In [102]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

%matplotlib inline
```

In [132]:
```
G = nx.Graph()

nodes = pd.read_csv('pgvertices.csv')
nodes.columns = ["ID", "Longitude", "Latitude","Type","Voltage", "Fregue
ncy", "Name","Operator","ref","wkt_srid_4326"]
nodes["Voltage"] = nodes["Voltage"].str[:6]

links = pd.read_csv('pglinks.csv')
links["voltage"] = links["voltage"].str.split(pat = ";",expand=True)
links = links[(links['v_id_1'] != 341) & (links['v_id_2'] != 341) & (lin
ks['v_id_1'] != 9924) & (links['v_id_2'] != 9924)]
links = links[links['voltage'].notna()]
```

In [133]:
```
nodes.head()
```

Out[133]:

	ID	Longitude	Latitude	Type	Voltage	Freguency	Name	Operator	ref	
0	23470	-90.040427	38.738671	joint	345000	60	NaN	NaN	NaN	SRID=4
1	16854	-93.895894	49.064320	substation	NaN	NaN	NaN	NaN	NaN	SRID=4
2	25563	-75.159609	40.707010	joint	230000	NaN	NaN	NaN	NaN	SRID=
3	6044	-78.839873	42.789666	sub_station	NaN	NaN	NaN	NaN	NaN	SRID=4
4	8477	-94.638969	47.379042	substation	NaN	NaN	NaN	NaN	NaN	SRID=4

In [134]:
```
links.head()
```

Out[134]:

	l_id	v_id_1	v_id_2	voltage	cables	wires	frequency	name	operator	ref
0	36040	13322	13394	230000	6;3;3	NaN	60	NaN	NaN	NaN
3	11669	21844	21845	345000	3	NaN	NaN	West Medway - West Walpole 345kV transmission ...	NaN	389
5	37167	14527	14530	138000	3	NaN	60	NaN	FPL	NaN
7	7195	7383	29044	230000	3	NaN	60	NaN	NaN	NaN
8	42207	20477	2794	500000	3	NaN	NaN	NaN	Bonneville Power Administration	Grizzly - Captain Jack

In [135]:
```
nodes = nodes.transpose()
n_attr = len(nodes.index.values)
attr = nodes.index.values
```

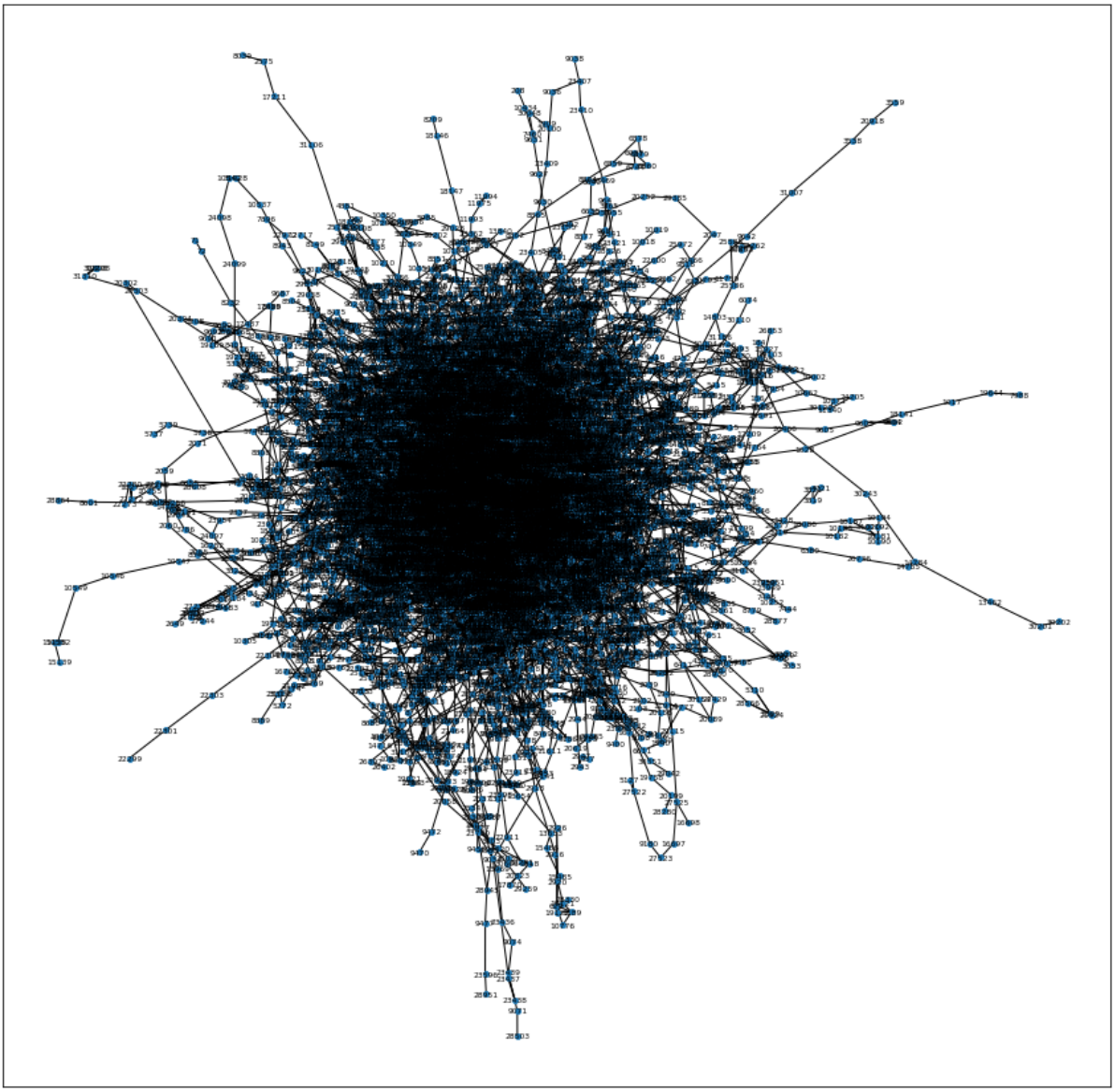
In [136]:
```
for n in nodes.columns:
        attr_node=dict(list(zip(attr, nodes[n].values)))
        G.add_node(n, v_id=nodes[n].values[0],pos=(nodes[n].values[2],no
des[n].values[1]),
                   voltage=nodes[n].values[4])
```

In [137]:
```
for L in links.index.values:
        G.add_edge(links['v_id_1'][L],links['v_id_2'][L],
                   weight = links['voltage'][L],distance=links['length_m'
][L])
```

In [138]:
```
comps = sorted(nx.connected_components(G),key = len, reverse=True)
nodes_gc=comps[0]
g = nx.subgraph(G,nodes_gc)
```

In [139]:
```
plt.figure(1,figsize=(15,15))
pos=nx.spring_layout(g)
nx.draw_networkx(g,pos,node_size=15,font_size=6)
```

