```
# %%
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import threading
import random
import pandas as pd
from tangle import node, Transaction

# %%
### This is the implementation of the PC attack###
class mal_node(node):

    def __init__(self, edges, nodeID, tangle, ww, watcher):
        self.id = nodeID
        self.neighbourhood = edges
        self.signature = np.random.randint(2048)
        self.ww = 1
        self.tangle = tangle
        self.ds_start = None
        self.chain = []
        self.watcher = watcher
        self.approve_point = None

    def issue_bad_transaction(self):
        content = random.randint(0, 100)
        nodeSig = self.signature
        self.tangle.next_transaction(self.ww, content, True)
        self.catch_PC_tr()

    def spam_transactions(self, num):
        for i in range(num):
            content = random.randint(0, 100)
            nodeSig = self.signature
            self.mal_next_transaction(self.ww, content, False)
            self.watcher.update()

    def catch_PC_tr(self):
        for t in self.tangle.transactions:
            if t.DS_transaction == True:
                self.ds_start = t
                self.chain.append(t)

    ### These are adapted from the original base simulator  ###
    def mal_mcmc(self):
        found = False
        len_low = np.floor(len(self.tangle.transactions))
        while found == False:
            num_particles = 10
            lower_bound = int(np.maximum(0, self.tangle.count - 20.0*self.tangle.rate))
            upper_bound = int(np.maximum(len_low,
                                         self.tangle.count - 10.0*self.tangle.rate))
            candidates = self.tangle.transactions[lower_bound:upper_bound]
            particles = np.random.choice(candidates, num_particles)
            threads = []
            for p in particles:
                t = threading.Thread(target=self.tangle._walk2(p))
                threads.append(t)
                t.start()
            for th in threads:
                th.join()
            tips = self.tangle.tip_walk_cache[:1]
            found = True

        self.tangle.tip_walk_cache = list()
        return tips
```

```
    def mal_next_transaction(self, NodeWeight, content, DS):
        self.tangle.count += 1
        tip2 = self.tangle.chain_attach_point
        tip1 = self.chain[-1]
        approved_tips = [tip1, tip2]
        transaction = Transaction(self.tangle, self.tangle.time,
                                  approved_tips, self.tangle.count - 1, NodeWeight,
                                  content, DS)
        for t in approved_tips:
            t.approved_time = np.minimum(self.tangle.time, t.approved_time)
            t._approved_directly_by.add(transaction)

            if hasattr(self.tangle, 'G'):
                self.tangle.G.add_edges_from([(transaction.num, t.num)])

        self.tangle.transactions.append(transaction)
        self.chain.append(transaction)
        self.tangle.cw_cache = {}
```