Enron Email Network Analysis

Analysis is based on Enron Email network data which is derived from http://www.cis.jhu.edu/~parky/Enron/ (http://www.cis.jhu.edu/~parky/Enron/). This contains a small subset of the large network. This data was originally made public, and posted to the web, by the Federal Energy Regulatory Commission during its investigation. Nodes of the network are email addresses and if an address i sent at least one email to address j, the graph contains an undirected edge from i to j.

Notebook Goal- We are trying to parse the data into clusters and find out the central person in the network based on its clusters. Its an easy task but this model can be used for variety of use cases like Citation Network, Twitter network and so on.

Further Analysis can be done based on the notebook result.

Process-

1. Explore data
2. Compute the partition of the graph nodes which maximises the modularity by using the Louvain heuristices. (Cluster Partition)
3. Connected component to check whether the nodes are connected.
4. Betweeness Centrality for a particular cluster
5. Bfs_tree based on the most central node.

In [ ]:
```
!conda install seaborn # This is for seaborn package for scatter plot using se
aborn
```

Import Environment variables needed for the notebook model

In [84]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline
```

Read the Text file into networkx graph

In [4]:
```
enron = nx.read_edgelist("actualdata.txt", create_using = nx.Graph(), nodetype
 = int)
```

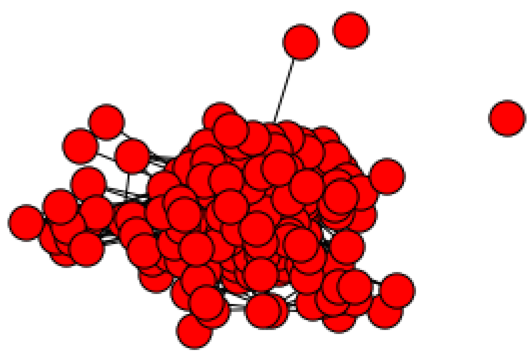
Print and check the data (no. of nodes and edges)

In [5]:
```
print (nx.info(enron))
```

```
Name:
Type: Graph
Number of nodes: 184
Number of edges: 2216
Average degree:  24.0870
```

In [6]:
```
sp= nx.spring_layout(enron) # Using the spring Layout in networkx
```

In [7]:
```
plt.axis("off")
nx.draw_networkx(enron,pos = sp, with_labels = False)
```

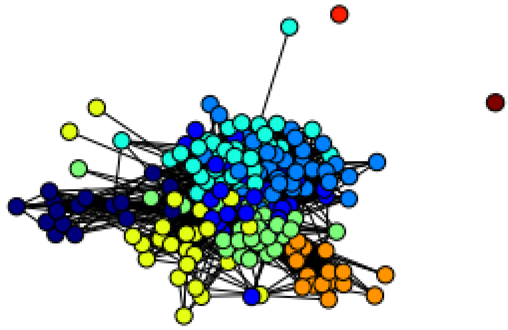


Community Detection (Cluster Partition)

In [9]:
```
import community
```

In [10]:
```
parts = community.best_partition(enron)
values = [parts.get(node) for node in enron.nodes()]
```

In [11]:
```
plt.axis("off")
x = nx.draw_networkx(enron, pos = sp, cmap = plt.get_cmap("jet"), node_color =
 values, node_size = 70, with_labels = False)
```



In [12]:
```
mod = community.modularity(parts,enron)
print("modularity:", mod)
```

```
modularity: 0.375837260846616
```

This means that the network is not a dense network. Higher the modularity denser the network.

In [16]:
```
print(parts) # Lets print the result of the partition
```

