Importing the required libraries

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        %matplotlib inline
        import sqlite3 as sq3
        import sqlalchemy
        import sqlalchemy.orm as orm
        import datetime, dateutil
        from collections import Counter
        from ipaddlib.ipaddlib import ipv4, ipv6
        import ipaddlib as ipa
        import ipgeotools
        import ipgeotools.ipgeotools as ipt
        import ipgeotools.ipwhois as ipw
        import ipwhois
```

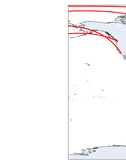
Reading the data from the .csv file

Displaying the data of the network in the dataframe

```
In [2]: in_frame_names = {
            'type':int,
            'ipver':int,
            'protocol':int,
            'ttl':int,
            'ihl':int,
            'tos':int,
            'total_length':int,
            'id':int,
            'flags':int,
            'frag_off':int,
            'checksum':int,
            'ip_offset':int,
            'ip_ttl':int,
            'ip_proto':int,
            'ip_length':int,
            'ip_ihl':int,
            'ip_tos':int,
            'ip_id':int,
            'ip_flags':int,
            'ip_frag_off':int,
            'ip_checksum':int,
            'tcp_seqnum':int,
            'tcp_acknum':int,
            'tcp_offset':int,
            'tcp_flags':int,
            'tcp_window':int,
            'tcp_urgptr':int,
            'tcp_length':int,
            'tcp_chksum':int,
            'tcp_opt':str,
            'tcp_payload':str,
            'ts':int,
            'ts_diff':int,
            'ts_delta':int,
            'ts_rel':int,
            'ts_abs':int,
            'ip_saddr':str,
            'ip_daddr':str,
            'ip_saddrtyp':int,
            'ip_daddrtyp':int,
            'ip_saddrcnt':int,
            'ip_daddrcnt':int,
            'ip_saddrasn':int,
            'ip_daddrasn':int,
            'ip_saddrprv':int,
            'ip_daddrprv':int,
            'ip_saddrcnt_prv':int,
            'ip_daddrcnt_prv':int,
            'ip_saddrasn_prv':int,
            'ip_daddrasn_prv':int,
            'ip_saddrcnt_ttl':int,
            'ip_daddrcnt_ttl':int,
            'ip_saddrasn_ttl':int,
            'ip_daddrasn_ttl':int,
            'ip_saddrprv_ttl':int,
            'ip_daddrprv_ttl':int,
            'ip_saddrcnt_prv_ttl':int,
            'ip_daddrcnt_prv_ttl':int,
            'ip_saddrasn_prv_ttl':int,
            'ip_daddrasn_prv_ttl':int,
            'ip_saddrcnt_ttl_prv':int,
            'ip_daddrcnt_ttl_prv':int,
            'ip_saddrasn_ttl_prv':int,
            'ip_daddrasn_ttl_prv':int,
            'ip_saddrprv_ttl_prv':int,
            'ip_daddrprv_ttl_prv':int,
            'ip_saddrcnt_prv_ttl_prv':int,
            'ip_daddrcnt_prv_ttl_prv':int,
            'ip_saddrasn_prv_ttl_prv':int,
            'ip_daddrasn_prv_ttl_prv':int,
            'ip_saddrcnt_prv':int,
            'ip_daddrcnt_prv':int,
            'ip_saddrasn_prv':int,
            'ip_daddrasn_prv':int,
            'ip_saddrprv_prv':int,
            'ip_daddrprv_prv':int,
            'ip_saddrcnt_ttl_prv':int,
            'ip_daddrcnt_ttl_prv':int,
            'ip_saddrasn_ttl_prv':int,
            'ip_daddrasn_ttl_prv':int,
            'ip_saddrprv_ttl_prv':int,
            'ip_daddrprv_ttl_prv':int,
            'ip_saddrcnt_prv_ttl_prv':int,
            'ip_daddrcnt_prv_ttl_prv':int,
            'ip_saddrasn_prv_ttl_prv':int,
            'ip_daddrasn_prv_ttl
```

9	203.80.214	97.210.52.21	1.2400	315800	1618400	TCP

In [101]:
```
fig = go.Figure()

fig.add_trace(go.Scattergeo:
    lon = top_10_pairs_location['lon'].append(top_10_pairs_location['lon2'],
    lat = top_10_pairs_location['lat'].append(top_10_pairs_location['lat2'],
    marker = dict:
        size = 2,
        color = 'rgb(255, 0, 0)',
        line = dict:
            width = 3,
            color = 'rgba(68, 68, 68, 0)'
    )
)

lons = []
lats = []
import numpy as np
lons = np.empty(2 * top_10_pairs_location.shape[0])
lats = np.empty(2 * top_10_pairs_location.shape[0])
lons[::2] = top_10_pairs_location['lon']
lats[::2] = top_10_pairs_location['lat']
lons[:1:2] = None
lons[:1:2] = top_10_pairs_location.shape[0]
lats[::2] = top_10_pairs_location['lat2']
lats[:1:2] = top_10_pairs_location['lat2']
lats[:1:2] = None