```
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/networkx/
drawing/nx_pylab.py:579: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be remo
ved in 3.3. Use np.iterable instead.
  if not cb.iterable(width):
```



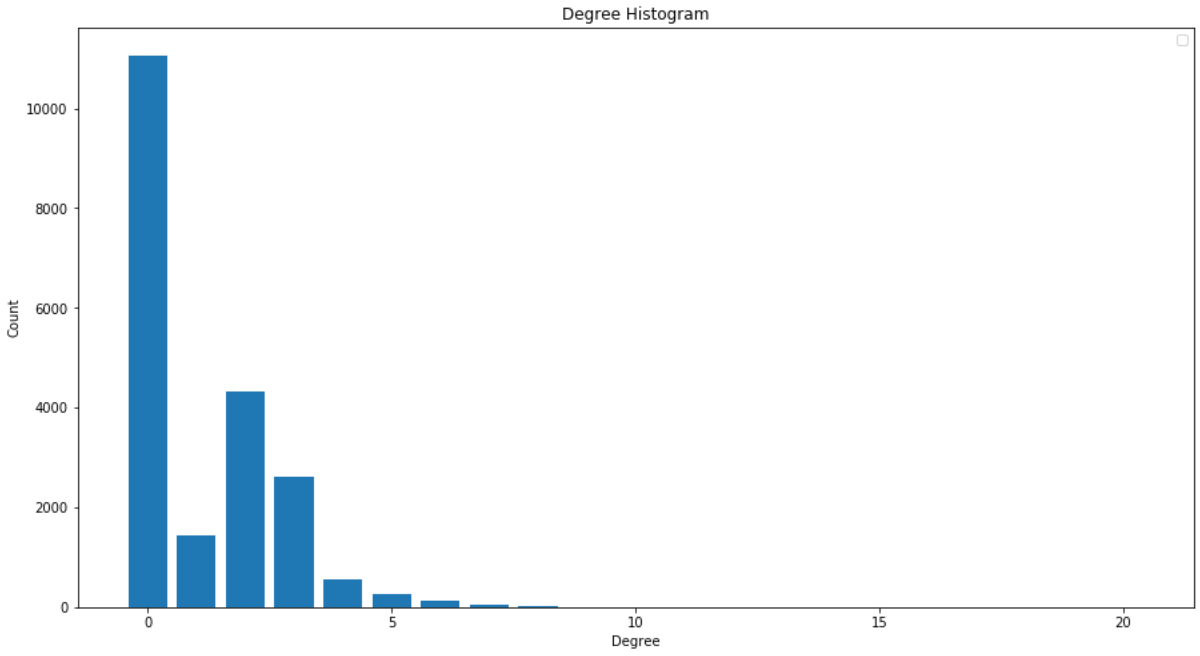
In [92]:
```
nx.average_clustering(g)
```

Out[92]:
```
0.07055493843679166
```

The plot is not clustered, so it's very hard to make meaning of the graph by simply looking at it. Perhaps making the graph based on latitude and longitude coordinates will help show the network in a more clustered and neat way.

In [86]:
```
import collections
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

plt.figure(1,figsize=(15,8))

plt.bar(np.array(list(deg)[::-1]), list(cnt)[::-1])

plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
plt.legend()


plt.show()
```

```
No handles with labels found to put in legend.
```



There aren't as many loads/generators that transport their power to multiple buses as most of the edges degrees are 0. Even then, seems like 2-3 is about what most of them supply/recieve power from.

In [88]:
```
lengths = nx.shortest_path_length(G)
st_length = {}  #this will saves the maximun path per station
ml=0
for key in lengths:   #iterates in all the lengths per station
    ll = key      #only gives an intuitive name
    i = max(ll[1].values())  #finds the maximun value of length
    st_length[ll[0]] = i  #ll[0] has the name of the station and and i t
he value
    if i > ml:
        ml = i  #this will save the overall maximun

print("The maximun length in the network is ",ml)
```

```
The maximun length in the network is  129
```

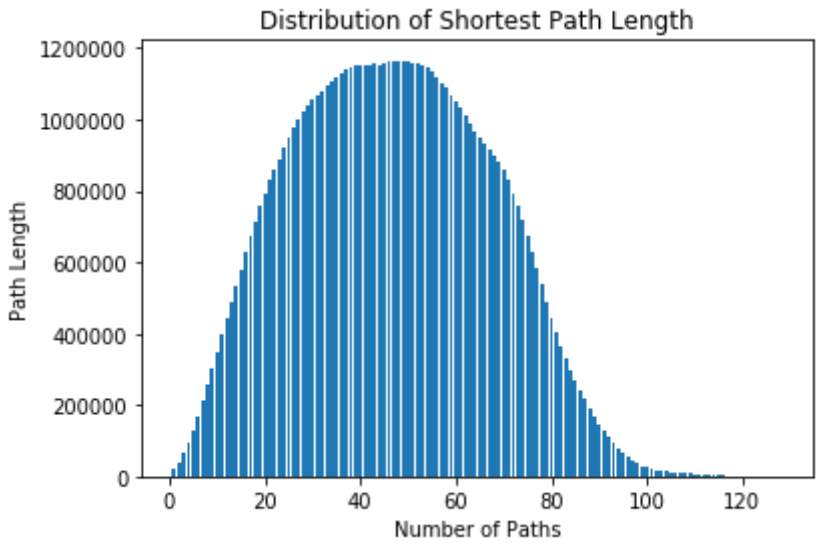
In [89]:
```
lengths = nx.shortest_path_length(G)
plengths=[]
for key in lengths:   #iterates all the kengths
    ll = key
    for i in list(ll[1].values()):    #saves a list with the lengths gre
ater than zero
        if i > 0:plengths.append(i)
```

In [90]:
```
len(plengths)
```

Out[90]:
```
70462350
```

In [200]:
```
hist4, bins4 = np.histogram(plengths, bins = range(1,130))
plt.bar(np.arange(1,129),hist4)
plt.title('Distribution of Shortest Path Length')
plt.xlabel('Number of Paths')
plt.ylabel('Path Length')
```

Out[200]:
```
Text(0, 0.5, 'Path Length')
```



In [101]:
```
nx.average_shortest_path_length(g)
```

Out[101]:
```
46.978304474760186
```

With a small clustering coefficient (0.07) and shortest path length, this network DOES have small world properties.

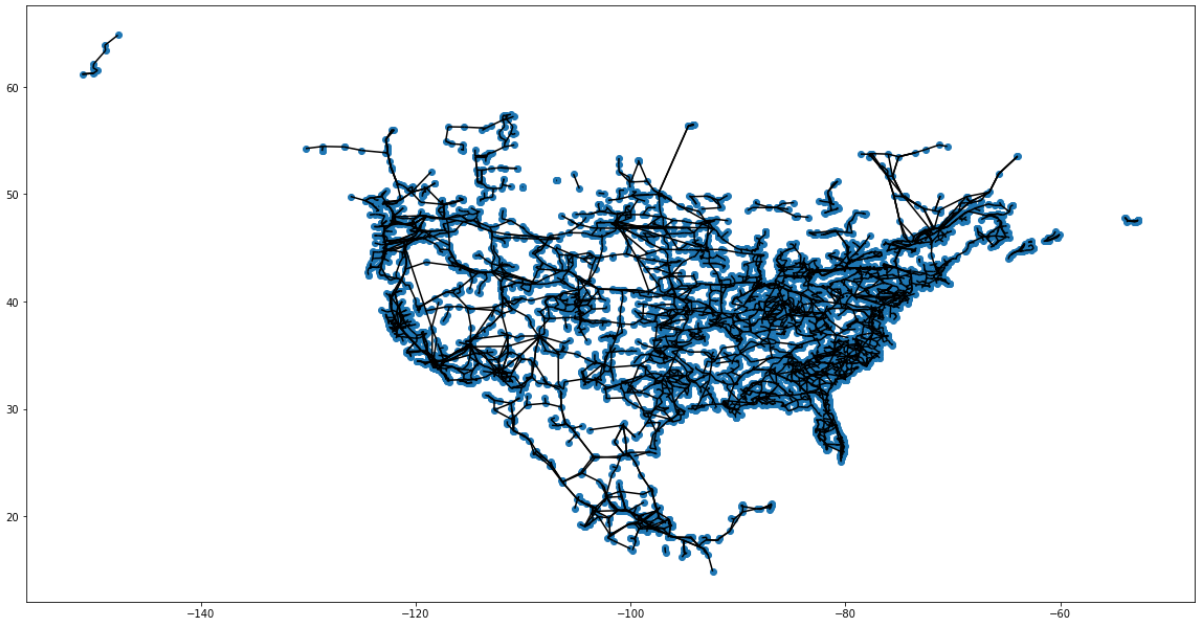
In [156]:
```
import geopandas as gpd
```

In [173]:
```
nod = pd.read_csv('pgvertices.csv')
nod.columns = ["ID", "Longitude", "Latitude","Type","Voltage", "Freguenc
y", "Name","Operator","ref","wkt_srid_4326"]