```
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 4, 6: 3, 7: 5, 8: 6, 9: 0, 10: 2, 11: 1, 1
2: 4, 13: 0, 14: 2, 15: 1, 16: 3, 17: 2, 18: 5, 19: 3, 20: 0, 21: 5, 22: 1, 2
3: 6, 24: 3, 25: 6, 26: 3, 27: 4, 28: 5, 29: 1, 30: 0, 31: 6, 32: 5, 33: 6, 3
4: 4, 35: 5, 36: 2, 37: 4, 38: 1, 39: 3, 40: 5, 41: 2, 42: 4, 43: 2, 44: 5, 4
5: 2, 46: 2, 47: 3, 48: 0, 49: 2, 50: 1, 51: 4, 52: 5, 53: 4, 54: 2, 55: 0, 5
6: 2, 57: 4, 58: 4, 59: 3, 60: 1, 61: 3, 62: 3, 63: 4, 64: 5, 65: 1, 66: 4, 6
7: 4, 68: 1, 69: 4, 70: 1, 71: 7, 72: 4, 73: 4, 74: 4, 75: 1, 76: 5, 77: 5, 7
8: 2, 79: 0, 80: 2, 81: 2, 82: 2, 83: 4, 84: 2, 85: 5, 86: 5, 87: 5, 88: 2, 8
9: 3, 90: 3, 91: 0, 92: 1, 93: 3, 94: 4, 95: 6, 96: 1, 97: 2, 98: 6, 99: 3, 1
00: 5, 101: 2, 102: 3, 103: 6, 104: 0, 105: 4, 106: 2, 107: 4, 108: 6, 109:
 4, 110: 1, 111: 1, 112: 1, 113: 6, 114: 1, 115: 3, 116: 2, 117: 8, 118: 3, 1
19: 0, 120: 3, 121: 5, 122: 6, 123: 4, 124: 6, 125: 5, 126: 3, 127: 2, 128:
 4, 129: 0, 130: 2, 131: 0, 132: 3, 133: 5, 134: 3, 135: 3, 136: 3, 137: 2, 1
38: 5, 139: 2, 140: 3, 141: 3, 142: 0, 143: 3, 144: 4, 145: 1, 146: 4, 147:
 4, 148: 0, 149: 5, 150: 3, 151: 6, 152: 0, 153: 5, 154: 2, 155: 1, 156: 2, 1
57: 2, 158: 6, 159: 2, 160: 1, 161: 6, 162: 1, 163: 4, 164: 3, 165: 1, 166:
 2, 167: 5, 168: 1, 169: 1, 170: 6, 171: 3, 172: 1, 173: 1, 174: 3, 175: 2, 1
76: 3, 177: 6, 178: 4, 179: 2, 180: 2, 181: 4, 182: 2, 183: 5}
```

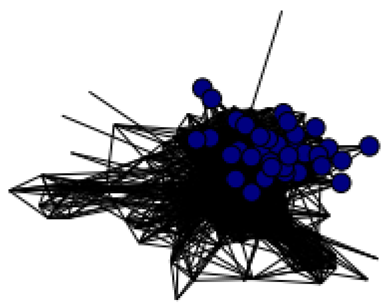
As we can see, each key value of node is given a partition value. Suppose now we only want the partition value 2:

In [27]:
```
bparts = {k:v for (k, v) in parts.items() if v == 2}
print(bparts)
```

```
{2: 2, 182: 2, 137: 2, 10: 2, 139: 2, 130: 2, 78: 2, 80: 2, 81: 2, 82: 2, 84:
 2, 14: 2, 88: 2, 154: 2, 156: 2, 157: 2, 159: 2, 97: 2, 36: 2, 101: 2, 166:
 2, 17: 2, 41: 2, 106: 2, 43: 2, 45: 2, 46: 2, 175: 2, 49: 2, 179: 2, 116: 2,
 54: 2, 56: 2, 180: 2, 127: 2}
```

In [28]:
```
values1 = [bparts.get(node) for node in enron.nodes()]
```

In [29]:
```
plt.axis("off")
x = nx.draw_networkx(enron, pos = sp, cmap = plt.get_cmap("jet"), node_color =
 values1, node_size = 90, with_labels = False)
```



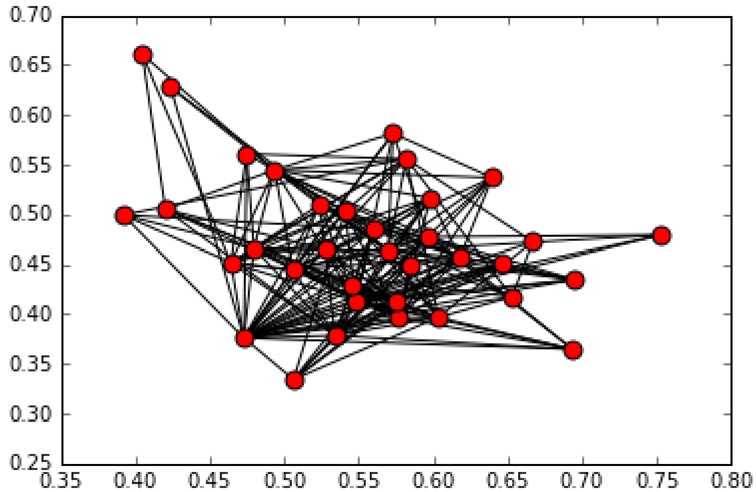
Shows only the network of patition Value 2

In [86]:
```
key = {k for (k, v) in parts.items() if v == 2} # Extracting the Keys from the
 cluster
print(key)
```