fig.add_trace:
    go.Scattergeo:
        lon = lons,
        lat = lats,
        mode = 'lines',
        line = dict:width = 1,color = 'red',
        opacity = 1
)

fig.update_layout:
    showlegend = False,
    geo = go.layout.Geo:
        showland = True,
    ),
    height=700,
)

fig.update_layout:(
    geo:
        projection:: 50
    )
)

fig.show:
```



As you can see, the top 10 pairs mainly connect Singapore with USA and UK

Top 5% Network Graph

In this part, we will inspect the top 5% of the interaction of the talker-listener directional pairs based on the number of packets send between the pair in the given sample of data.

In [102]:
```
t_1_pairs = data.groupby:['src_IP','dst_IP'].size:).reset_index(name:' # of Packets')
t_1_pairs = t_1_pairs.sort_values:[' # of Packets'], ascending=False:
t_1_pairs
```

Out[102]:

	src_IP	dst_IP	# of Packets
4236	193.62.192.8	137.132.228.15	3041
975	130.14.250.11	103.37.198.100	2599
1607	14.139.196.94	192.101.107.153	2368
1639	140.112.8.139	103.21.126.2	2056
1297	137.132.228.15	193.62.192.8	1910
...	...	...	...
2748	155.69.193.14	172.217.27.1	1
2747	155.69.193.108	172.217.27.3	1
2745	155.69.193.115	74.125.130.188	1
2744	155.69.193.112	74.125.68.99	1
6588	fe80:0000:0000:0000:b2a5:8a51:ca76:8776	fe80:0000:0000:0000:c4e:24ff:fe6c:8314	1

6589 rows × 3 columns

In [103]:
```
persons_5 = int(t_1_pairs.shape[0]/20)
```

In [104]:
```
top_5p_pairs = t_1_pairs.sort_values:[' # of Packets'], ascending=False:
top_5p_pairs = top_5p_pairs.head(persons_5):
top_5p_pairs
```

Out[104]:

	src_IP	dst_IP	# of Packets
4236	193.62.192.8	137.132.228.15	3041
975	130.14.250.11	103.37.198.100	2599
1607	14.139.196.94	192.101.107.153	2368
1639	140.112.8.139	103.21.126.2	2056
1297	137.132.228.15	193.62.192.8	1910
...	...	...	...
4250	185.231.231.157	140.110.112.99	26
1853	140.116.168.246	203.80.21.41	26
1057	130.246.176.41	146.115.32.42	26
222	193.6.192.11	140.111.1.2	26
1712	137.132.232.202	158.246.2.202	26

329 rows × 3 columns

In [105]:
```
pairs_5 = [tuple(row) for row in top_5p_pairs.to_numpy()]
len(top_5p_pairs)
```

Out[105]:
```
329
```

In [106]:
```
network_graph = igraph.Graph.TupleList:
    pairs_5,
    directed=True,
    weights=True,
)

network_graph.vs['size'] = 15
network_graph.vs['label_size'] = 0
network_graph.vs['label'] = network_graph.vs['name']
network_graph.vs['color'] = 'powderblue'

network_graph.es['width'] = (
    maximum.log(weight, 5.1)
    for weight in network_graph.es['weight']
)
network_graph.es['arrow_size'] = 1

network_graph.es['color'] = 'dodgerblue'


layout = network_graph.layout:'fr':
igraph.plot(network_graph, layout=layout, bbox=(2560, 1440)):
igraph.plot(network_graph, 'computer_network5.png', layout=layout, bbox=(2560, 1440)):
Image('computer_network5.png')
```

Out[106]:

In the graph above each node represents each IP address of the network nodes in the top 5% of the sample data. The directed edges represent packets' paths.

Top 1% Network Graph

In this part, we will inspect the top 1% of the interaction of the talker-listener directional pairs based on the number of packets send between the pair in the given sample of data.

In [107]:
```
persons_1 = int(t_1_pairs.shape[0]/100)
```

In [108]:
```
top_1p_pairs = t_1_pairs.sort_values:[' # of Packets'], ascending=False:
top_1p_pairs = top_1p_pairs.head(persons_1):
top_1p_pairs
```

Out[108]:

	src_IP	dst_IP	# of Packets
4236	193.62.192.8	137.132.228.15	3041
975	130.14.250.11	103.37.198.100	2599
1607	14.139.196.94	192.101.107.153	2368
1639	140.112.8.139	103.21.126.2	2056
1297	137.132.228.15	193.62.192.8	1910
...	...	...	...
1667	140.138.144.170	192.248.87.225	186
4236	194.68.32.16	202.158.207.20	186
4213	193.49.97.157	137.132.2.10	183
1172	134.219.229.225	140.113.55.80	179
1108	133.39.224.12	137.132.243.142	177

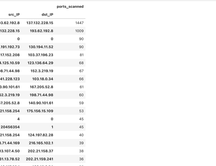
65 rows × 3 columns

In [109]:
```
pairs_1 = [tuple(row) for row in top_1p_pairs.to_numpy()]
len(top_1p_pairs)
```

Out[109]:
```
65
```

In [110]:
```
network_graph = igraph.Graph.TupleList:
    pairs_1,
    directed=True,
    weights=True,
)

network_graph.vs['size'] = 15
network_graph.vs['label_size'] = 0
network_graph.vs['label'] = network_graph.vs['name']
network_graph.vs['color'] = 'powderblue'

network_graph.es['width'] = (
    maximum.log(weight, 5.1)
    for weight in network_graph.es['weight']
)
network_graph.es['arrow_size'] = 1

network_graph.es['color'] = 'dodgerblue'


# plot graph

layout = network_graph.layout:'fr':
igraph.plot(network_graph, layout=layout, bbox=(2560, 1440)):
igraph.plot(network_graph, 'computer_network1.png', layout=layout, bbox=(2560, 1440)):
Image('computer_network1.png')
```

Out[110]:



The above graph represents the interactions between the nodes in the top 1% of the most popular talker-listener pairs.

Port Scanning

This part will focus on whether a node is being port scanned by another node.

In [111]:
```
port_scanning = pd.DataFrame:data.groupby:['src_IP','dst_IP'].['dst_port'].nunique():
port_scanning = port_scanning.reset_index():
port_scanning.sort_values:['port_scanned'], ascending=False:.head(20)
```

Out[111]:

	src_IP	dst_IP	port_scanned
4	193.62.192.8	137.132.228.15	1447
137.132.228.15	**193.62.192.8**	**1009**	
0	**0**	**90**	
59.191.192.7	**150.164.15.52**	**90**	
203.117.152.208	**103.37.198.23**	**81**	
76.125.10.59	**133.130.64.29**	**60**	
166.71.44.58	**192.3.219.10**	**57**	
207.241.226.123	**163.16.8.34**	**56**	
140.90.101.61	**167.205.32.8**	**51**	
192.3.219.10	**166.71.44.58**	**50**	
167.205.32.8	**140.90.101.61**	**50**	
202.21.158.254	**175.106.15.109**	**53**	
4	**0**	**45**	
204.60.254	**1**	**45**	
202.21.158.254	**134.197.62.20**	**42**	
166.71.44.163	**216.165.163.1**	**39**	
13.107.4.50	**202.21.158.37**	**36**	
31.13.76.52	**202.21.158.241**	**36**	
143.117.65.1	**163.16.8.34**	**36**	
184.224.137.58	**165.237.142.250**	**31**	

We group the entries by their source and destination IP address. In this way, we can create a new column to to count the unique destination ports. This will indicate if a talker communicates with many ports of a listener.

In [112]:
```
sus_data = data[(data['src_IP'] == '193.62.192.8') & (data['dst_IP'] == '137.132.228.15')] [data['src_IP'] == '137.132.228.15'] & (data['dst_IP'] == '193.62.192.8')]
sus_data['src_IP', 'dst_IP', 'dst_port']
```

Out[112]:

	src_IP	dst_IP	dst_port
4	137.132.228.15	193.62.192.8	34282
9	193.62.192.8	137.132.228.15	53923
12	193.62.192.8	137.132.228.15	53923
19	137.132.228.15	193.62.192.8	20480
38	193.62.192.8	137.132.228.15	53923
...	...	...	...
69285	193.62.192.8	137.132.228.15	34753
69501	137.132.228.15	193.62.192.8	3123
69536	137.132.228.15	193.62.192.8	49413
69542	193.62.192.8	137.132.228.15	57296
69567	137.132.228.15	193.62.192.8	3123

4951 rows × 3 columns

In [113]:
```
scanner_data = pd.DataFrame:
scanner_data['connection'] = sus_data['src_IP'] + " - " + sus_data['dst_IP']
scanner_data['dst_port'] = sus_data['dst_port']
scanner_data
```

Out[113]:

	connection	dst_port
4	137.132.228.15 - 193.62.192.8	34282
9	193.62.192.8 - 137.132.228.15	53923
12	193.62.192.8 - 137.132.228.15	53923
19	137.132.228.15 - 193.62.192.8	20480
38	193.62.192.8 - 137.132.228.15	53923
...	...	...
69285	193.62.192.8 - 137.132.228.15	34753
69501	137.132.228.15 - 193.62.192.8	3123
69536	137.132.228.15 - 193.62.192.8	49413
69542	193.62.192.8 - 137.132.228.15	57296
69567	137.132.228.15 - 193.62.192.8	3123

4951 rows × 2 columns

In [115]:
```
plt.figure:figsize=(8,10):
sb.stripplot:data=scanner_data, x='connection', y='dst_port'
```

Out[115]:
```
<AxesSubplot:xlabel='connection', ylabel='dst_port'>
```