lin = pd.read_csv('pglinks.csv')
lin["voltage"] = lin["voltage"].str.split(pat = ";",expand=True)
```

In [175]:
```
from shapely import wkt

lin['geometry'] = lin['wkt_srid_4326'].str[10:].apply(wkt.loads)
my_geo_df = gpd.GeoDataFrame(lin, geometry='geometry')

nod['geometry'] = nod['wkt_srid_4326'].str[10:].apply(wkt.loads)
geonodes = gpd.GeoDataFrame(nod, geometry='geometry')
```

In [180]:
```
f, ax = plt.subplots(1, figsize=(20, 25))
my_geo_df['geometry'].plot(axes=ax,color='black')
geonodes['geometry'].plot(axes=ax)
plt.show()
```

```
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/geopanda
s/plotting.py:314: FutureWarning: 'axes' is deprecated, please use 'ax'
instead (for consistency with pandas)
  FutureWarning,
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/geopanda
s/plotting.py:314: FutureWarning: 'axes' is deprecated, please use 'ax'
instead (for consistency with pandas)
  FutureWarning,
```



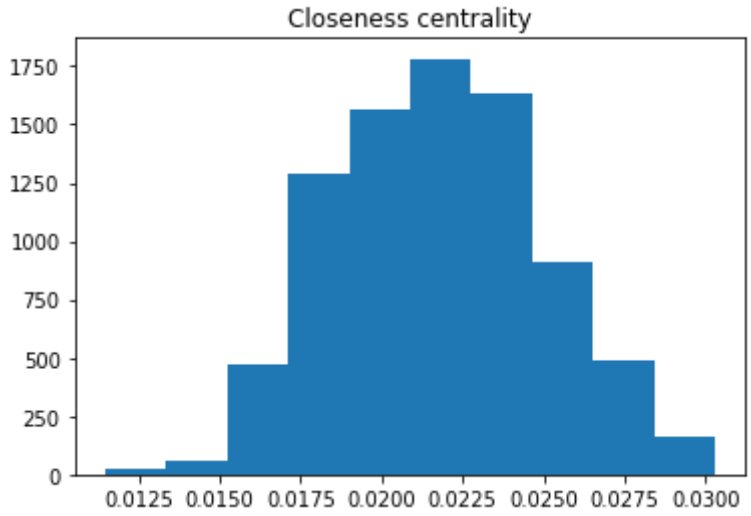
In [202]:
```
cl=nx.closeness_centrality(g)
```

In [203]:
```
dc=nx.degree_centrality(g)
```

In [204]:
```
bc=nx.betweenness_centrality(g)
```

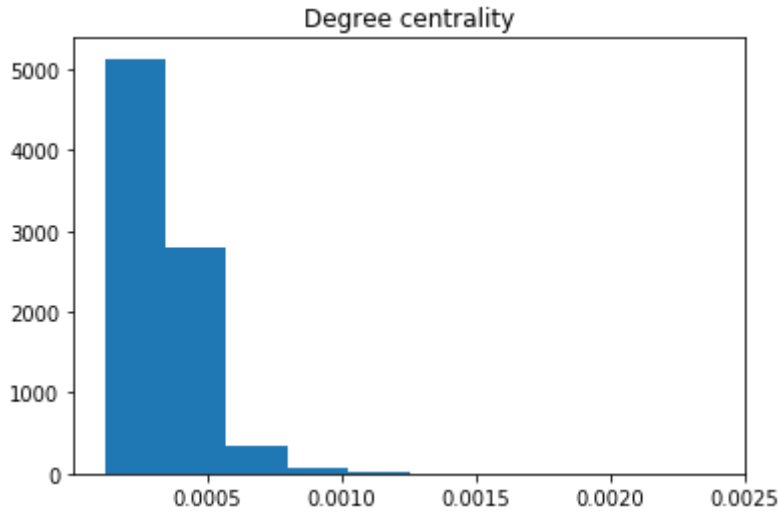
In [214]:
```
plt.hist(cl.values())
plt.title("Closeness centrality")
```

Out[214]:
```
Text(0.5, 1.0, 'Closeness centrality')
```



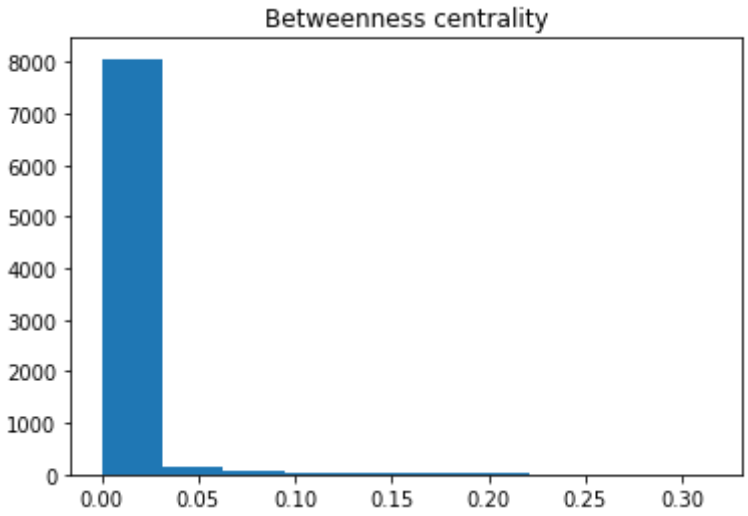
In [216]:
```
plt.hist(dc.values())
plt.title("Degree centrality")
```

Out[216]:
```
Text(0.5, 1.0, 'Degree centrality')
```



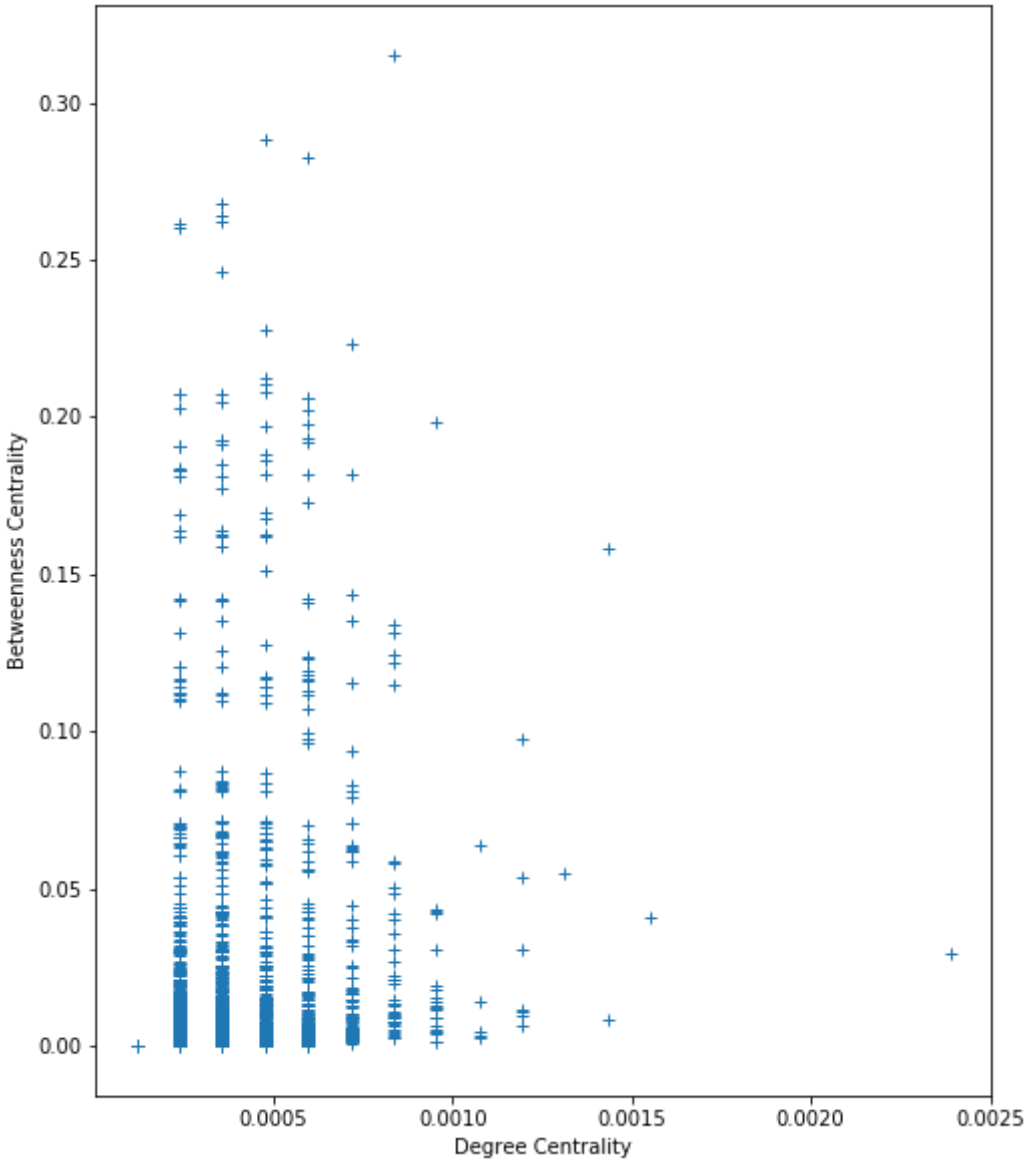
In [217]:
```
plt.hist(bc.values())
plt.title("Betweenness centrality")
```

Out[217]:
```
Text(0.5, 1.0, 'Betweenness centrality')
```



In [219]:
```
xdata=list(dc.values())
ydata=list(bc.values())
plt.figure(1,figsize=(8,10))
plt.plot(xdata,ydata,'+')
plt.xlabel('Degree Centrality')
plt.ylabel('Betweenness Centrality')
```

Out[219]: Text(0, 0.5, 'Betweenness Centrality')



In [225]:
```
sum(dc.values())/len((dc.values()))
```

Out[225]: 0.00029327449212028083

In [226]:
```
sum(bc.values())/len((bc.values()))
```

Out[226]: 0.005491258148185908

In [227]:
```
sum(cl.values())/len((cl.values()))
```

Out[227]: 0.021758318477106457

In [228]:
```
import operator
max(cl.items(), key=operator.itemgetter(1))[0]
```

Out[228]: 29489

In [229]:
```
max(dc.items(), key=operator.itemgetter(1))[0]
```

Out[229]: 1898

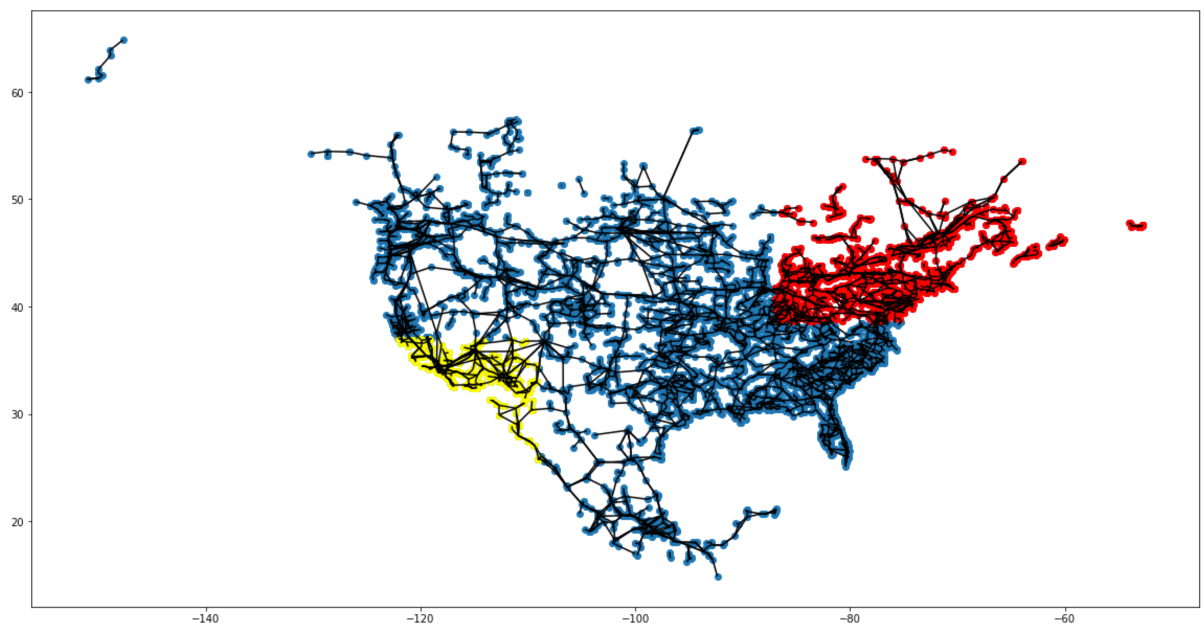
In [230]:
```
max(bc.items(), key=operator.itemgetter(1))[0]
```

Out[230]: 2465

In [240]:
```
b03 = geonodes[(geonodes['Longitude'] > -87) & (geonodes['Latitude'] > 3
8.6)]
b11 = geonodes[(geonodes['Longitude'] < -109) & (geonodes['Latitude'] <
37)]
```

In [242]:
```
f, ax = plt.subplots(1, figsize=(20, 25))
my_geo_df['geometry'].plot(axes=ax,color='black')
geonodes['geometry'].plot(axes=ax)
b03['geometry'].plot(axes=ax, color = 'red')
b11['geometry'].plot(axes=ax, color = 'yellow')
plt.show()
```

```
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/geopanda
s/plotting.py:314: FutureWarning: 'axes' is deprecated, please use 'ax'
instead (for consistency with pandas)
  FutureWarning,
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/geopanda
s/plotting.py:314: FutureWarning: 'axes' is deprecated, please use 'ax'
instead (for consistency with pandas)
  FutureWarning,
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/geopanda
s/plotting.py:314: FutureWarning: 'axes' is deprecated, please use 'ax'
instead (for consistency with pandas)
  FutureWarning,
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/geopanda
s/plotting.py:314: FutureWarning: 'axes' is deprecated, please use 'ax'
instead (for consistency with pandas)
  FutureWarning,
```



In [ ]:

In [259]:
```
b03_links = my_geo_df[(my_geo_df['v_id_1'].isin(b03['ID'].values)) | (my
_geo_df['v_id_2'].isin(b03['ID'].values))]
len(b03_links)
```

Out[259]: 6101

In [260]:
```
b11_links = my_geo_df[(my_geo_df['v_id_1'].isin(b11['ID'].values)) | (my
_geo_df['v_id_2'].isin(b11['ID'].values))]
len(b11_links)
```

Out[260]:
```
1665
```

In [ ]:
```

```

In [ ]:
```

```

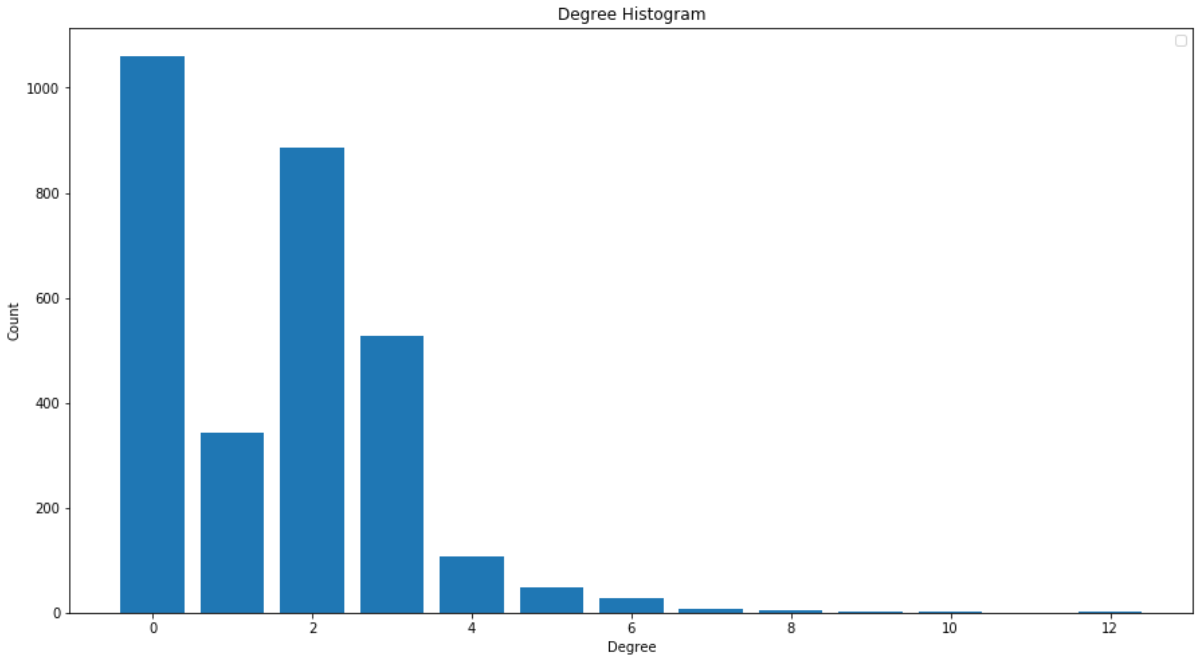
In [256]:
```
b03_network = G.subgraph(b03['ID'].values)
```

In [262]:
```
nx.average_clustering(b03_network)
```

Out[262]:
```
0.053020593739851324
```

In [264]:
```
import collections
degree_sequence = sorted([d for n, d in b03_network.degree()], reverse=T
rue)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

plt.figure(1,figsize=(15,8))

plt.bar(np.array(list(deg)[::-1]), list(cnt)[::-1])

plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
plt.legend()


plt.show()
```

```
No handles with labels found to put in legend.
```



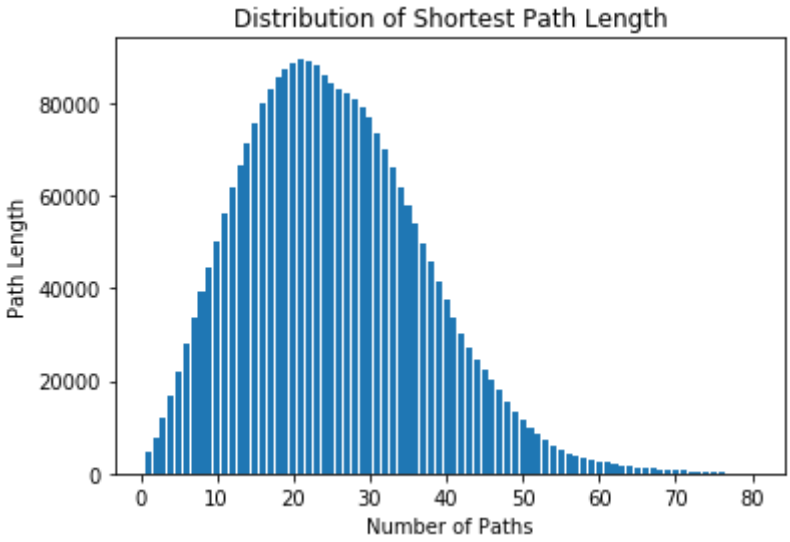
In [266]:
```
lengths = nx.shortest_path_length(b03_network)
st_length = {}  #this will saves the maximun path per station
ml=0
for key in lengths:   #iterates in all the lengths per station
    ll = key      #only gives an intuitive name
    i = max(ll[1].values())  #finds the maximun value of length
    st_length[ll[0]] = i  #ll[0] has the name of the station and and i t
he value
    if i > ml:
        ml = i  #this will save the overall maximun

print("The maximun length in the network is ",ml)
```

```
The maximun length in the network is  82
```

In [267]:
```
lengths = nx.shortest_path_length(b03_network)
plengths=[]
for key in lengths:   #iterates all the kengths
    ll = key
    for i in list(ll[1].values()):    #saves a list with the lengths gre
ater than zero
        if i > 0:plengths.append(i)
```

In [268]:
```
hist4, bins4 = np.histogram(plengths, bins = range(1,82))
plt.bar(np.arange(1,81),hist4)
plt.title('Distribution of Shortest Path Length')
plt.xlabel('Number of Paths')
plt.ylabel('Path Length')
```

Out[268]:
```
Text(0, 0.5, 'Path Length')
```

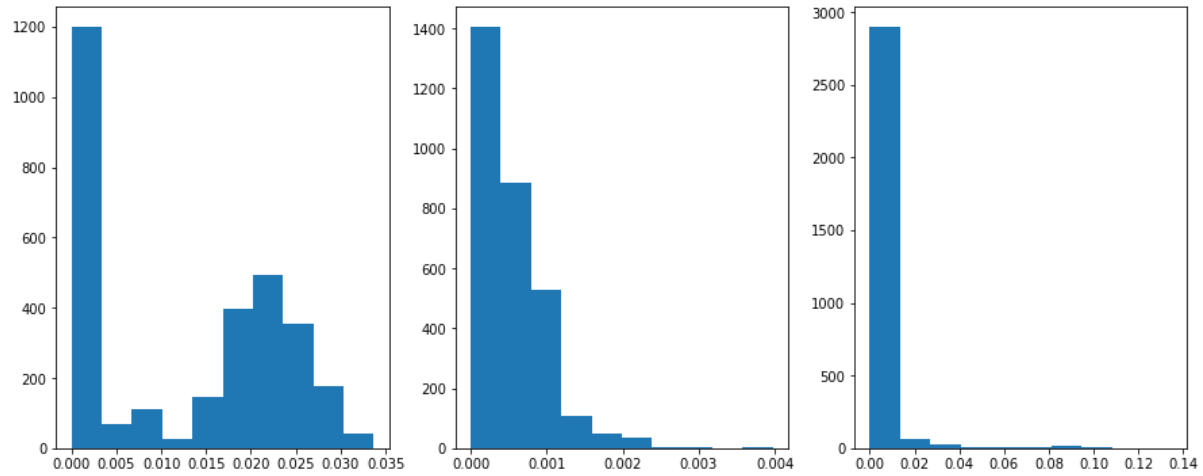


In [275]:
```
sum(plengths)/len(plengths)
```

Out[275]:
```
25.30379751523717
```

In [276]:
```
cl=nx.closeness_centrality(b03_network)
dc=nx.degree_centrality(b03_network)
bc=nx.betweenness_centrality(b03_network)
```

In [286]:
```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,6))
ax1.hist(cl.values())
ax2.hist(dc.values())
ax3.hist(bc.values())
plt.show()
```

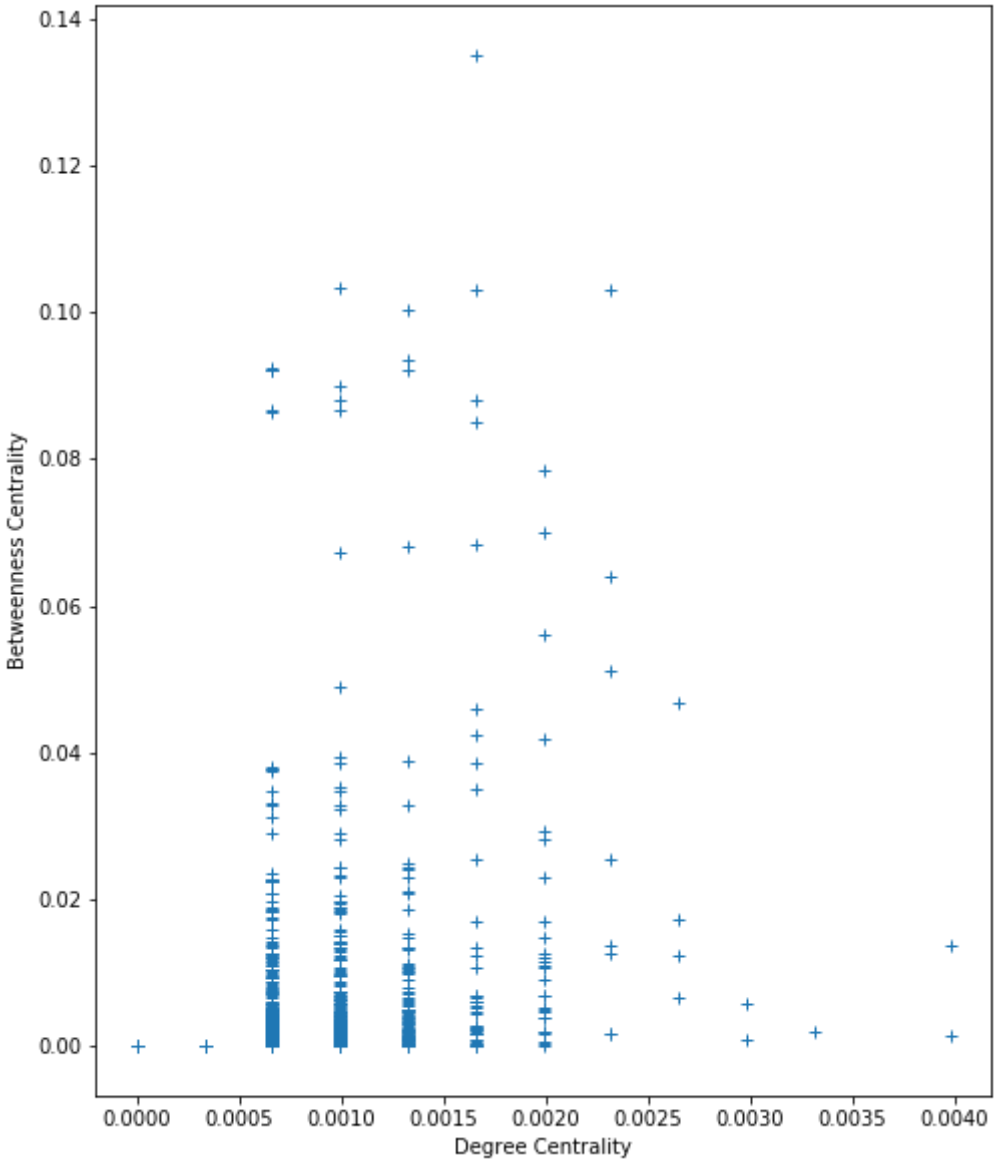


In [288]:
```
print("avg closeness centrality: " + str(sum(cl.values())/len((cl.values
())))))
print("avg degree centrality: " + str(sum(dc.values())/len((dc.values
())))))
print("avg betweenness centrality: " + str(sum(bc.values())/len((bc.valu
es())))))
```

```
avg closeness centrality: 0.01241051253522033
avg degree centrality: 0.0005132278714165242
avg betweenness centrality: 0.0023918838041878353
```

In [277]:
```
xdata=list(dc.values())
ydata=list(bc.values())
plt.figure(1,figsize=(8,10))
plt.plot(xdata,ydata,'+')
plt.xlabel('Degree Centrality')
plt.ylabel('Betweenness Centrality')
```

Out[277]: Text(0, 0.5, 'Betweenness Centrality')



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

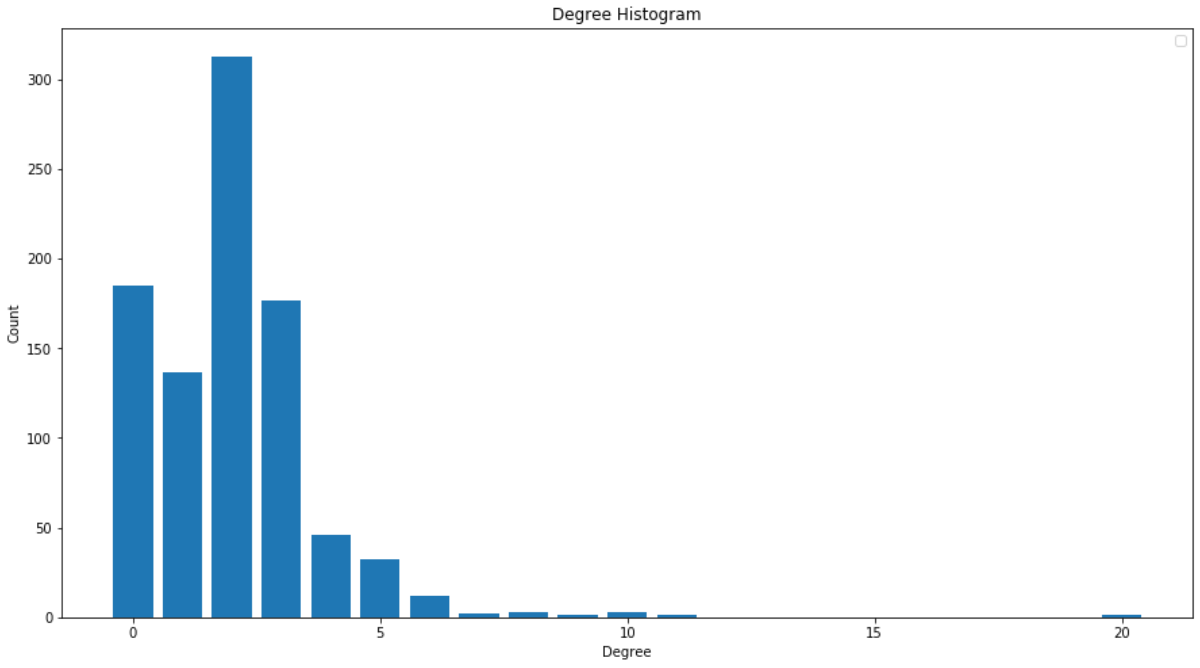
In [261]:
```
b11_network = G.subgraph(b11['ID'].values)
```

In [263]:
```
nx.average_clustering(b11_network)
```

Out[263]:
```
0.08357608232802798
```

In [265]:
```
import collections
degree_sequence = sorted([d for n, d in b11_network.degree()], reverse=T
rue)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

plt.figure(1,figsize=(15,8))

plt.bar(np.array(list(deg)[::-1]), list(cnt)[::-1])

plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
plt.legend()


plt.show()
```

```
No handles with labels found to put in legend.
```



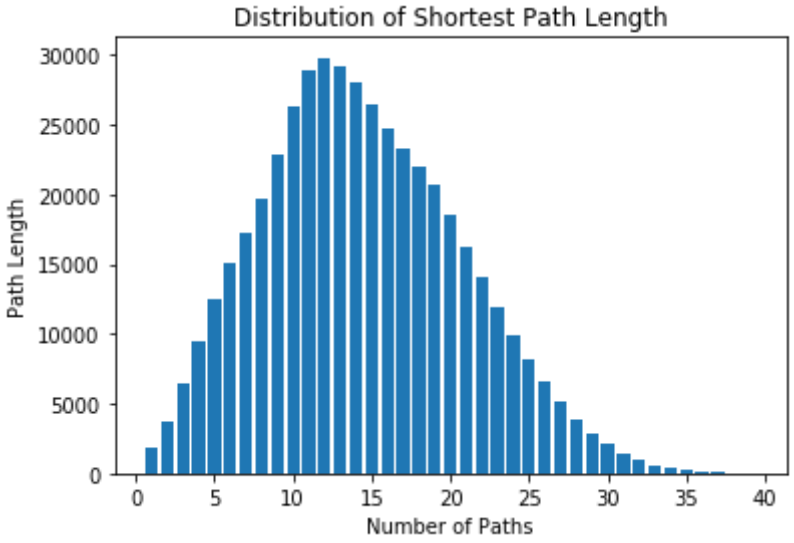
In [289]:
```
lengths = nx.shortest_path_length(b11_network)
st_length = {}  #this will saves the maximun path per station
ml=0
for key in lengths:   #iterates in all the lengths per station
    ll = key      #only gives an intuitive name
    i = max(ll[1].values())  #finds the maximun value of length
    st_length[ll[0]] = i  #ll[0] has the name of the station and and i t
he value
    if i > ml:
        ml = i  #this will save the overall maximun

print("The maximun length in the network is ",ml)
```

```
The maximun length in the network is  41
```

In [290]:
```
lengths = nx.shortest_path_length(b11_network)
plengths=[]
for key in lengths:   #iterates all the kengths
    ll = key
    for i in list(ll[1].values()):    #saves a list with the lengths gre
ater than zero
        if i > 0:plengths.append(i)
```

In [291]:
```
hist4, bins4 = np.histogram(plengths, bins = range(1,41))
plt.bar(np.arange(1,40),hist4)
plt.title('Distribution of Shortest Path Length')
plt.xlabel('Number of Paths')
plt.ylabel('Path Length')
```

Out[291]:
```
Text(0, 0.5, 'Path Length')
```

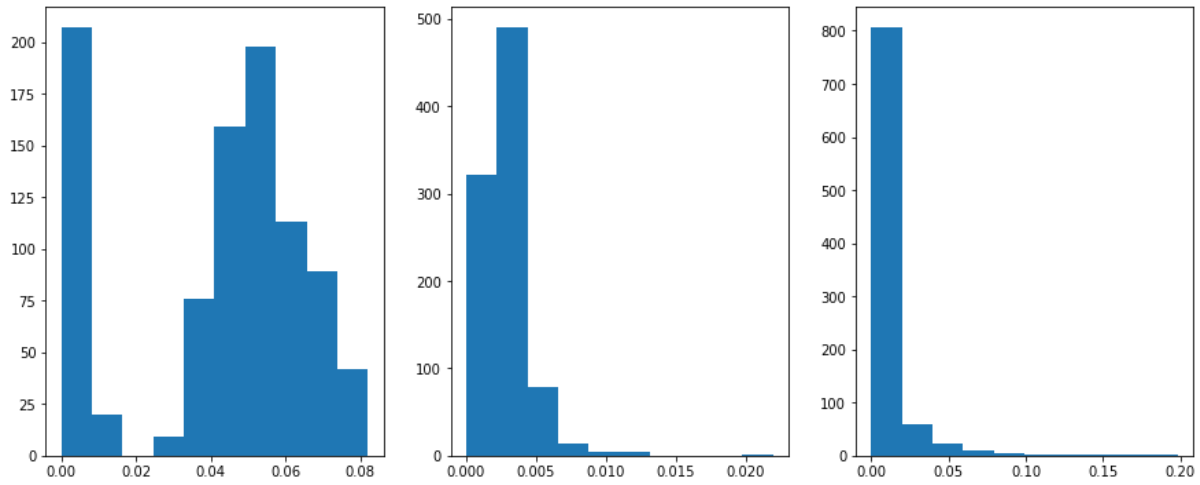


In [292]:
```
sum(plengths)/len(plengths)
```

Out[292]:
```
14.454795264104217
```

In [293]:
```
cl=nx.closeness_centrality(b11_network)
dc=nx.degree_centrality(b11_network)
bc=nx.betweenness_centrality(b11_network)
```

In [294]:
```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,6))
ax1.hist(cl.values())
ax2.hist(dc.values())
ax3.hist(bc.values())
plt.show()
```

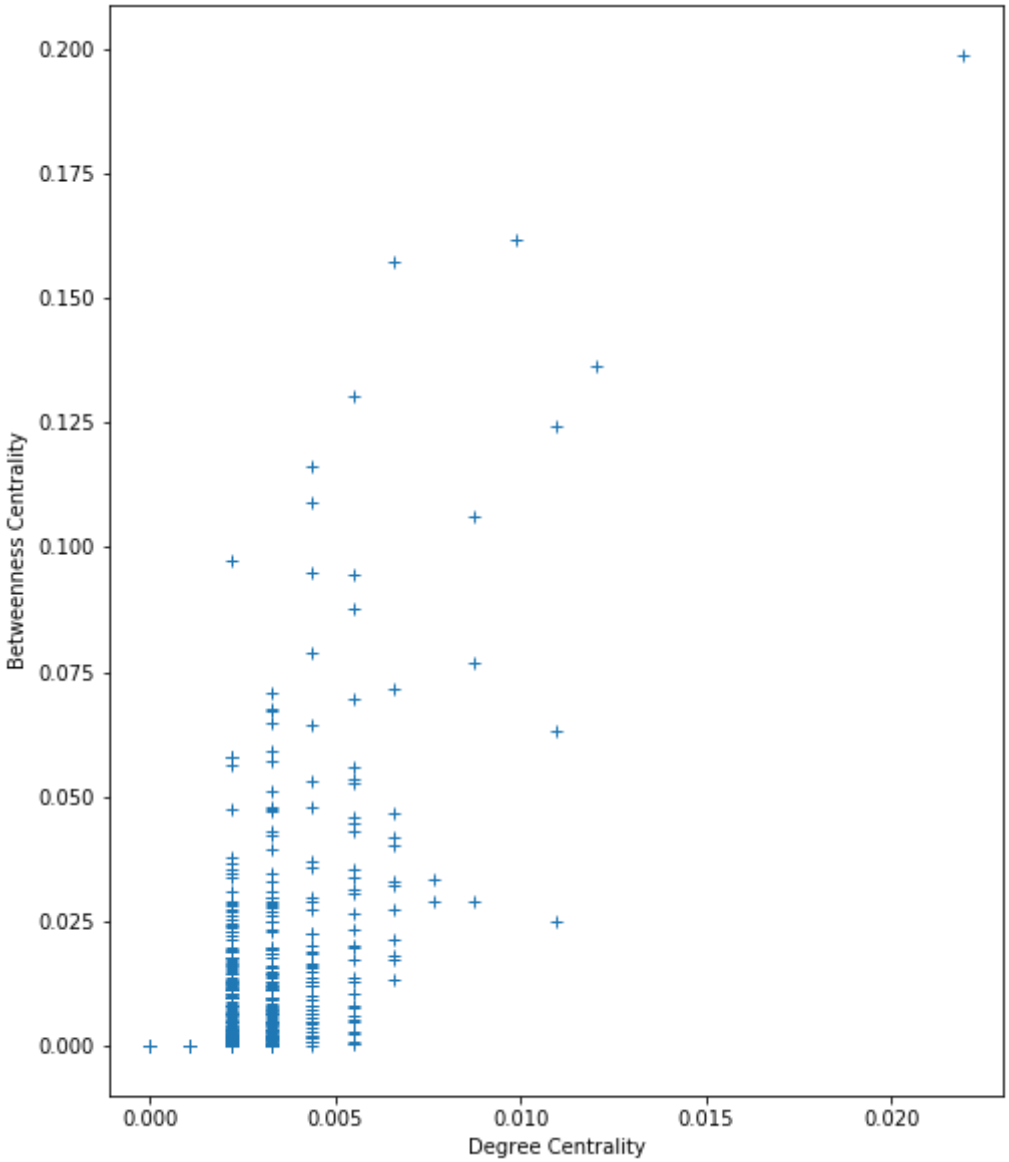


In [295]:
```
print("avg closeness centrality: " + str(sum(cl.values())/len((cl.values
())))))
print("avg degree centrality: " + str(sum(dc.values())/len((dc.values
())))))
print("avg betweenness centrality: " + str(sum(bc.values())/len((bc.valu
es())))))
```

```
avg closeness centrality: 0.04117117237939127
avg degree centrality: 0.002183374646913026
avg betweenness centrality: 0.008365585936833429
```

In [296]:
```
xdata=list(dc.values())
ydata=list(bc.values())
plt.figure(1,figsize=(8,10))
plt.plot(xdata,ydata,'+')
plt.xlabel('Degree Centrality')
plt.ylabel('Betweenness Centrality')
```

Out[296]: Text(0, 0.5, 'Betweenness Centrality')



In [297]:
```
G.number_of_nodes()
```

Out[297]: 20522

In [298]:
```
G.number_of_edges()
```

Out[298]: 11556

In [ ]:

In [306]:
```
p1 = geonodes[~((geonodes['Longitude'] > -87) & (geonodes['Latitude'] > 38.6))]
rest = p1[~((geonodes['Longitude'] < -109) & (p1['Latitude'] < 37))]
```

```
/Users/MohamedRakha/opt/anaconda3/lib/python3.7/site-packages/geopandas/geodataframe.py:576: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  result = super(GeoDataFrame, self).__getitem__(key)
```

In [308]:
```
rest_network = G.subgraph(rest['ID'].values)
```

In [309]:
```
nx.average_clustering(rest_network)
```

Out[309]:
```
0.05051521865412586
```

In [313]:
```
lengths = nx.shortest_path_length(rest_network)
plengths=[]
for key in lengths:   #iterates all the kengths
    ll = key
    for i in list(ll[1].values()):    #saves a list with the lengths greater than zero
        if i > 0:plengths.append(i)
```

In [314]:
```
sum(plengths)/len(plengths)
```

Out[314]:
```
47.99805497759653
```

In [ ]:
```

```