```
{2, 130, 137, 10, 139, 14, 17, 154, 156, 157, 159, 36, 166, 41, 43, 45, 46, 1
75, 49, 179, 180, 54, 182, 56, 78, 80, 81, 82, 84, 88, 97, 101, 106, 116, 12
7}
```

In [87]:
```
sub_enron = enron.subgraph(key) # Making a subgraph out of the keys of the par
tition
```

In [32]:
```
subgraph = nx.draw_networkx(sub_enron, pos = sp,node_size = 80, with_labels = False)
```



This is the Sub graph of the cluster 2. We will be focusing on this for finding the most cenral node

In [34]:
```
sorted(nx.connected_components(sub_enron), key = len, reverse=True) # Check an
d grouping the conncetd component.
```

Out[34]:
```
[{2,
  10,
  14,
  17,
  36,
  41,
  43,
  45,
  46,
  49,
  54,
  56,
  78,
  80,
  81,
  82,
  84,
  88,
  97,
  101,
  106,
  116,
  127,
  130,
  137,
  139,
  154,
  156,
  157,
  159,
  166,
  175,
  179,
  180,
  182}]
```

This network sub graph has only 1 connceted component. Now Lets check the betweeness centrality for this sub graph

In [89]:
```
nx.betweenness_centrality(sub_enron)
```

Out[89]:
```
{2: 0.014589723012182905,
 10: 0.026366056463708168,
 14: 0.0009358288770053476,
 17: 0.04551603798957356,
 36: 0.00695494592553416,
 41: 0.01253570622869144З,
 43: 0.01604348810231163,
 45: 0.0020209093738505503,
 46: 0.001502419149477973,
 49: 0.026581145258744176,
 54: 0.0,
 56: 0.027853882599871895,
 78: 0.03954197562250409,
 80: 0.03530459006887197,
 81: 0.0033712474888945476,
 82: 0.2136933794551038,
 84: 0.021612357727761808,
 88: 0.0009602787096393234,
 97: 0.014902244449408582,
 101: 0.012037403807614564,
 106: 0.0006111535523300229,
 116: 0.017223449211906064,
 127: 0.004620923738570796,
 130: 0.0631339060360355,
 137: 0.0,
 139: 0.0021475256769374414,
 154: 0.0007299889652830828,
 156: 0.0028987352516764275,
 157: 0.07116726001458692,
 159: 0.036583646373093415,
 166: 0.0,
 175: 0.003932345368331528,
 179: 0.0038982902619266254,
 180: 0.0032221524374512053,
 182: 0.00012732365673542143}
```

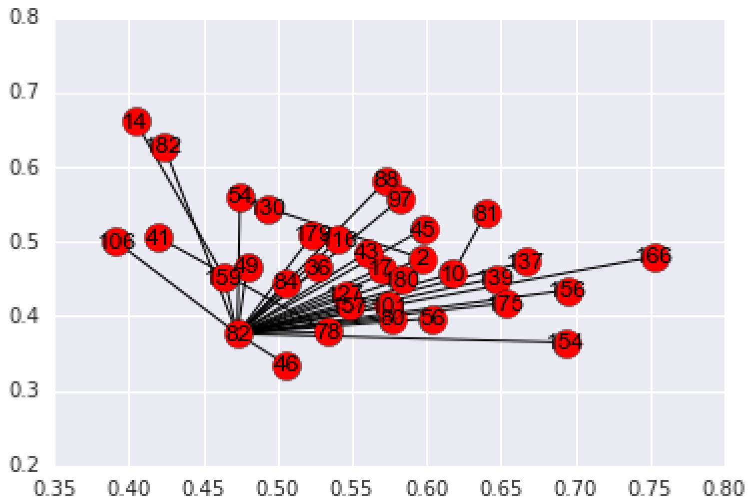
As you can see the 82 has the maximum betweeness centrality. That means it acts as the bridge in the network. WE will now try to apply the breadth first serach tree algorithm taking 82 as the starting point

In [62]:
```
bfs_tree = list(nx.bfs_edges(sub_enron,82))
print(bfs_tree)
```

```
[(82, 2), (82, 54), (82, 137), (82, 10), (82, 139), (82, 78), (82, 80), (82,
 17), (82, 84), (82, 88), (82, 154), (82, 156), (82, 157), (82, 159), (82, 9
7), (82, 36), (82, 101), (82, 166), (82, 106), (82, 43), (82, 45), (82, 46),
 (82, 175), (82, 49), (82, 179), (82, 180), (82, 182), (82, 56), (82, 116),
 (82, 127), (2, 130), (10, 81), (78, 41), (159, 14)]
```

In [63]:
```
bfs = nx.Graph(bfs_tree)
```

In [92]:
```
nx.draw_networkx(bfs, pos = sp,node_size = 200, with_labels = True) #Plot the
 BFS tree
```



From the above graph we can makeout that there is some kind of a heirarchy in the network. We can even find out who was used as a cc or bcc in this email network. For Example, in above graph - look at the link 82-78-154.. We can say that 78 sent some data to 154 keeping 82 as cc. Since its an undirected graph its hard to make out but we can surely get some major points from this to understand the network.

Lets try to run hits

In [93]:
```
hits = nx.hits(sub_enron, max_iter=100, tol=1e-08, nstart=None, normalized=True) # cannot make out the output. Please give your inputs.
print(hits)
```

```
({2: 0.031817349869161154, 182: 0.00954208834280252, 137: 0.009823224541016784, 10: 0.046357776911443216, 139: 0.024999342277107577, 130: 0.037713614400224325, 14: 0.006238644266323655, 80: 0.04416051260379766, 17: 0.041601242340441054, 82: 0.061999138232737214, 84: 0.04835227853275319, 78: 0.046357230308187995, 88: 0.02005738881940745, 154: 0.0230484998760797, 156: 0.02092054283226148, 157: 0.052792817851167434, 159: 0.033334948246978316, 97: 0.02273976834448666, 36: 0.022172326273056923, 101: 0.028083744896090976, 166: 0.012708731920856248, 81: 0.022764378463125422, 41: 0.015104281490668992, 106: 0.01370670538815923, 43: 0.034291897959348686, 45: 0.032366122803062336, 46: 0.022381853656058555, 175: 0.01611087282942599, 49: 0.03691762394416883, 179: 0.028789487056251443, 180: 0.023330097064370867, 54: 0.013013989296203003, 56: 0.03480449044539484, 116: 0.03773979261242065, 127: 0.02385719530495966}, {2: 0.03181734986836458, 182: 0.00954208834402947, 137: 0.009823224542817316, 10: 0.04635777690908403, 139: 0.024999342274258395, 130: 0.03771361440387287, 14: 0.006238644267460254, 80: 0.044160512599597995, 17: 0.04160124234369831, 82: 0.0619991382330177, 84: 0.04835227853378155, 78: 0.04635723030405851, 88: 0.020057388820366482, 154: 0.023048499873518626, 156: 0.020920542830199423, 157: 0.052792817849430934, 159: 0.03333494825153892, 97: 0.022739768349418384, 36: 0.022172326269623233, 101: 0.02808374489138619, 166: 0.012708731921028045, 81: 0.02276437846297017, 41: 0.015104281487725845, 106: 0.013706705391168044, 43: 0.03429189796037765, 45: 0.03236612280346297, 46: 0.02238185365416197, 175: 0.016110872832268947, 49: 0.03691762394715635, 179: 0.028789487056402108, 180: 0.023330097062721256, 54: 0.013013989298942473, 56: 0.03480449044542084, 116: 0.03773979261484644, 127: 0.023857195301823803})
```

In [ ]: