In [1]:
```
import json
import pandas as pd
import os
import re
import glob  # determine if this is needed
import numpy as np
import warnings
```

In [2]:
```
# Display all columns in the pd data frame
pd.set_option('display.max_columns', None)

# Ignore warning displays
warnings.filterwarnings('ignore')
```

Stage 1. Read the files, extract the AODV messages and make data frames for each node

In [3]:
```
# Set default AODV parameters
# Most of these will not be required so can delete them.
# Keep for now incase I will build routing tables.
NODE_TRAVERSAL_TIME = 10e-3 # 10 milliseconds
NET_DIAMETER = 20
NET_TRAVERSAL_TIME = 2 * NODE_TRAVERSAL_TIME * NET_DIAMETER
PATH_DISCOVERY_TIME = 2 * NET_TRAVERSAL_TIME
ACTIVE_ROUTE_TIMEOUT = 2 * NET_TRAVERSAL_TIME
DELETE_PERIOD = 5 * NET_TRAVERSAL_TIME
RREQ_RETRIES = 3
RREQ_RATELIMIT = 10
RERR_RATELIMIT = 10
HELLO_INTERVAL = 1 # 1000 milliseconds
ALLOWED_HELLO_LOSS = 2
NEXT_HOP_WAIT = 2 * NODE_TRAVERSAL_TIME
BLACKLIST_TIMEOUT = RREQ_RETRIES * NET_TRAVERSAL_TIME # Double check RREQ_R
```

In [4]:
```
# Get the current working directory
os.getcwd()
```

Out[4]: 'C:\\Users\\alang\\ITS_CP'

In [5]:
```
print("Ensure that the directory only contains the necessary.json files")
#directory_path = input("Set Working Directory. Include ./: ")
directory_path = 'Datasets\DS1\Test_3'
```

```
Ensure that the directory only contains the necessary.json files
```

In [6]:
```
# Verify that the directory path contains the right files only.
print(os.listdir(directory_path))
print()
print(directory_path)
```

```
['dataset3_mod_BHN.csv', 'dataset_merged.csv', 'node-37-manet-routing-com
pare.pcap-37-0.pcap', 'node-38-manet-routing-compare.pcap-38-0.pcap', 'no
de-39-manet-routing-compare.pcap-39-0.pcap', 'node-40-manet-routing-compa
re.pcap-40-0.pcap', 'node-41-manet-routing-compare.pcap-41-0.pcap', 'node
-42-manet-routing-compare.pcap-42-0.pcap', 'node-43-manet-routing-compar
e.pcap-43-0.pcap', 'node-44-manet-routing-compare.pcap-44-0.pcap', 'node-
45-manet-routing-compare.pcap-45-0.pcap', 'node-46-manet-routing-compare.
pcap-46-0.pcap', 'node-47-manet-routing-compare.pcap-47-0.pcap', 'node-48
-manet-routing-compare.pcap-48-0.pcap', 'node-49-manet-routing-compare.pc
ap-49-0.pcap', 'Node_37_Stage_2.csv', 'Node_37_Test3.json', 'Node_38_Stag
e_2.csv', 'Node_38_Test3.json', 'Node_39_Stage_2.csv', 'Node_39_Test3.jso
n', 'Node_40_Stage_2.csv', 'Node_40_Test3.json', 'Node_41_Stage_2.csv',
'Node_41_Test3.json', 'Node_42_Stage_2.csv', 'Node_42_Test3.json', 'Node_
43_Stage_2.csv', 'Node_43_Test3.json', 'Node_44_Stage_2.csv', 'Node_44_Te
st3.json', 'Node_45_Stage_2.csv', 'Node_45_Test3.json', 'Node_46_Stage_2.
csv', 'Node_46_Test3.json', 'Node_47_Stage_2.csv', 'Node_47_Test3.json',
'Node_48_Stage_2.csv', 'Node_48_Test3.json', 'Node_49_Stage_2.csv', 'Node
_49_Test3.json', 'Run2_Test_3_BHN.csv', 'Run_3_Test_3_sim_BHN.csv']

Datasets\DS1\Test_3
```

In [7]:
```
# read all the Node files into a list.
# use os.listdir() to get a list of all files in the directory
all_files = os.listdir(directory_path)

# use a list comprehension to filter the .json files
node_files = [file for file in all_files if file.endswith('.json')]

node_files
```

Out[7]:
```
['Node_37_Test3.json',
 'Node_38_Test3.json',
 'Node_39_Test3.json',
 'Node_40_Test3.json',
 'Node_41_Test3.json',
 'Node_42_Test3.json',
 'Node_43_Test3.json',
 'Node_44_Test3.json',
 'Node_45_Test3.json',
 'Node_46_Test3.json',
 'Node_47_Test3.json',
 'Node_48_Test3.json',
 'Node_49_Test3.json']
```

Enter the list of black hole node IDs here

In [8]:
```
# This list contains the black hole node IDs.
black_hole_list = ['45', '49']

# Concatenate the ip address to the black hole node ID.
# The ip address may vary per simulation and may need to be modified.
for i, node in enumerate(black_hole_list):
    black_hole_list[i] = '10.1.1.' + node

print(black_hole_list)
```

```
['10.1.1.45', '10.1.1.49']
```

Stage 1 functions

In [9]:
```
# It is assumed that the RREQ is received and not sent.
def get_rreq_info(aodv):
    sn = aodv['aodv.orig_ip']
    dn = aodv['aodv.dest_ip']
    hc = aodv['aodv.hopcount']
    ssn = aodv['aodv.orig_seqno']
    dsn = aodv['aodv.dest_seqno']
    bid = aodv['aodv.rreq_id']
    return sn, dn, hc, ssn, dsn, bid
```

In [10]:
```
# It is assumed that the RREP is received and not sent.
def get_rrep_info(aodv):
    sn = aodv['aodv.orig_ip']
    dn = aodv['aodv.dest_ip']
    hc = aodv['aodv.hopcount']
    dsn = aodv['aodv.dest_seqno']
    return sn, dn, hc, dsn
```

In [11]:
```
def get_rrer_info(aodv):
    dn = aodv['Unreachable Destinations']['aodv.unreach_dest_ip']
    hc = aodv['aodv.destcount']
    dsn = aodv['Unreachable Destinations']['aodv.dest_seqno']
    return dn, hc, dsn
```

In [12]:
```
# Max Sequence Number allowed in AODV is 2**32 - 1 = 4294967295
def calc_seq_inc(seq_rrep, seq_rreq):
    max_seq_num = 2**32 -1
    diff = seq_rrep - seq_rreq
    # If the maximum sequence number has been exceeded, calculate the diffe
    if diff < -1e7:
        diff = max_seq_num + seq_rrep - seq_rreq
    return diff
```

In [13]:
```
# Make a list of the node IDs
node_id_list = []
for item in node_files:
    match = re.search(r'\d+', item)
    if match:
        node_id_list.append(match.group())

print(node_id_list)
```

```
['37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48',
'49']
```

Stage 1 for loop

This will loop through all the .json files and convert them into data frames of AODV messages as seen by each node.

In [14]:
```
stage_2_list = []
```

In [15]:

```
for index in range(len(node_files)):
    # Find the number of the current node and store as a string
    crt_node = '10.1.1.' + node_id_list[index]
    print(f'Current Node: {crt_node}')
    type_4 = 0

    # Read the current json file and load it.
    file_path = os.path.join(directory_path, node_files[index])
    j = open(file_path, 'r')
    j1 = json.load(j)
    # Close the JSON file
    j.close()

    # Create an empty data frame
    df_aodv = pd.DataFrame(columns = ['Msg_Index', 'frame_time', 'frame_tir
                                      'Next_Hop', 'TTL', 'AODV_Msg', 'Sourc
                                      'Source_Seq_Num', 'Dest_Seq_Num', 'Br

    # This for loop will iterate through all the AODV messaging lines in th
    # data frame that can be further processed in Stage 2.
    for idx in range(0,len(j1)):
        if 'aodv' in j1[idx]['_source']['layers'].keys():
            frame_time = j1[idx]['_source']['layers']['frame']['frame.time_
            frame_time_rel = j1[idx]['_source']['layers']['frame']['frame.t
            # Double check the next two final keys as there are multiple op
            ip_src = j1[idx]['_source']['layers']['ip']['ip.src']
            ip_dest = j1[idx]['_source']['layers']['ip']['ip.dst']
            ttl = j1[idx]['_source']['layers']['ip']['ip.ttl']
            # Possibly remove the udp timers.
            udp_time_rel = j1[idx]['_source']['layers']['udp']['Timestamps'
            udp_time_delta = j1[idx]['_source']['layers']['udp']['Timestamp
            aodv_tp = j1[idx]['_source']['layers']['aodv']['aodv.type']
            if aodv_tp == '1':
                aodv_type = 'RREQ'
                q_j1 = get_rreq_info(j1[idx]['_source']['layers']['aodv'])
                source_node = q_j1[0]
                dest_node = q_j1[1]
                hop_count = q_j1[2]
                s_seq_num = q_j1[3]
                d_seq_num = q_j1[4]
                bc_id = q_j1[5]

            elif aodv_tp == '2':
                aodv_type = 'RREP'
                p_j1 = get_rrep_info(j1[idx]['_source']['layers']['aodv'])
                source_node = p_j1[0]
                dest_node = p_j1[1]
                hop_count = p_j1[2]
                d_seq_num = p_j1[3]
                s_seq_num = None
                bc_id = None

            elif aodv_tp == '3':
                aodv_type = 'RERR'
                r_j1 = get_rrer_info(j1[idx]['_source']['layers']['aodv'])
                source_node = None
                dest_node = r_j1[0]
```

```
                hop_count = r_j1[1]
                d_seq_num = r_j1[2]
                s_seq_num = None
                bc_id = None

            elif aodv_tp == '4':
                # This is not relevant and causes issues. It will be remove
                # that respond to it.
                #print('"aodv.type": "4"')
                aodv_type = 'RREP-ACK'
                source_node = None
                dest_node = None
                hop_count = 0
                d_seq_num = 0
                s_seq_num = 0
                bc_id = 0
                type_4 += 1

            else:
                print("Error")

            new_row = {'Msg_Index': idx+1, 'frame_time': frame_time, 'frame
                       'This_Node': crt_node, 'Nbr_Node': ip_src, 'TTL': tt
                       'Source_Node': source_node, 'Destination_Node': dest
                       'Source_Seq_Num': s_seq_num, 'Dest_Seq_Num': d_seq_r

            df_aodv = df_aodv.append(new_row, ignore_index=True)

    # Convert various columns from string to float or integer.
    df_aodv['Msg_Index'] = df_aodv['Msg_Index'].astype(int)
    df_aodv['frame_time'] = df_aodv['frame_time'].astype(float)
    df_aodv['frame_time_relative'] = df_aodv['frame_time_relative'].astype(
    df_aodv['TTL'] = df_aodv['TTL'].astype(int)
    df_aodv['Hop_Count'] = df_aodv['Hop_Count'].astype(int)
    df_aodv['Source_Seq_Num'] = pd.to_numeric(df_aodv['Source_Seq_Num'], er
    df_aodv['Dest_Seq_Num'] = df_aodv['Dest_Seq_Num'].astype(int)
    df_aodv['Broadcast_ID'] = pd.to_numeric(df_aodv['Broadcast_ID'], errors

    """
    # Write the Stage 1 .csv files
    file_name = f'Node_{node_id_list[index]}_Stage_1.csv'
    dir_path = directory_path + "\\"
    file_path = dir_path + file_name
    df_aodv.to_csv(file_path)
    """

    # Store the data frame in a list for Stage 2
    stage_2_list.append(df_aodv)
    print(f'{type_4} RREP-ACK messages found')
    print()
```

```
Current Node: 10.1.1.37
110 RREP-ACK messages found

Current Node: 10.1.1.38
15 RREP-ACK messages found

Current Node: 10.1.1.39
4 RREP-ACK messages found

Current Node: 10.1.1.40
48 RREP-ACK messages found

Current Node: 10.1.1.41
29 RREP-ACK messages found

Current Node: 10.1.1.42
89 RREP-ACK messages found

Current Node: 10.1.1.43
```

In [16]:
```
stage_2_list[0].head(15)
```

Out[16]:

	Msg_Index	frame_time	frame_time_relative	This_Node	Nbr_Node	Next_Hop	TTL	AOI
0	1	0.053304	0.000000	10.1.1.37	10.1.1.46	10.1.1.255	1	
1	2	0.055304	0.002000	10.1.1.37	10.1.1.26	10.1.1.255	1	
2	3	0.071304	0.018000	10.1.1.37	10.1.1.36	10.1.1.255	1	
3	4	0.073050	0.019746	10.1.1.37	10.1.1.38	10.1.1.255	1	
4	5	0.080304	0.027000	10.1.1.37	10.1.1.18	10.1.1.255	1	
5	6	1.068304	1.015000	10.1.1.37	10.1.1.36	10.1.1.255	1	
6	7	1.075050	1.021746	10.1.1.37	10.1.1.38	10.1.1.255	1	
7	8	1.092304	1.039000	10.1.1.37	10.1.1.28	10.1.1.255	1	
8	9	2.020304	1.967000	10.1.1.37	10.1.1.44	10.1.1.255	1	
9	10	2.050304	1.997000	10.1.1.37	10.1.1.26	10.1.1.255	1	
10	11	2.054304	2.001000	10.1.1.37	10.1.1.46	10.1.1.255	1	
11	12	2.067304	2.014000	10.1.1.37	10.1.1.36	10.1.1.255	1	
12	13	2.071050	2.017746	10.1.1.37	10.1.1.38	10.1.1.255	1	
13	14	2.075304	2.022000	10.1.1.37	10.1.1.18	10.1.1.255	1	
14	15	2.089304	2.036000	10.1.1.37	10.1.1.28	10.1.1.255	1	

Stage 2. Extract certain information from the messaging and modify the data frames

Set certain parameters for creating the table.

In [17]:
```
# The increment below is considered to be too high and indicative of a BHN
# This value needs to be determined after unning simulations. 20 is an exam
HIGH_DEST_SEQ_NUM_INC = 10
# Assume maximum response time for RREP to be received after and RREQ is se
# This is currently set to more than enough time but we want to ensure that
MAX_RESP_TIME = 5
```

Stage 2 for loop

This will loop through all the data frames created in Stage 1 and add some new features for each row.

In [18]:
```
stage_3_list = []
```

In [19]:

```
for df_idx in range(len(stage_2_list)):
    df_mod = stage_2_list[df_idx].copy()
    # Maybe remove these two features in the future.
    df_mod['This_Node_Is_Dest'] = ""
    df_mod['This_Node_Is_Orig'] = ""
    # Create new columns.
    df_mod['Hello'] = ""
    df_mod['Nbr_Is_Orig'] = ""
    df_mod['Nbr_Is_Dest'] = ""
    df_mod['RREQ_Msg_Idx'] = ""
    df_mod['RREP_Resp_Time'] = ""
    df_mod['RREP_Resp_Time_Per_Hop'] = ""
    df_mod['Hop_Cnt_Over_1'] = ""
    df_mod['Dest_Seq_Num_Increment'] = ""
    df_mod['Orig_Seq_Num_Increment'] = ""
    # This indicates which rows will be deleted in the dataset creation.
    df_mod['Tagged_For_Del'] = ""

    type_4 = 0
    no_rreq = 0

    # Rows that are affected by the RREP-ACK message will be removed.
    delete_list = []
    print(f'Working on Node {node_id_list[df_idx]} **********************

    # Modify the data frame to create more features.
    for idx in range(len(df_mod)):
        # RERR Messages do not contain useful information and cause problem
        # They will be removed.
        if df_mod['AODV_Msg'][idx] == 'RREP-ACK':
            delete_list.append(df_mod['Msg_Index'][idx])
            type_4 += 1
            df_mod['Tagged_For_Del'][idx] = 'RREP-ACK'

        # Check if This Node is source or destination
        if df_mod['This_Node'][idx] == df_mod['Source_Node'][idx]:
            df_mod['This_Node_Is_Orig'][idx] = 'Yes'
        else:
            df_mod['This_Node_Is_Orig'][idx] = 'No'

        if df_mod['This_Node'][idx] == df_mod['Destination_Node'][idx]:
            df_mod['This_Node_Is_Dest'][idx] = 'Yes'
        else:
            df_mod['This_Node_Is_Dest'][idx] = 'No'

        # Extract information from the RREQ messages
        if df_mod['AODV_Msg'][idx] == 'RREQ':
            if df_mod['Nbr_Node'][idx] == df_mod['Source_Node'][idx]:
                df_mod['Nbr_Is_Orig'][idx] = True
            else:
                df_mod['Nbr_Is_Orig'][idx] = False

        # Extract information from the RREP messages
        # Determine if the RREP message is a "Hello" broadcast.
        if df_mod['AODV_Msg'][idx] == 'RREP':
            if (df_mod['Next_Hop'][idx].endswith('.255') and
               (df_mod['Source_Node'][idx] == df_mod['Destination_Node'][id
```

```
            # Condition met for Hello, label as such
            df_mod['Hello'][idx] = True

        # If RREP is not a "Hello" broadcast.
        else:
            df_mod['Hello'][idx] = False
            # Determine if the neighbor was the destination node of the
            if ((df_mod['Source_Node'][idx] != df_mod['Destination_Node
                (df_mod['Nbr_Node'][idx] == df_mod['Destination_Node']]
                # The neighbour is the destination of the RREP message
                df_mod['Nbr_Is_Dest'][idx] = True

            else:
                # Neighbor is not the destination node
                df_mod['Nbr_Is_Dest'][idx] = False

            candidate_idx = None
            idx2 = idx # This starts from the current RREP but will not

            # Only RREQ messages within the max response time will be c
            # This while loop will determine the 'candidate' or correct
            # Modify code and add comments here on how this is done onc
            while (df_mod['frame_time_relative'][idx] - df_mod['frame_t
                #print(idx2, df_mod['frame_time_relative'][idx] - df_mo
                if df_mod['AODV_Msg'][idx2] == 'RREQ':
                    #print(idx2, df_mod['Msg_Index'][idx2], df_mod['fro
                          #df_mod['Next_Hop'][idx2],
                          #df_mod['Source_Node'][idx2], df_mod['Destino
                    # Determine if RREQ is valid for RREP
                    # It will be valid if the Source_Node and Destinati
                    # This code will be modified once the RREP in the .
                    if ((df_mod['Source_Node'][idx2] == df_mod['Source_
                        (df_mod['Destination_Node'][idx2] == df_mod['De
                        (df_mod['TTL'][idx2] >= df_mod['Hop_Count'][idx
                        if candidate_idx == None:
                            #print()
                            #print(f"Candidate, idx2: {idx2}, TTL: {df_
                            candidate_idx = idx2
                        else:
                            if df_mod['frame_time_relative'][idx2] <= c
                                #print()
                                #print(f"Candidate, idx2: {idx2}, TTL:
                                #RREP Hop_Count: {df_mod['Hop_Count'][i
                                candidate_idx = idx2
                            #else:
                                #print()
                                #print(f"Not A Candidate, idx2: {idx2},
                                #RREP Hop_Count: {df_mod['Hop_Count'][i

                # Decremenet the index counter.
                idx2 -= 1
                if idx2 < 0: # If the index is less than 0, break the l
                    break

            # If the correct RREQ message was found pertaining to the R
            if candidate_idx != None:
                #print(f"final candidate: {candidate_idx}, TTL: {df_mod
```

```
                    # Update the required columns
                    df_mod['RREQ_Msg_Idx'][candidate_idx] = idx
                    df_mod['RREQ_Msg_Idx'][idx] = candidate_idx
                    df_mod['RREP_Resp_Time'][idx] = \
                        df_mod['frame_time_relative'][idx] - df_mod['frame_
                    # + 1 added to the demoniator as the hop count should b
                    # This will also avoid a demoniator of 0.
                    df_mod['RREP_Resp_Time_Per_Hop'][idx] = \
                        df_mod['RREP_Resp_Time'][idx] / (df_mod['Hop_Count'
                    df_mod['Dest_Seq_Num_Increment'][idx] = \
                        calc_seq_inc(df_mod['Dest_Seq_Num'][idx], df_mod['D
                    # df_mod['Orig_Seq_Num_Increment'][idx] = calc_seq_inc(

                # If the RREQ message was not found
                else:
                    # print(f'No RREQ found for RREP Index {idx}. This is l
                    no_rreq += 1
                    delete_list.append(df_mod['Msg_Index'][idx])
                    df_mod['Tagged_For_Del'][idx] = 'No matching RREQ'

                # Determine if the number of hop counts is greater than 1 i
                if  (df_mod['Nbr_Is_Dest'][idx] == False):
                    if (df_mod['Hop_Count'][idx] > 1):
                        df_mod['Hop_Cnt_Over_1'][idx] = True
                    else:
                        df_mod['Hop_Cnt_Over_1'][idx] = False


    # Write the Stage 2 .csv files
    file_name = f'Node_{node_id_list[df_idx]}_Stage_2.csv'
    dir_path = directory_path + "\\"
    file_path = dir_path + file_name
    df_mod.to_csv(file_path)

    # The rows will be deleted for Stage 3, but will be in the Stage 2 csv
    print(f'{type_4} RREP-ACK messages. and {no_rreq} RREPs with no matchin
    # filter the rows to be deleted based on the values in column Msg_Index
    rows_to_delete = df_mod[df_mod['Msg_Index'].isin(delete_list)].index
    # drop the rows from the dataframe
    df_mod = df_mod.drop(index=rows_to_delete)
    # Reset the indices
    df_mod = df_mod.reset_index()


    # Store the data frame in a list for Stage 3
    stage_3_list.append(df_mod)

    print()
```

```
Working on Node 37 ****************************************
110 RREP-ACK messages. and 457 RREPs with no matching RREQ. They will b
e deleted

Working on Node 38 ****************************************
15 RREP-ACK messages. and 101 RREPs with no matching RREQ. They will be
deleted

Working on Node 39 ****************************************
4 RREP-ACK messages. and 6 RREPs with no matching RREQ. They will be de
leted

Working on Node 40 ****************************************
48 RREP-ACK messages. and 408 RREPs with no matching RREQ. They will be
deleted

Working on Node 41 ****************************************
29 RREP-ACK messages. and 111 RREPs with no matching RREQ. They will be
deleted
```

Stage 3. Create individual Subject Node datasets and merge them into one dataset.

In [20]:
```
node_dataset_list = []
```

In [21]:

```
for df_idx in range (len(stage_3_list)):
    print(f'Working on Node {node_id_list[df_idx]} **********************
    df_mod = stage_3_list[df_idx].copy()
    # Remove all rows where the Subject Node is the same as the Neighbour N
    df_mod = df_mod[df_mod['This_Node'] != df_mod['Nbr_Node']]
    # Reset the indices
    df_mod = df_mod.reset_index()
    # Create an empty dataset
    ds_temp = pd.DataFrame(columns = ['Node', 'Nbr_Node', 'Nbr_Count', 'Hel
                                      'RREQs_From_Nbr', 'Nbr_Never_Sends_RF
                                      'Nbr_Is_Dest_Cnt','Nbr_Never_Dest',
                                      'RREP_Resp_Pct', 'Pct_Of_All_RREPs',
                                      'High_Dest_Seq_Num_Inc_Cnt', 'High_De
                                      'Avg_Resp_Dly_Per_Hop', 'RERRs_From_N
                                      'RREP_To_Nbrs_Ratio', 'Black_Hole_Noc

    # Create Empty lists
    # Concatenation of Source Node and Broadcast ID
    sn_bc_id_list = []

    # Set initial counters to 0
    node = df_mod['This_Node'][0]
    All_RREPs_Rcvd_This_Node = 0
    RREQs_Sent_To_Nbr = 0
    # Create the Individual dataset
    for idx in range(0,len(df_mod)):
        # Error in current NS3 files. There should be no messaging between
        if df_mod['Nbr_Node'][idx] != node:
            # If the neighbour node appears for the first time
            if (df_mod['Nbr_Node'][idx] not in ds_temp['Nbr_Node'].values):
                # Create a new row
                new_row = {'Node': node, 'Nbr_Count': 0, 'Nbr_Node': df_moc
                           'AODV_Msg_Nbr_Cnt': 0, 'RREQs_Sent_To_Nbr': 0,
                           'Nbr_Is_Orig_Cnt': 0, 'Nbr_Never_Orig': False,
                           'All_RREPs_Rcvd_This_Node': 0, 'RREPs_From_Nbr':
                           'Pct_Of_All_RREPs': np.nan, 'Hop_Cnt_Over_1_Cnt'
                           'Hop_Cnt_Over_1_Pct': np.nan, 'High_Dest_Seq_Num
                           'Avg_Resp_Dly': 0, 'Avg_Resp_Dly_Per_Hop': 0, 'F
                           'Pct_of_All_Nbrs': np.nan, 'RREP_To_Nbrs_Ratio':

                # Add the new row
                ds_temp = ds_temp.append(new_row, ignore_index=True)

            # Add the appropriate values
            # If message is "Hello Broadcast" then increment counter
            if df_mod['Hello'][idx]:
                ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][idx],

            # check if a message in not a Hello and incremenet the neighbou
            if (df_mod['AODV_Msg'][idx] in ('RREQ', 'RREP', 'RERR')) and (d
                ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][idx],

            # Check if TTL > 1
            if ((df_mod['AODV_Msg'][idx] == 'RREQ') and (df_mod['TTL'][idx]
                (df_mod['This_Node'][idx] != df_mod['Nbr_Node'][idx])):
                sn_bc_id = df_mod['Source_Node'][idx] + '_' + str(df_mod['B
                # Increment counter only if the source node has not already
```

```
                if sn_bc_id not in sn_bc_id_list:
                    sn_bc_id_list.append(sn_bc_id)
                    RREQs_Sent_To_Nbr += 1

            # Set RREQ Features
            if df_mod['AODV_Msg'][idx] == 'RREQ':
                ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][idx],

                if df_mod['Source_Node'][idx] == df_mod['Nbr_Node'][idx]:
                    ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][i

                if df_mod['Destination_Node'][idx] == df_mod['Nbr_Node'][id
                    ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][i

            # Set RREP Features if not a 'Hello' broadcast
            if (df_mod['AODV_Msg'][idx] == 'RREP') and (df_mod['Hello'][idx
                All_RREPs_Rcvd_This_Node += 1
                ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][idx],
                # Note: Below the totals delays are being accumulated. They
                ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][idx],
                ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][idx],
                        'Avg_Resp_Dly_Per_Hop'] += df_mod['RREP_Resp_Time_P

                if df_mod['Hop_Cnt_Over_1'][idx]:
                    ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][i
                if df_mod['Dest_Seq_Num_Increment'][idx] > HIGH_DEST_SEQ_NU
                    ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][i

            # Set RERR Features
            if df_mod['AODV_Msg'][idx] == 'RERR':
                ds_temp.loc[ds_temp['Nbr_Node'] == df_mod['Nbr_Node'][idx],


        # Add totals to the appropriate columns
        ds_temp['All_RREPs_Rcvd_This_Node'] = All_RREPs_Rcvd_This_Node
        ds_temp['RREQs_Sent_To_Nbr'] = RREQs_Sent_To_Nbr

        # Set appropriate data types
        ds_temp['Nbr_Count'] = ds_temp['Nbr_Count'].astype(int)
        ds_temp['Hello_Cnt'] = ds_temp['Hello_Cnt'].astype(int)
        ds_temp['AODV_Msg_Nbr_Cnt'] = ds_temp['AODV_Msg_Nbr_Cnt'].astype(int)
        ds_temp['RREQs_Sent_To_Nbr'] = ds_temp['RREQs_Sent_To_Nbr'].astype(int)
        ds_temp['RREQs_From_Nbr'] = ds_temp['RREQs_From_Nbr'].astype(int)
        ds_temp['Nbr_Never_Sends_RREQ'] = ds_temp['Nbr_Never_Sends_RREQ'].astyp
        ds_temp['Nbr_Is_Orig_Cnt'] = ds_temp['Nbr_Is_Orig_Cnt'].astype(int)
        ds_temp['Nbr_Never_Orig'] = ds_temp['Nbr_Never_Orig'].astype(bool)
        ds_temp['Nbr_Is_Dest_Cnt'] = ds_temp['Nbr_Is_Dest_Cnt'].astype(int)
        ds_temp['Nbr_Never_Dest'] = ds_temp['Nbr_Never_Dest'].astype(bool)
        ds_temp['All_RREPs_Rcvd_This_Node'] = ds_temp['All_RREPs_Rcvd_This_Node
        ds_temp['RREPs_From_Nbr'] = ds_temp['RREPs_From_Nbr'].astype(int)
        ds_temp['RREP_Resp_Pct'] = ds_temp['RREP_Resp_Pct'].astype(float)
        ds_temp['Pct_Of_All_RREPs'] = ds_temp['Pct_Of_All_RREPs'].astype(float)
        ds_temp['Hop_Cnt_Over_1_Cnt'] = ds_temp['Hop_Cnt_Over_1_Cnt'].astype(in
        ds_temp['Hop_Cnt_Over_1_Pct'] = ds_temp['Hop_Cnt_Over_1_Pct'].astype(fl
        ds_temp['High_Dest_Seq_Num_Inc_Cnt'] = ds_temp['High_Dest_Seq_Num_Inc_C
        ds_temp['High_Dest_Seq_Num_Inc_Pct'] = ds_temp['High_Dest_Seq_Num_Inc_P
        ds_temp['Avg_Resp_Dly'] = ds_temp['Avg_Resp_Dly'].astype(float)
```

```
    ds_temp['Avg_Resp_Dly_Per_Hop'] = ds_temp['Avg_Resp_Dly_Per_Hop'].astyp
    ds_temp['RERRs_From_Nbr'] = ds_temp['RERRs_From_Nbr'].astype(int)
    ds_temp['RERRs_From_Nbr_Pct'] = ds_temp['RERRs_From_Nbr_Pct'].astype(fl
    ds_temp['Pct_of_All_Nbrs'] = ds_temp['Pct_of_All_Nbrs'].astype(float)
    ds_temp['RREP_To_Nbrs_Ratio'] = ds_temp['RREP_To_Nbrs_Ratio'].astype(fl
    ds_temp['Black_Hole_Node'] = ds_temp['Black_Hole_Node'].astype(bool)

    # Change the remaining values
    # Change the target variable to True if the Neighbour Node is in the bl
    ds_temp.loc[ds_temp['Nbr_Node'].isin(black_hole_list), 'Black_Hole_Node
    # Number of Neighbors for this Node
    ds_temp['Nbr_Count'] = len(ds_temp['Nbr_Node'].values)
    # Nbr never sends a RREQ
    ds_temp['Nbr_Never_Sends_RREQ'] = (ds_temp['RREQs_From_Nbr'] == 0)
    # Nbr never originates a RREQ
    ds_temp['Nbr_Never_Orig'] = (ds_temp['Nbr_Is_Orig_Cnt'] == 0)
    # Nbr is never the destination of an RREQ
    ds_temp['Nbr_Never_Dest'] = (ds_temp['Nbr_Is_Dest_Cnt'] == 0)

    # Calculate to percentage of RREP Responses from RREQs.
    # There are 2 counters. Decide which one I prefer.
    ds_temp.loc[ds_temp['RREQs_Sent_To_Nbr'] != 0, 'RREP_Resp_Pct'] = \
        round((100 * ds_temp['RREPs_From_Nbr'] / ds_temp['RREQs_Sent_To_Nbr

    ds_temp.loc[ds_temp['All_RREPs_Rcvd_This_Node'] != 0, 'Pct_Of_All_RREPs
        round((100 * ds_temp['RREPs_From_Nbr'] / ds_temp['All_RREPs_Rcvd_Th

    ds_temp.loc[ds_temp['RREPs_From_Nbr'] != 0, 'Hop_Cnt_Over_1_Pct'] = \
        round((100 * ds_temp['Hop_Cnt_Over_1_Cnt'] / ds_temp['RREPs_From_Nb

    ds_temp.loc[ds_temp['RREPs_From_Nbr'] != 0, 'High_Dest_Seq_Num_Inc_Pct'
        round((100 * ds_temp['High_Dest_Seq_Num_Inc_Cnt'] / ds_temp['RREPs_

    ds_temp.loc[ds_temp['RREPs_From_Nbr'] != 0, 'RERRs_From_Nbr_Pct'] = \
        round((100 * ds_temp['RERRs_From_Nbr'] / ds_temp['RREPs_From_Nbr'])

    ds_temp.loc[ds_temp['Nbr_Count'] != 0, 'Pct_of_All_Nbrs'] = \
        round((100 * 1 / ds_temp['Nbr_Count']).astype(float),2)

    ds_temp.loc[ds_temp['Pct_of_All_Nbrs'] != 0, 'RREP_To_Nbrs_Ratio'] = \
        round((ds_temp['Pct_Of_All_RREPs'] / ds_temp['Pct_of_All_Nbrs']).as

    ds_temp['Avg_Resp_Dly'] = np.where(ds_temp['RREPs_From_Nbr'] != 0, \
           (ds_temp['Avg_Resp_Dly'] / ds_temp['RREPs_From_Nbr']), np.nan)

    ds_temp['Avg_Resp_Dly_Per_Hop'] = np.where(ds_temp['RREPs_From_Nbr'] !=
           (ds_temp['Avg_Resp_Dly_Per_Hop'] / ds_temp['RREPs_From_Nbr']), n

    # Append this data set to the list
    node_dataset_list.append(ds_temp)
    print(f'Completed Working on Node {node_id_list[df_idx]} **************
    print()
```

```
Working on Node 37 ************************************************************
**********
Completed Working on Node 37 **************************************************
**********

Working on Node 38 ************************************************************
**********
Completed Working on Node 38 **************************************************
**********

Working on Node 39 ************************************************************
**********
Completed Working on Node 39 **************************************************
**********

Working on Node 40 ************************************************************
**********
Completed Working on Node 40 **************************************************
**********
```

In [22]:
```
node_dataset_list[0]
```

Out[22]:

	Node	Nbr_Node	Nbr_Count	Hello_Cnt	AODV_Msg_Nbr_Cnt	RREQs_Sent_To_Nbr	F
0	10.1.1.37	10.1.1.46	36	37	100	159	
1	10.1.1.37	10.1.1.26	36	44	15	159	
2	10.1.1.37	10.1.1.36	36	21	0	159	
3	10.1.1.37	10.1.1.38	36	155	219	159	
4	10.1.1.37	10.1.1.18	36	48	56	159	
5	10.1.1.37	10.1.1.28	36	44	0	159	
6	10.1.1.37	10.1.1.44	36	17	0	159	
7	10.1.1.37	10.1.1.16	36	33	0	159	
8	10.1.1.37	10.1.1.25	36	30	94	159	
9	10.1.1.37	10.1.1.5	36	30	58	159	
10	10.1.1.37	10.1.1.34	36	23	10	159	
11	10.1.1.37	10.1.1.43	36	14	31	159	
12	10.1.1.37	10.1.1.15	36	8	0	159	
13	10.1.1.37	10.1.1.7	36	12	0	159	
14	10.1.1.37	10.1.1.29	36	83	8	159	
15	10.1.1.37	10.1.1.6	36	25	84	159	
16	10.1.1.37	10.1.1.10	36	70	0	159	
17	10.1.1.37	10.1.1.14	36	44	29	159	
18	10.1.1.37	10.1.1.8	36	63	35	159	
19	10.1.1.37	10.1.1.39	36	23	0	159	
20	10.1.1.37	10.1.1.33	36	34	73	159	
21	10.1.1.37	10.1.1.2	36	20	94	159	
22	10.1.1.37	10.1.1.49	36	22	0	159	
23	10.1.1.37	10.1.1.27	36	33	13	159	
24	10.1.1.37	10.1.1.1	36	39	124	159	
25	10.1.1.37	10.1.1.32	36	35	91	159	
26	10.1.1.37	10.1.1.9	36	51	0	159	
27	10.1.1.37	10.1.1.20	36	28	0	159	
28	10.1.1.37	10.1.1.17	36	23	103	159	
29	10.1.1.37	10.1.1.31	36	15	12	159	
30	10.1.1.37	10.1.1.4	36	12	0	159	
31	10.1.1.37	10.1.1.3	36	19	126	159	
32	10.1.1.37	10.1.1.30	36	12	67	159	
33	10.1.1.37	10.1.1.22	36	15	66	159	
34	10.1.1.37	10.1.1.23	36	6	23	159	

	Node	Nbr_Node	Nbr_Count	Hello_Cnt	AODV_Msg_Nbr_Cnt	RREQs_Sent_To_Nbr	F
35	10.1.1.37	10.1.1.47	36	2	17	159	

In [23]:
```
# Create the final dataset
dataset_merged = pd.concat(node_dataset_list)

# Remove the rows where the subject node is a black hole.
# The dataset should be used to train normal nodes to detect black hole noc
dataset_merged = dataset_merged[~dataset_merged['Node'].isin(black_hole_lis

# Reset the indices
dataset_merged = dataset_merged.rename_axis('Index').reset_index()
dataset_merged['Index'] = range(1, len(dataset_merged) + 1)
```

In [24]:
```
dataset_merged
```

Out[24]:

	Index	Node	Nbr_Node	Nbr_Count	Hello_Cnt	AODV_Msg_Nbr_Cnt	RREQs_Sen
0	1	10.1.1.37	10.1.1.46	36	37	100	
1	2	10.1.1.37	10.1.1.26	36	44	15	
2	3	10.1.1.37	10.1.1.36	36	21	0	
3	4	10.1.1.37	10.1.1.38	36	155	219	
4	5	10.1.1.37	10.1.1.18	36	48	56	
...	...	...	...	...	...	...	
375	376	10.1.1.48	10.1.1.41	47	10	177	
376	377	10.1.1.48	10.1.1.15	47	25	180	
377	378	10.1.1.48	10.1.1.19	47	2	27	
378	379	10.1.1.48	10.1.1.11	47	5	93	
379	380	10.1.1.48	10.1.1.13	47	7	29	

In [25]:
```
# Store the final dataset as a .csv file
file_name = f'dataset_merged.csv'
dir_path = directory_path + "\\"
file_path = dir_path + file_name
dataset_merged.to_csv(file_path)
```