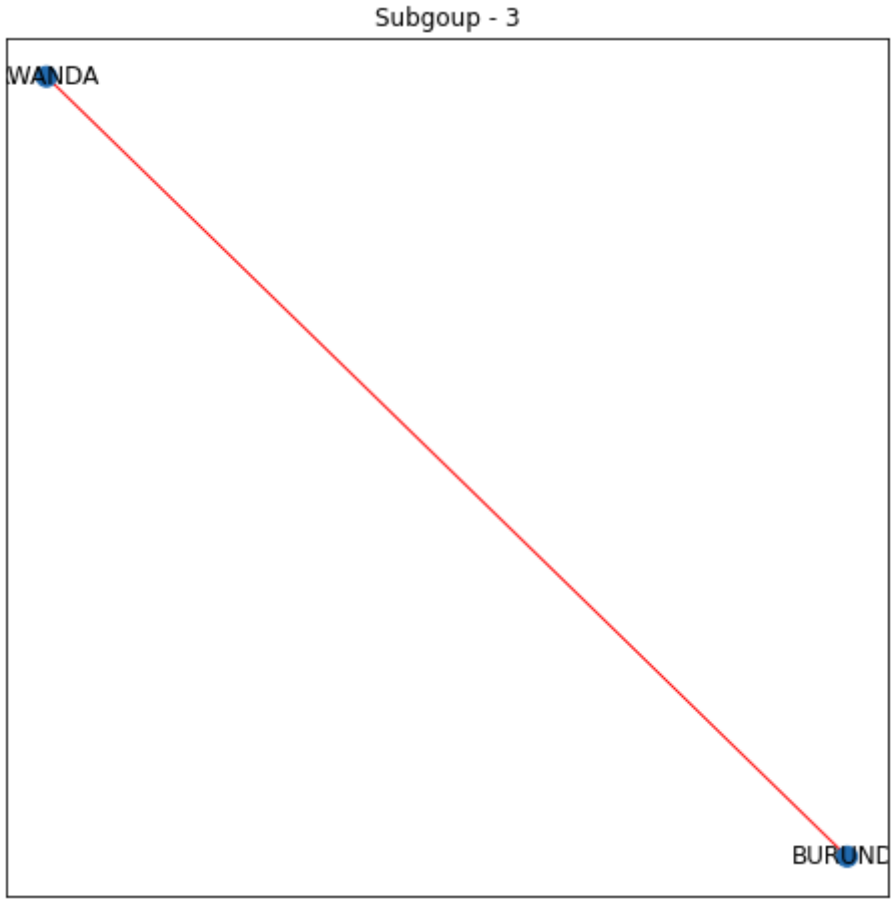
Subgoup - 1



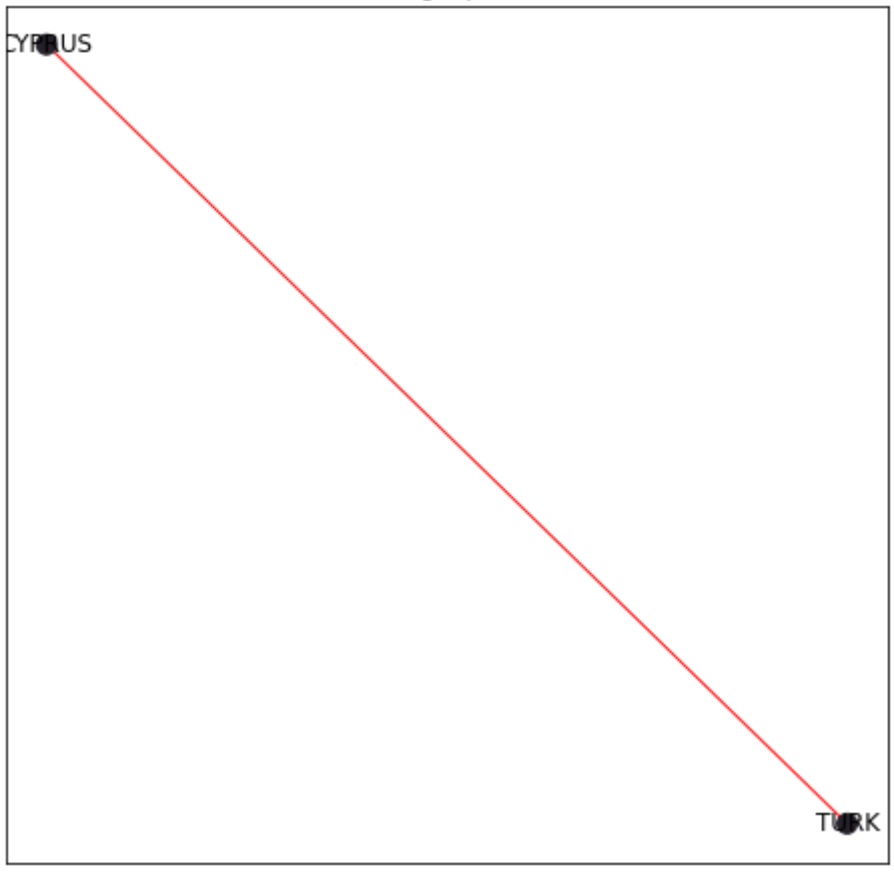
Subgoup - 2



Subgoup - 3

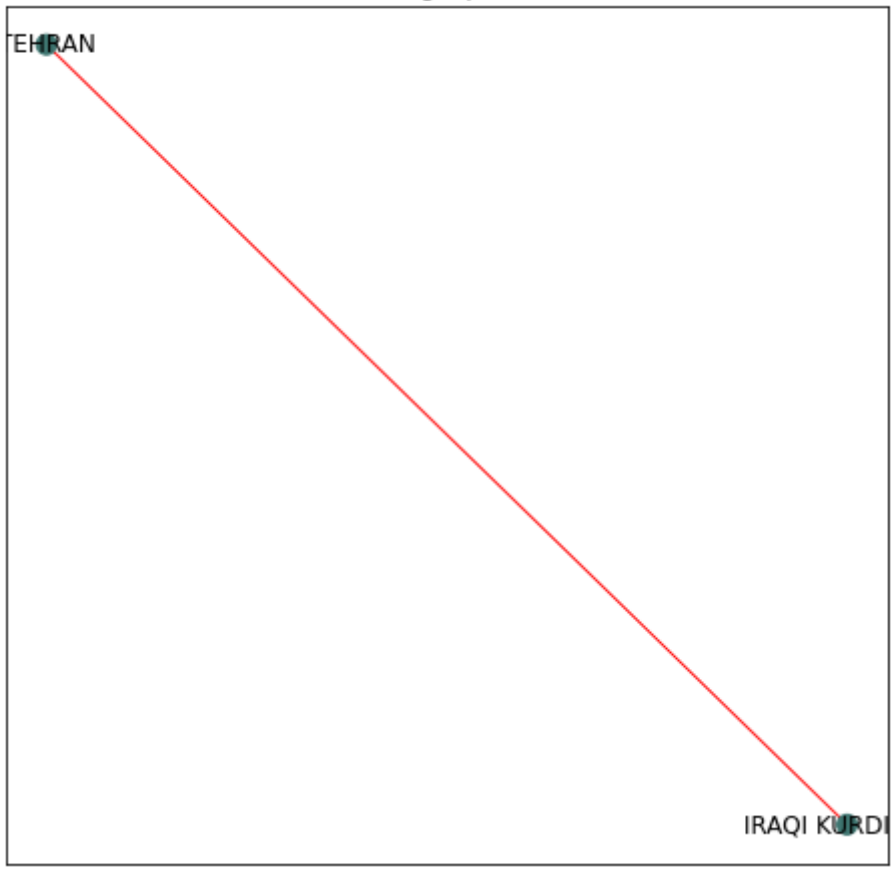


Subgoup - 4

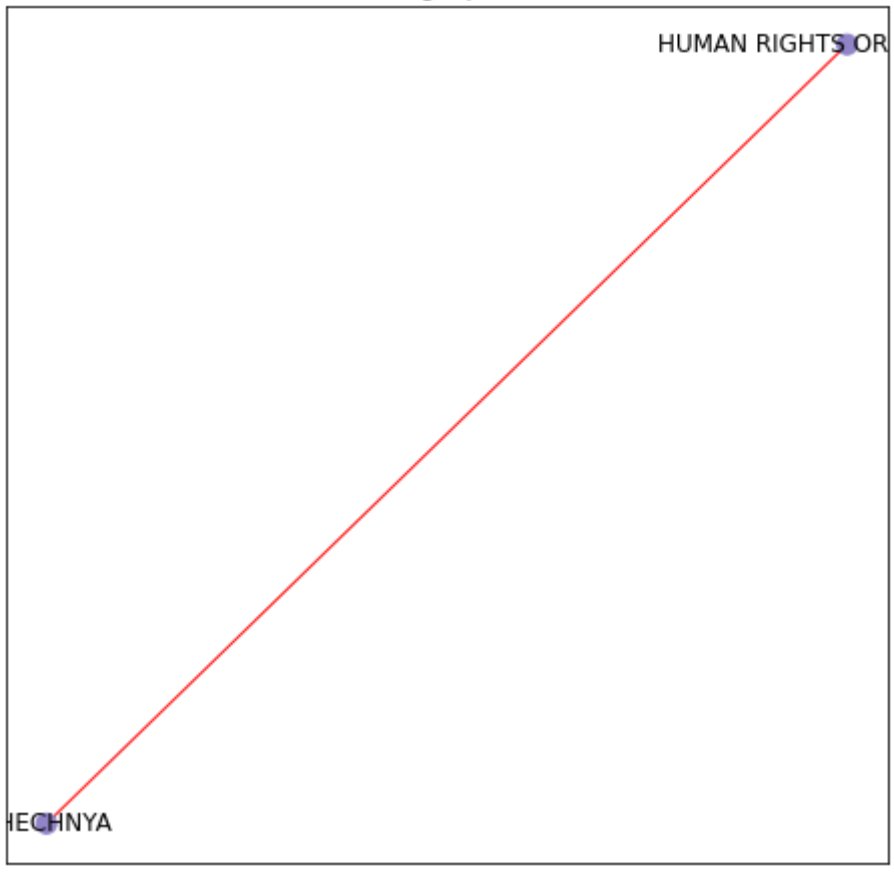


Subgoup - 5

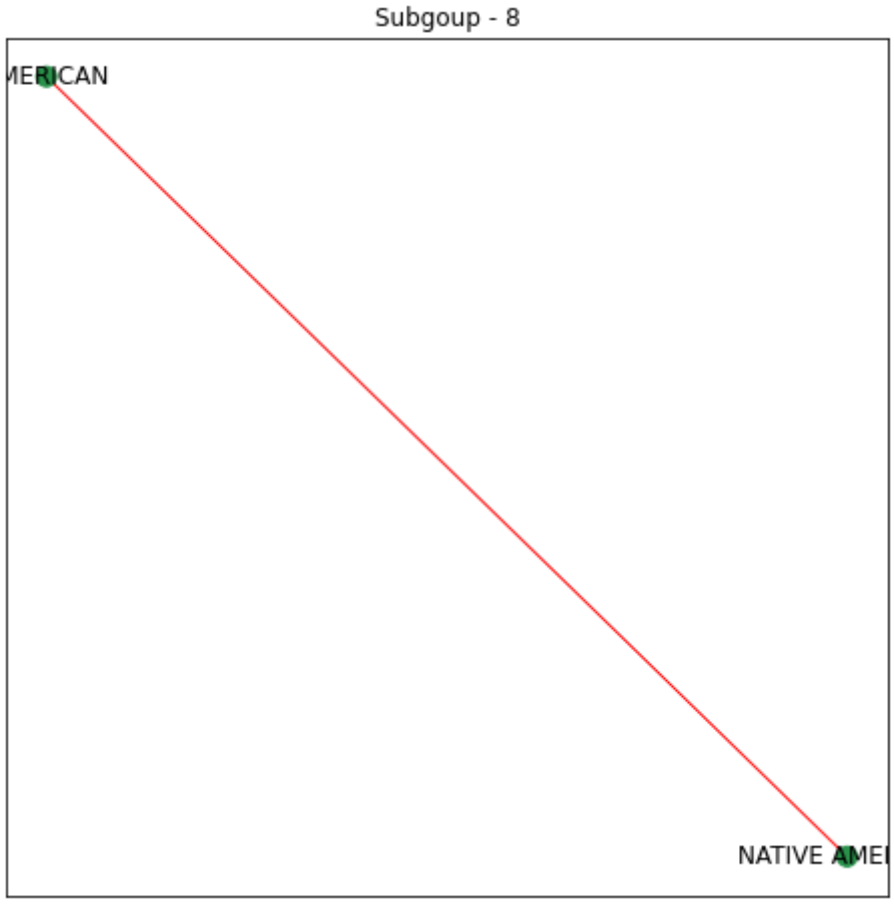
Subgoup - 6

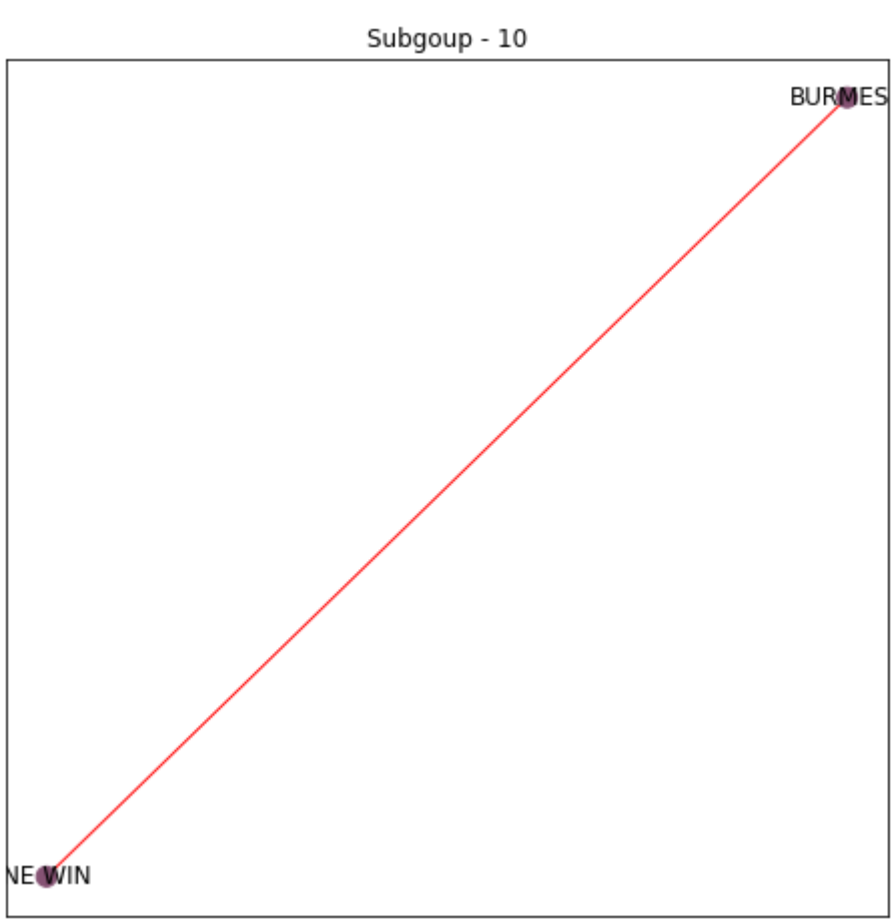


Subgoup - 7



Subgoup - 8

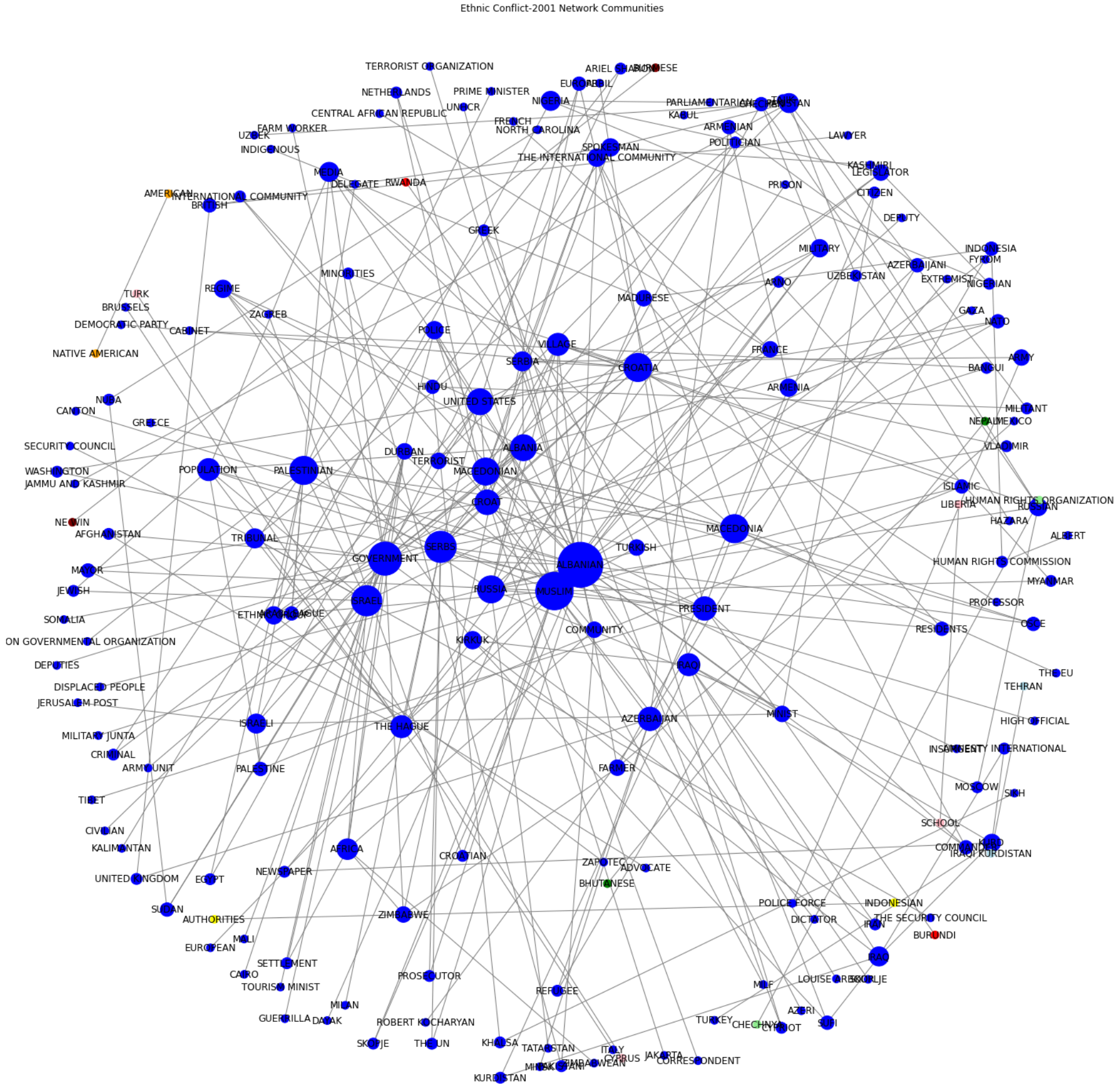




Implementing community detection using Girvan-Newman

In [27]:
```python
def edge_to_remove(graph):
    G_dict = nx.edge_betweenness_centrality(graph)
    edge = ()
    # extract the edge with highest edge betweenness centrality score
    for key, value in sorted(G_dict.items(), key=lambda item: item[1], reverse = True):
        edge = key
        break

    return edge

def girvan_newman(graph):
    # find number of connected components
    sg = nx.connected_components(graph)
    sg_count = nx.number_connected_components(graph)

    while(sg_count == 1):
        graph.remove_edge(edge_to_remove(graph)[0], edge_to_remove(graph)[1])
        sg = nx.connected_components(graph)
        sg_count = nx.number_connected_components(graph)
    return sg

# find communities in the graph
c = girvan_newman(G1.copy())

# find the nodes forming the communities
node_groups = []
for i in c:
    node_groups.append(list(i))

# plot the communities
color_map = []
for node in G1:
```

```
    if node in node_groups[0]:
        color_map.append('blue')
    elif node in node_groups[1]:
        color_map.append('green')
    elif node in node_groups[2]:
        color_map.append('red')
    elif node in node_groups[3]:
        color_map.append('pink')
    elif node in node_groups[4]:
        color_map.append('yellow')
    elif node in node_groups[5]:
        color_map.append('lightblue')
    elif node in node_groups[6]:
        color_map.append('lightgreen')
    elif node in node_groups[7]:
        color_map.append('orange')
    elif node in node_groups[8]:
        color_map.append('lightpink')
    else:
        color_map.append('maroon')


pos = nx.spring_layout(G1,.9)
d = dict(G1.degree)
plt.figure(figsize=(cm_to_inch(50),cm_to_inch(50)))
nx.draw(G1, pos, cmap=plt.get_cmap('jet'), edge_color="gray", node_color=color_map, with_labels=True, node_size=[v * 100 for v in
plt.title("Ethnic Conflict-2001 Network Communities")
plt.show()
```



Find the strongly connected component, weakly connected component, average shortest path length, diameter,transitivity and average clustering coefficient.

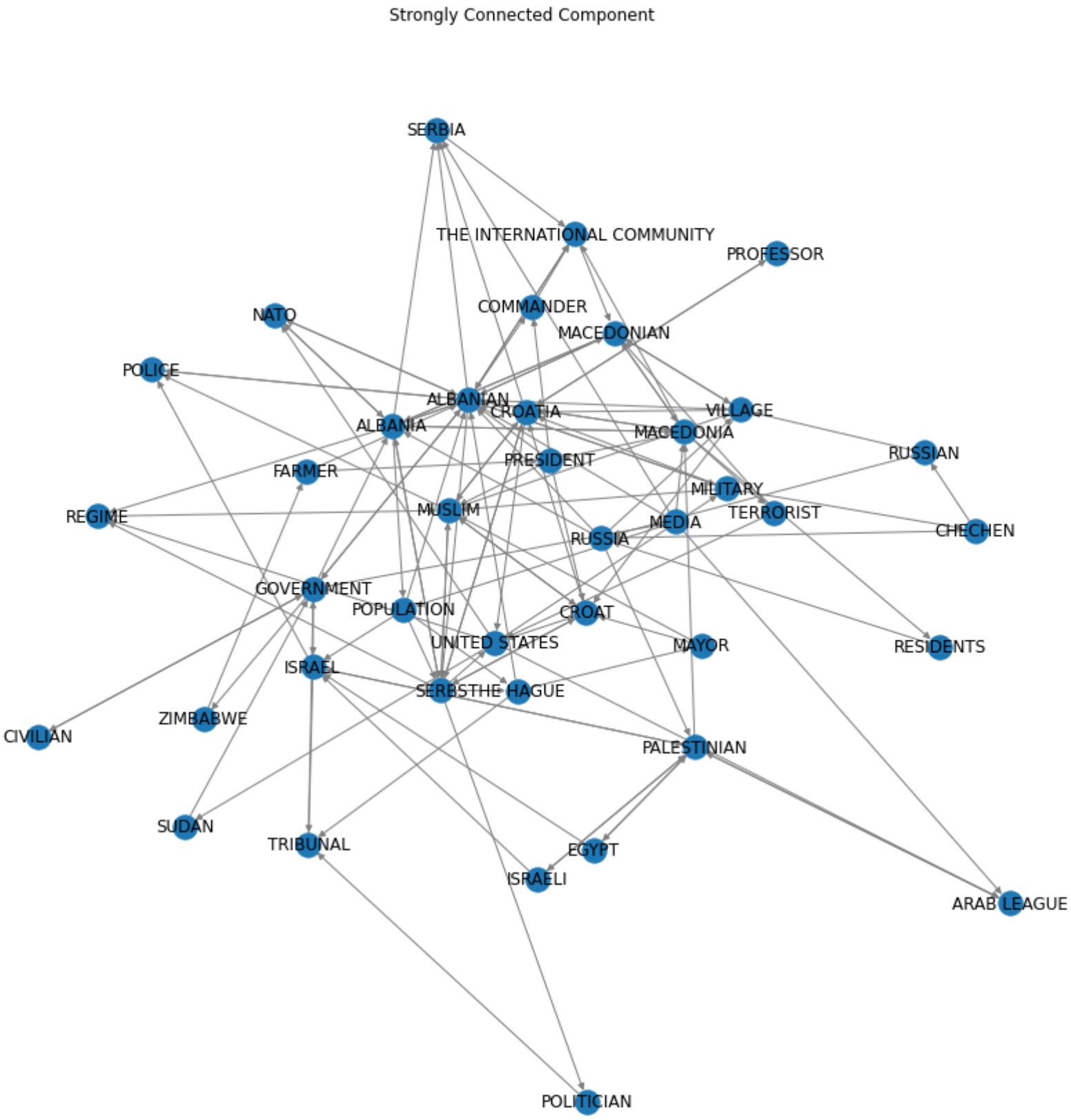
In [28]:
```
scc = nx.number_strongly_connected_components(G)
```

```
print("Toal number of strongly connected Components are ",scc)
```

```
Toal number of strongly connected Components are  145
```

In [29]:
```
largest = max(nx.strongly_connected_components(G), key=len)
scg = nx.strongly_connected_components(G)
print("Largest strongly connected Component having {} nodes ".format(len(list(largest))))
for graph in scg:
    k = G.subgraph(graph)
    if(len(k.nodes) == len(list(largest))):
        plt.figure(figsize=(cm_to_inch(30),cm_to_inch(30)))
        pos = nx.spring_layout(k)
        nx.draw(k, pos,edge_color="gray", with_labels=True)
        plt.title("Strongly Connected Component ")
        plt.show()
        break
```

```
Largest strongly connected Component having 40 nodes
```

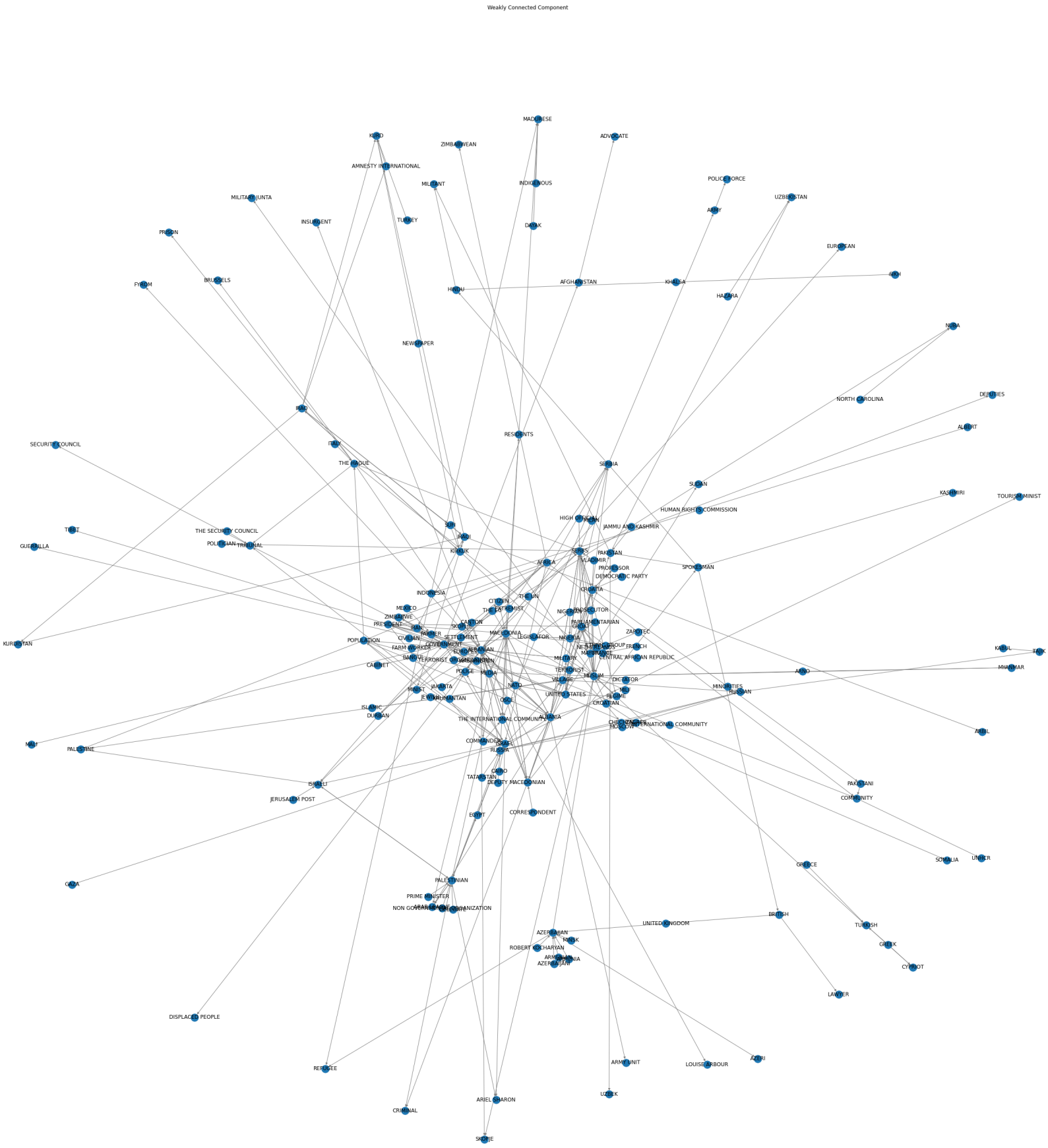


In [30]:
```
wcc = nx.number_weakly_connected_components(G)
print("Toal number of weakly connected Components are ",wcc)
```

```
Toal number of weakly connected Components are  10
```

In [31]:
```
largest_cc = max(nx.weakly_connected_components(G), key=len)
wcg = nx.weakly_connected_components(G)
print("Largest weakly connected Component having {} nodes ".format(len(list(largest_cc))))
for graph in wcg:
    k1 = G.subgraph(graph)
    if(len(k1.nodes) == len(list(largest_cc))):
        plt.figure(figsize=(cm_to_inch(100),cm_to_inch(100)))
        pos = nx.spring_layout(k1)
        nx.draw(k1, pos,edge_color="gray", with_labels=True)
        plt.title("Weakly Connected Component ")
        plt.show()
        break
```

```
Largest weakly connected Component having 172 nodes
```



In [32]:
```
print("Average Shortest Path for longest weakly connected graph is ",nx.average_shortest_path_length(k1))
print("Diameter of largest Strongly connected graph is ",nx.diameter(k))
print("Transitivity of Original graph is ",nx.transitivity(G))
print("Average Clustering of Original graph is ",nx.average_clustering(G))
```

```
Average Shortest Path for longest weakly connected graph is  1.1651026791785666
Diameter of largest Strongly connected graph is  6
Transitivity of Original graph is  0.13418803418803418
Average Clustering of Original graph is  0.11533424068646471
```

In [ ]: