St. Thomas' College of Engineering and Technology

Department of Computer Science and Engineering

Implementation of Custom Routing Algorithm in Cloud

Prepared by

Disha Bhattacharya	**181220110024**
Biswajeet Chakraborty	**181220110019**
Palash Dey	**181220110037**
Ananya Laha	**181220110006**

Under the guidance of

Prof. Sanghamitra De.

Assistant Professor

Project Report

Submitted in the partial fulfillment of the requirement for the degree of

B. Tech in Computer Science and Engineering.

Affiliated to

Maulana Abul Kalam Azad University of Technology, West Bengal

June, 2022

St. Thomas' College of Engineering and Technology

Department of Computer Science and Engineering

This is to certify that the work in preparing the project entitled "**Implementation of Custom Routing Algorithm in Cloud**" has been carried out by **Disha Bhattacharya, Biswajeet Chakraborty, Palash Dey** and **Ananya Laha** under my guidance during the session **2021-2022** and accepted in the partial fulfillment of the requirement for the degree of **B. Tech in Computer Science and Engineering**.

Prof. Sanghamitra De

Assistant Professor

Department of Computer

Science and Engineering

St. Thomas' College of Engineering Technology

Dr. Mousumi Dutt

Head of the Department

Department of Computer

Science and Engineering

St. Thomas' College of Engineering and Technology

St. Thomas' College of Engineering and Technology

Department of Computer Science and Engineering

Acknowledgement

We respect and thank Prof. Sanghamitra De, for providing us an opportunity to do the project work and giving us all support and guidance, which made us complete the project duly. We are extremely thankful to her for providing such a nice support and guidance.

We owe our deep gratitude to our project mentor Prof. Sanghamitra De, who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

We heartily thank Dr. Mousumi Dutt, Head of Department, Computer Science & Engineering, for her guidance and suggestions during this project work.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of Computer Science & Engineering which helped us in successfully completing our project work. Also, we would like to extend our sincere esteems to all staff in laboratory for their timely support.

Signature with date

Disha Bhattacharya, 12200118051

Signature with date

Biswajeet Chakraborty, 12200118056

Signature with date

Palash Dey, 12200118038

Signature with date

Ananya Laha, 12200118069

St. Thomas' College of Engineering and Technology

Department of Computer Science and Engineering

DECLARATION

We declare that this written submission represents our ideas in our own words and we have adequately cited and referenced the original sources. We also declare that we have adhered to all the principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Disha Bhattacharya	**Biswajeet Chakraborty**
Roll No.: 12200118051	**Roll No.: 12200118056**
Palash Dey	**Ananya Laha**
Roll No.: 12200118038	**Roll No.: 12200118069**

INDEX

1. Pre-amble

1.1 Vision and Mission

Vision and Mission of the Institute

Vision of the Institute

• To evolve as an industry oriented, research-based Institution for creative solutions in various engineering domains, with an ultimate objective of meeting technological challenges faced by the Nation and the Society.

Mission of the Institute

• To enhance the quality of engineering education and delivery through accessible, comprehensive and research-orient teaching-learning- assessment processes in the state-of-art environment.

• To create opportunities for students and faculty members to acquire professional knowledge and develop managerial, entrepreneurial and social attitudes with highly ethical and moral values.

• To satisfy the ever-changing needs of the nation with respect to evolution and absorption of sustainable and environment friendly technologies for effective creation of knowledge-based society in the global era.

Vision and Mission of the Department

Vision of the Department

• To continually improve upon the teaching-learning processes and research with a goal to develop quality technical manpower with sound academic and practical experience, who can respond to challenges and changes happening dynamically in Computer Science and Engineering.

Mission of the Department

• To inspire the students to work with latest tools and to make them industry ready.

• To impart research based technical knowledge.

• To groom the department as a learning centre to inculcate advanced technologies in Computer Science and Engineering with social and environmental awareness.

1.2 Program Outcome (PO) and Program Specific Outcomes (PSO)

Program Outcome (PO)

• Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and engineering specialization to the solution of complex engineering problems.

• Problem Analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using the first principles of mathematics, natural sciences, and engineering science.

• Design & Development of Solutions: Design solutions for complex engineering problems and design system components, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

• Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

• Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

• The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

• Environment and sustainability: Understand the impact of the engineering based automations and solutions in environmental and societal contexts and demonstrate the knowledge towards the need for sustainable development and support.

• Ethics: Apply ethical principles and commit to professional.

Ethics and responsibilities and norms of the engineering practice.

• Individual and team work: Facing the challenge and inculcating a strong problem solving attitude, along with being a good team member/leader who can communicate well in multidisciplinary settings.

• Communication: Comprehend, analyse and formulate.

Technical reports. Design formal reports, make efficient presentations, execute instructions from seniors and optimally distribute work load to juniors.

• Project Management and Finance: Demonstrate knowledge.

And understanding of the engineering and management principals in implementing project planning, staffing, and scheduling, preparing cost estimation reports and handle other financial aspects to manage projects as a member/leader of a team.

• Lifelong learning: Recognize the need and preparation for technological upgradation with the advancement of technology.

Program Specific Outcomes (PSO)

• Programming skills: Apply fundamental knowledge and programming aptitude to identify, design and solve real life problems.

• Professional skills: Students shall understand, analyse and develop software solutions to meet the requirements of industry and society.

• Competency: Students will be competent for competitive examinations for employment, higher studies and research.

1.3 PO and PSO Mapping with Justification

Project	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
PROJ-CS794	3	3	3	3	2	3	3	3	3	3	2	3	3	3	3

Justification:

1. PO1(Engineering Knowledge): We apply the knowledge of engineering fundamentals for our project.

2. PO2(Problem Analysis): We identified and formulated a solution to problems of our project using engineering science

3. PO3(Design & Development): We analysed and interpreted the data for complex engineering problems and design system components, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

4. PO4(Conduct Investigations of Complex Problems): It may arise during implementation of the project.

5. PO5(Modern Tool Usage): Google Cloud Platform and Python programming language will be used

6. PO6(The Engineer and Society): This will help the user to distribute the data in a network in minimal time. It can be used for recovery of an infected network.

7. PO7(Environment and sustainability): The basic environment for this project will be Google Cloud Platform. And all the codes will be written using Python.

8. PO8(Ethics): We shall follow all the engineering ethics.

9. PO9(Individual and team work): Discussion in group and contribution at an individual level has helped to develop.

10. PO10(Communication): Communication has been an effective way for completing the pre-implementation stage.

11. PO11(Project Management and Finance): The project has been designed keeping financial expense in mind.

12. PO12(Lifelong learning): Python is a very popular language, so learning of this language will help us in future. Having knowledge on Google Cloud Platform will be beneficial in future too.

13. PSO1(Programming skills: Apply fundamental knowledge and programming aptitude to identify, design and solve real life): Will be tested in the implementation stage of this project.

14. PSO2(Professional skills): The requirements have been analysed and a prototype will be designed for the same.

15. PSO3(Competency): This project will help us in employment, higher studies and research.

1. INTRODUCTION

1.1. Objective of the Project

The purpose of this project is to implement a routing algorithm in Google Cloud Platform (GCP) for a network of seven nodes. It is an enhancement of the controlled flooding algorithm by adding on the concept of percolation centrality to broadcast the messages via the adjacent nodes. The nodes of our network distribute the necessary message throughout the network.

1.2. Brief description of Project

The routing algorithm is a formula that is stored in the router's memory. The routing algorithm that our protocol uses is a major factor in the performance of our routing environment. The purpose of the routing algorithm is to make decisions for the router concerning the best paths for data.

We can consider the routing algorithm as the traffic officer of the router. In the same way that traffic officers guide and shape the way cars drive through busy intersections, routing algorithms make decisions concerning the path data will take from one network to another.

The router uses the routing algorithm to compute the path that would best serve to transport the data from the source to the destination. However, one cannot directly choose the algorithm that their router uses. Rather, the routing protocol that one chooses for their network determines which algorithm one will use. For example, whereas the routing protocol RIP may use one type of routing algorithm to help the router move data, the routing protocol OSPF uses another. **[1]**

1.3. Tools & Platform

Our project is done in the Google Cloud Platform (GCP). Our Virtual machines are present in the compute engine. We create 7 virtual machines or nodes to build a network. There's an extra node or the 8th node, which is the base system from where the connections begin.

Since we kept every node in same subnet i.e., 10.128.0, it formed a complete graph. So, we had to block the node connections, else every node would have pinged every other node. Like for example, node 5 would have pinged node 1, node 2, node 3, node 4, node 6 and node 7 instead of just having connection with node 7.

Thus, our required network would not have been achieved. So, there were two methods to resolve it. One was stating the firewall rules in GCP, that is, node 5 would have had an incoming connection only from node 7 as per our network.

The second method, the one that we applied in our program is we made a list called destination and it holds just the respective node's destination node. Like for node 7, its destination nodes are node 1, node 2, node 5, node 6. So now when we'll be using the 'accept' method, the resources will be passed only to these destination nodes.

1.4. Project Organization

The routing algorithm that we made is basically a broadcast algorithm which determines the fastest route to deliver a message. That is, we determine the node which can pass on the resource from source to destination at the least amount of time. Our routing algorithm is similar to the controlled flooding algorithm. We enhance the flooding algorithm using the concepts of percolation centrality and recovery procedure. We send a message that percolates via the nodes of our network.

The network setup:

In order to implement our algorithm, we have first created 7 nodes with the following Internet Protocols (IPs):

Start Node = 10.128.0.9

Node1 = 10.128.0.2

Node2 = 10.128.0.3

Node3 = 10.128.0.4

Node4 = 10.128.0.5

Node5 = 10.128.0.6

Node6 = 10.128.0.7

Node7 = 10.128.0.8

Firstly, we made a Virtual private cloud. It is an internal network where in every region it provides an IP range. Then the firewall rules were being added.

SSH is the protocol we used to login to these virtual machines.

1.5. Project Timeline

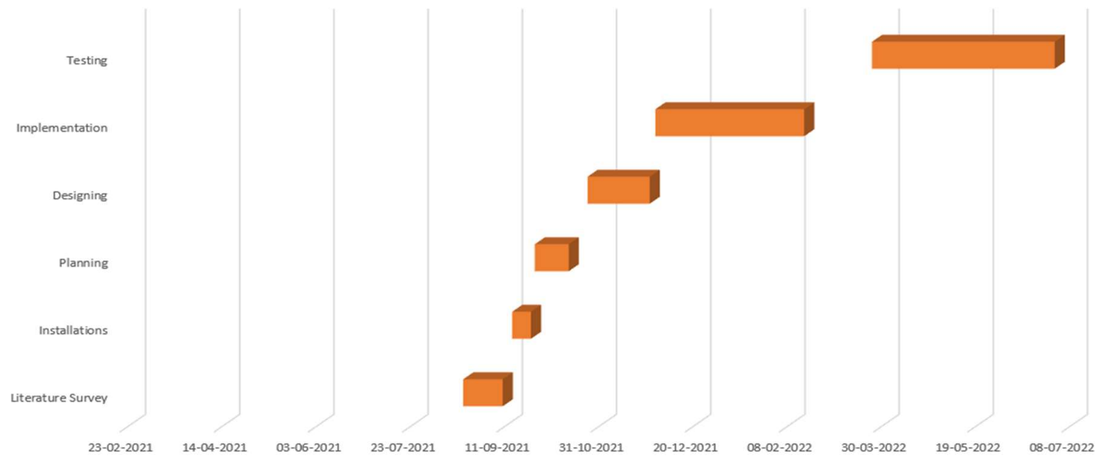


Fig 1. Gantt Chart

The project work started from the month of August, 2021. First, the basics were being checked and via the literature surveys the concept of the project was understood. Gradually the installations, plannings and designs were being done. The implementation was started from November and continued till mid February. The final testing was done and the project was successfully completed before the assigned schedule.

2. LITERATURE SURVEY

To understand the implementation of the routing algorithm, at first, we got ourselves acquainted with cloud computing and network-related terms like:

2.1 **Percolation Centrality(PC) or Betweenness Centrality (BC)**: These are measures the of importance of a graph in terms of its connectivity with other nodes of the graph. Higher values of PC or BC indicate that from a given node with highest value of PC or BC a procedure/mobile code/rumor/disease will spread or percolate faster to the entire network through its different connections than through a node with low values of PC or BC. [4]

2.2 **Flooding**: It occurs when source packets (without routing data) are transmitted to all attached network nodes. Because flooding uses every path in the network, the shortest path is also used. [7]

2.3 **Terms and Definitions**:

1. **Cloud**

2. **Types of Cloud**

3. **Google Cloud Platform (GCP)**

4. **Virtual Private Cloud (VPC)**

5. **Google Compute Engine (GCE)**

6. **Routing Algorithm**

7. **Internet Protocol (IP)**

1. **Cloud**: Cloud refers to the software and service that runs on the Internet, instead of locally on the computer. With the help of the cloud, we can access any data or application over the internet. Cloud not only handles data storage remotely but it also protects and recovers all crashed or lost data.

2. **Types of Cloud**: There are 4 main types of cloud computing: private clouds, public clouds, hybrid clouds, and multi-clouds.

Public cloud: A public cloud provides cloud services over the internet to the public. These services are available as pay-as-you-go billing models. It is meant to serve multiple users, not a single customer.

Private clouds: They are distributed systems that work on private infrastructure and provide the users with dynamic provisioning of computing resources.

Hybrid cloud: It is a heterogeneous distributed system formed by combining facilities of the public cloud and private cloud.

Multi-cloud: It is a cloud approach made up of more than 1 cloud service, from more than 1 cloud vendor—public or private. All hybrid clouds are multi-clouds, but not all multi-clouds are hybrid clouds. [2]

3. **Google Cloud Platform (GCP):** It is offered by Google and is a suite of cloud computing services that run on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail, Google Drive, and YouTube. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning.

4. **Virtual Private Cloud:** A virtual private cloud (VPC) is an on-demand configurable pool of shared resources allocated within a public cloud environment, providing a certain level of isolation between the different organizations (denoted as users hereafter) using the resources.

5. **Google Compute Engine (GCE):** It enables users to launch virtual machines (VMs) on demand. VMs can be launched from the standard images or custom images created by users.

6**. Routing Algorithm**: A routing algorithm is a procedure that lays down the route or path to transfer data packets from the source to the destination. They help in directing Internet traffic efficiently. The routing algorithm mathematically computes the best path, that is, "least-cost path" that the packet can be routed through. [6]

7**. Internet Protocol**: The Internet Protocol (IP) is the network layer communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking and essentially establishes the Internet. Each network has a unique IP address. [6]

3. CONCEPTS AND PROBLEM ANALYSIS

Our project is an enhancement of the controlled flooding algorithm by adding on the concept of percolation centrality to broadcast the messages via the adjacent nodes.

The routing algorithm that we made is basically a broadcast algorithm which determines the fastest route to deliver a message. That is, we determine the node which can pass on the resource from source to destination at the least amount of time. Our routing algorithm is similar to the controlled flooding algorithm. We enhance the flooding algorithm using the concepts of percolation centrality and recovery procedure. We send a message that percolates via the nodes of our network.

Percolation Centrality (PC) or Betweenness Centrality (BC) defines the measure of connectivity of a node with other nodes in a graph. Higher the value of PC or BC higher will be the rate of spreading of a procedure/code/rumor/disease throughout the network with respect to the nodes having lower value of PC or BC.[4]

The topologies of some networks are so dynamic and unstable that no reliable routes can be found in the network. Although not normally considered to be a routing algorithm, network flooding is often the only viable routing option in these situations. It has one interesting property that distinguishes it from other routing protocols: it is able to route packets without any knowledge of the network topology, so no control packets ever need to be exchanged in the network. [3]

In flooding, when a node wants to send a packet to a destination, it broadcasts the packet to all of its neighbours. Whenever any node receives any packet, it simply rebroadcasts it. This process continues until the packet has been rebroadcasted throughout the entire network, resulting in every node in the network receiving a copy of the packet. To prevent the packet from being rebroadcasted forever, nodes do not rebroadcast packets that they have already seen before. Often a random delay is inserted between packet broadcasts to lower the potential of packet collisions from two nodes receiving and rebroadcasting the same packet simultaneously. Despite its poor scalability and inefficient bandwidth usage, flooding is useful to study because it is an extreme example of how to minimize the effect of topology changes on routing. By completely eliminating any reliance on knowledge about the network topology, flooding performance does not degrade with increased node mobility. [4]

Broadcast Routing Algorithm:

• Most straightforward way: N-way-unicast – no new network-layer routing protocol, packet-duplication, or forwarding functionality is needed.

– Drawbacks:

• Inefficiency: As it would be more efficient for the network nodes themselves (rather than just the source node) to create duplicate copies of a packet

• Unrealistic assumption: An implicit assumption of N-way-unicast is that broadcast recipients, and their addresses, are known to the sender.

• More overhead: it would be unwise (at best!) to rely on the unicast routing infrastructure to achieve broadcast.

Broadcast Algorithms examples are:

1. **Controlled Flooding**
2. **Uncontrolled Flooding**

Controlled Flooding:

Key to avoiding a broadcast storm – for a node to judiciously choose when to flood and when not to flood a packet, that is, controlled way of flooding.

1) Sequence-number-controlled flooding
2) Reverse path forwarding (RPF)

Uncontrolled Flooding:

• Most obvious technique for achieving broadcast is a flooding.

– Source node sends a copy of the packet to all of its neighbours.

– When a node receives a broadcast packet, it duplicates the packet and forwards it to all of its neighbours (except the neighbour from which it received the packet).

– this scheme will eventually deliver a copy of the broadcast packet to all nodes if they are connected

• Disadvantages:

– If the graph has cycles, then one or more copies of each broadcast packet will cycle indefinitely.

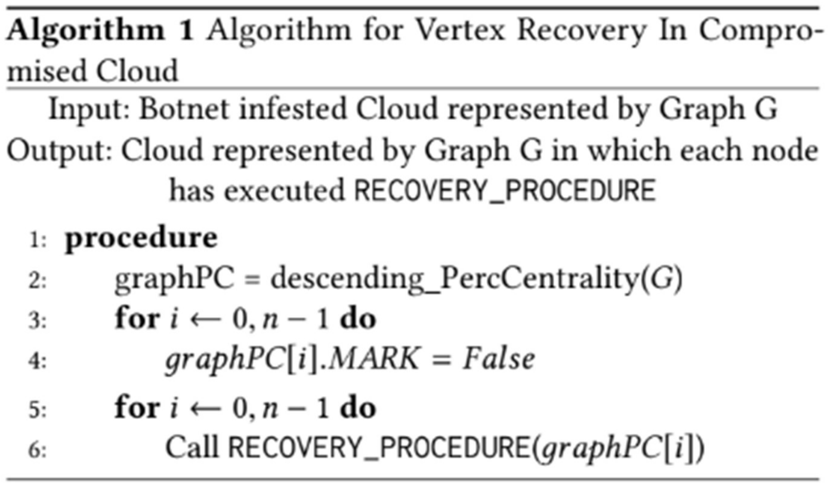
– When a node is connected to more than two other nodes, then it could result in broadcast storm (resulting from the endless multiplication of broadcast packets). [5]

4. DESIGN AND METHODOLOGY

4.1. Algorithm

The routing algorithm we are implementing is based on the Controlled Flooding algorithm which can distribute a resource in the entire network while taking the minimum time. For this time optimization and the enhancement of the flooding algorithm, we have used a property of a node in a graph called Betweenness Centrality or Percolation Centrality.

The algorithm is divided into two parts.



Algorithm 1 Algorithm for Vertex Recovery In Compromised Cloud

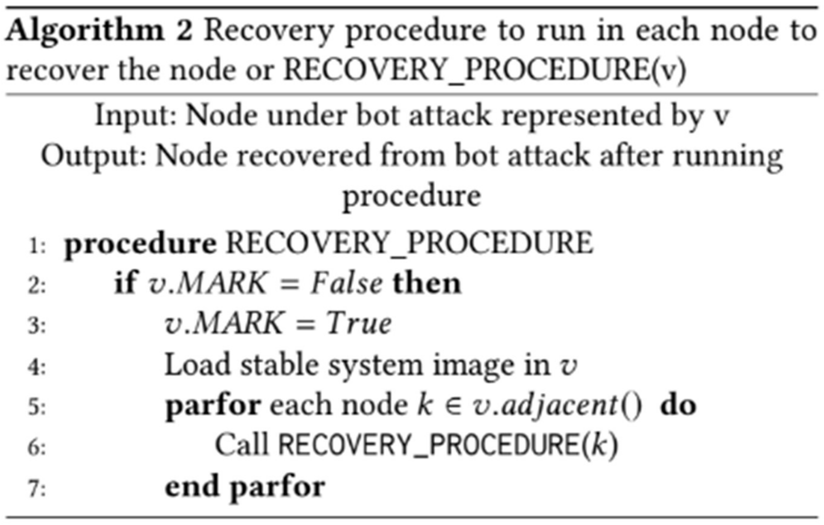
Input: Botnet infested Cloud represented by Graph G
Output: Cloud represented by Graph G in which each node has executed RECOVERY_PROCEDURE

```
1: procedure
2:     graphPC = descending_PercCentrality($G$)
3:     for $i \leftarrow 0, n-1$ do
4:         $graphPC[i].MARK = False$
5:     for $i \leftarrow 0, n-1$ do
6:         Call RECOVERY_PROCEDURE($graphPC[i]$)
```

Fig 2. Algorithm1 [Source: Bibliography item no. 2]

- **Explanation of Algorithm 1**

  Algorithm 1 sorts the nodes in the network in the descending order of their percolation centrality and calls the flooding procedure looping on the nodes in the sorted order i.e., it basically starts the routing from the node having highest percolation centrality. Also, it is responsible for making sure that every node in the network has received the resource. In case of link breakage or any disconnectivity in the network it will take care of that scenario and forward the packet to the isolated node or start the routing in the disconnected component.



Algorithm 2 Recovery procedure to run in each node to recover the node or RECOVERY_PROCEDURE(v)

Input: Node under bot attack represented by v
Output: Node recovered from bot attack after running procedure

```
1: procedure RECOVERY_PROCEDURE
2:     if $v.MARK = False$ then
3:         $v.MARK = True$
4:         Load stable system image in $v$
5:         parfor each node $k \in v.adjacent()$ do
6:             Call RECOVERY_PROCEDURE($k$)
7:         end parfor
```

Fig 3. Algorithm2 [Source: Bibliography item no. 2]

- **Explanation of Algorithm 2**

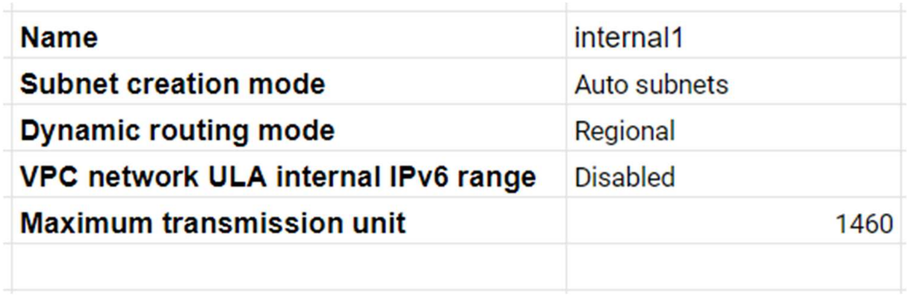
  Algorithm 2 states the design of the controlled flooding algorithm i.e., upon getting a broadcast packet, first a node will check if it has already received the packet or not. If already received, it will simply discard that packet, otherwise it will accept, duplicate and forward the packet to its adjacent nodes.

4.2. Platform Configurations

Our project is done in the Google Cloud Platform (GCP). We have mainly used two services provided by GCP:

- **Virtual Private Cloud (VPC)**

  We have configured the VPC network with the following specifications:

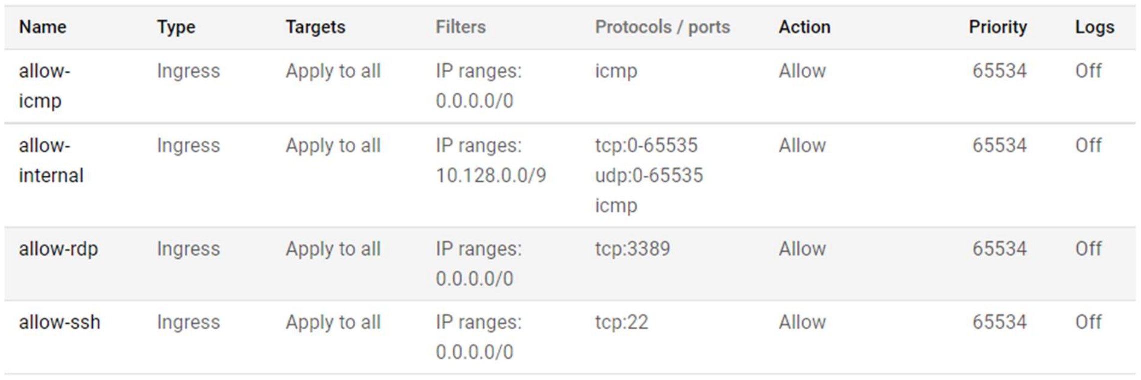


Name	internal1
Subnet creation mode	Auto subnets
Dynamic routing mode	Regional
VPC network ULA internal IPv6 range	Disabled
Maximum transmission unit	1460

Fig 4. VPC-Configuration

The firewall rules associated with this VPC network are:

- allowing ssh
- allowing icmp
- allowing rdp connections
- allowing internal connections between virtual machines in the network



Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Logs
allow-icmp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	icmp	Allow	65534	Off
allow-internal	Ingress	Apply to all	IP ranges: 10.128.0.0/9	tcp:0-65535 udp:0-65535 icmp	Allow	65534	Off
allow-rdp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:3389	Allow	65534	Off
allow-ssh	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:22	Allow	65534	Off

Fig 5. FirewallRules

- **Compute Engine**

  For the implementation and testing of our routing algorithm we have designed a network with seven nodes and the nodes are connected in such a way that different nodes have different percolation centrality values.

  The conceptual diagram of our network is:

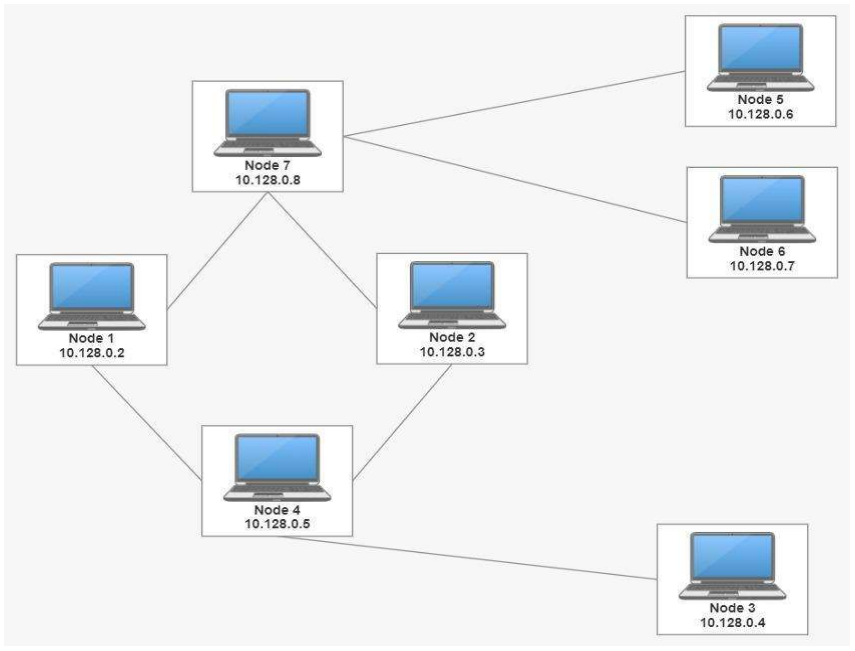
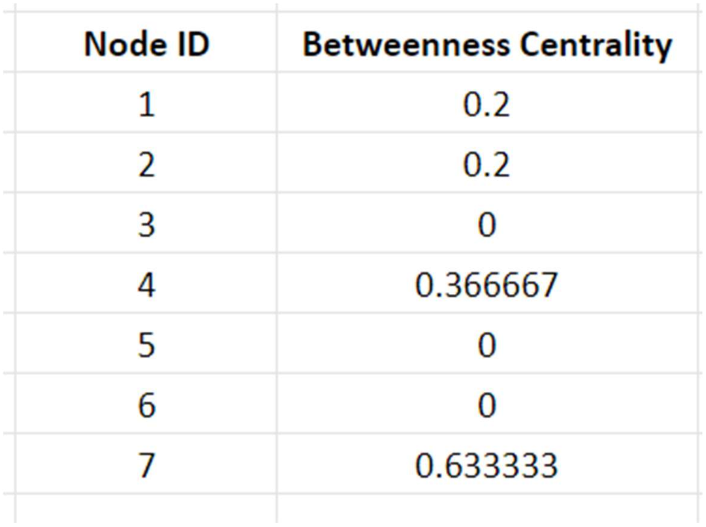


Fig 6. LAN Diagram

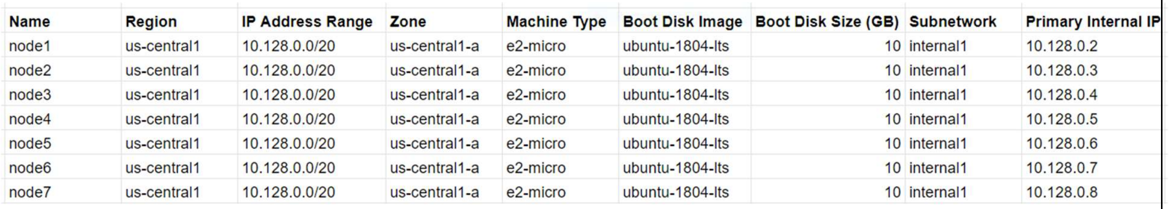
Cytoscape 3.5.1 can be used to find BC values of the network. On drawing the graph in Cytoscape it gives the BC values of the nodes. Betweenness centrality of each nodes in this graph is:



Node ID	Betweenness Centrality
1	0.2
2	0.2
3	0
4	0.366667
5	0
6	0
7	0.633333

Fig 7. BC-7

We have used GCP's Compute Engine to configure these nodes. It enables users to launch virtual machines (VMs) on demand. The specifications of each VM is as follows:



Name	Region	IP Address Range	Zone	Machine Type	Boot Disk Image	Boot Disk Size (GB)	Subnetwork	Primary Internal IP
node1	us-central1	10.128.0.0/20	us-central1-a	e2-micro	ubuntu-1804-lts	10	internal1	10.128.0.2
node2	us-central1	10.128.0.0/20	us-central1-a	e2-micro	ubuntu-1804-lts	10	internal1	10.128.0.3
node3	us-central1	10.128.0.0/20	us-central1-a	e2-micro	ubuntu-1804-lts	10	internal1	10.128.0.4
node4	us-central1	10.128.0.0/20	us-central1-a	e2-micro	ubuntu-1804-lts	10	internal1	10.128.0.5
node5	us-central1	10.128.0.0/20	us-central1-a	e2-micro	ubuntu-1804-lts	10	internal1	10.128.0.6
node6	us-central1	10.128.0.0/20	us-central1-a	e2-micro	ubuntu-1804-lts	10	internal1	10.128.0.7
node7	us-central1	10.128.0.0/20	us-central1-a	e2-micro	ubuntu-1804-lts	10	internal1	10.128.0.8

Fig 8. VM-Configuration

4.4. Physical Network Setup

Since we have launched all the seven virtual machines in the same subnet 10.128.0.0/20 (us-central1) and due to the firewall rules associated with the networks, all the nodes are connected to each other by default i.e., it forms a complete graph with seven nodes. So to achieve our conceptual network we need to block the unnecessary connections. For this there can be two approaches:

- writing firewall rules to deny connections from certain vm and specify its target using target tags
- programmatic approach

We have followed the second method, a programmatic approach. For this, in each vm, we have maintained a python list containing the IP addresses of its destination nodes. The implementation of the flooding algorithm will follow this list to forward packets to its adjacent nodes.

4.4. Prerequisites

We have implemented the above discussed routing algorithm using python socket programming. For the successful running of the program we need to install some libraries as prerequisites. Those are as follows:

- **Install pip3**
  - **Step-1 Update System:** sudo apt-get update
  - **Step-2 Install pip3:** sudo apt-get -y install python3-pip
  - **Step-3 Verification:** pip3 --version

- **Install the tqdm library**

  The command to install tqdm library: pip3 install tqdm

5. SAMPLE CODES

5.1. Forward Logic for start-node

Algorithm 1 is implemented in the start-node. It is a separate node, not included in the conceptual network diagram and acts as an administrative system. It maintains a python dictionary, ipPercCentralityMap with IP addresses as key and percolation centrality values as their corresponding values. The code will sort them in the descending order of the percolation centrality and start the routing of a resource from the node having the highest percolation centrality.

The code is as follows:

```
import socket

import threading

import tqdm

import os

import sys

import time

node = "10.128.0.9"

port = 4444


SEPARATOR = "<<SEPARATOR>>"

BUFFER_SIZE = 4096 # send 4096 bytes each time step


# the name of file we want to send

if(len(sys.argv) < 2):

    print("Resource not specified...")

    exit()

else:

    filename = sys.argv[1]

# get the file size

filesize = os.path.getsize(filename)


def forward():

    for dest in sortedIpList:

        forward_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        print(f"[+] Connecting to {dest}:{port} ...")
        forward_conn.connect((dest, port))
        print("[+] Connected.")
        # send the filename and filesize
        forward_conn.send(f"{filename}{SEPARATOR}{filesize}{SEPARATOR}".encode())
        # start sending the file
        progress = tqdm.tqdm(range(filesize), f"Sending {filename}", unit="B", unit_scale=True,
unit_divisor=1024)
        with open(filename, "rb") as f:
            while True:
                # read the bytes from the file
                bytes_read = f.read(BUFFER_SIZE)
                if not bytes_read:
                    # file transmitting is done
                    break
                # we use sendall to assure transimission in
                # busy networks
                forward_conn.sendall(bytes_read)
                # update the progress bar
                progress.update(len(bytes_read))
    except:
        print(f"{dest} already received message...")
        continue
    finally:
        forward_conn.close()
    time.sleep(5)


ipPercCentralityMap = {"10.128.0.2":0.2,"10.128.0.3":0.2 ,"10.128.0.4":0
,"10.128.0.5":0.366667,"10.128.0.6":0,"10.128.0.7":0,"10.128.0.8":0.633333}

sortedIpList = list(dict(sorted(ipPercCentralityMap.items(), key=lambda item:
item[1],reverse=True)).keys())
```

```
forward()
```

5.2. Forward & Accept Logic for Other Nodes

Every other node except start-node has the same forward and accept logic. Through these two logics the controlled flooding algorithm is implemented.

The code is as follows:

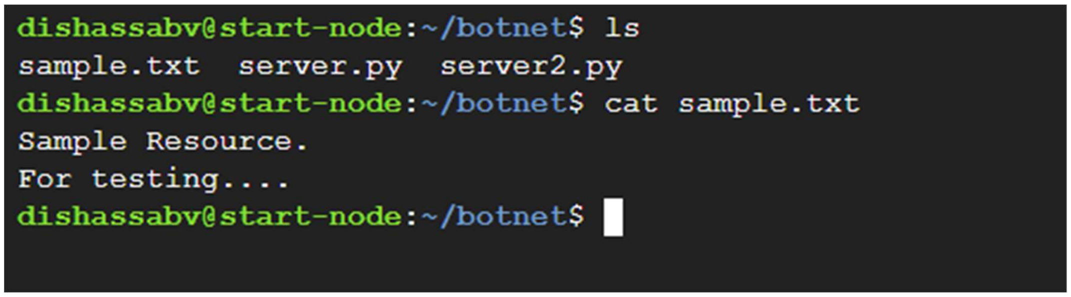
```
from datetime import datetime
import socket
import threading, queue
import tqdm
import os
node = "10.128.0.2" # ip address of the particular node
port = 4444

# receive 4096 bytes each time
BUFFER_SIZE = 4096
SEPARATOR = "<<SEPARATOR>>"

# Adjacent nodes to this node
destinations = ["10.128.0.5","10.128.0.8"]  # destination ip addresses of the particular node
q = queue.Queue()
recovered = True
def accept():
    accept_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # bind the socket to our local address
    accept_conn.bind((node, port))
    # enabling our server to accept connections
    # 5 here is the number of unaccepted connections that
    # the system will allow before refusing new connections
    accept_conn.listen(5)
    print(f"[*] Listening as {node}:{port}")
```

```
global recovered
    while recovered:
        # accept connection if there is any
        conn, addr = accept_conn.accept()
        print(f"[+] {addr} is connected.")
        # receive the file infos
        # receive using client socket, not server socket
        received = conn.recv(BUFFER_SIZE).decode()
        filename, filesize, message = received.split(SEPARATOR)
        # remove absolute path if there is
        filename = os.path.basename(filename)
        # convert to integer
        filesize = int(filesize)
        # start receiving the file from the socket
        # and writing to the file stream
        progress = tqdm.tqdm(range(filesize), f"Receiving {filename}", unit="B", unit_scale=True,
unit_divisor$
        with open(filename, "wb") as f:
            if(message != ""):
                f.write(message.encode())
            while True:
                # read 1024 bytes from the socket (receive)
                bytes_read = conn.recv(BUFFER_SIZE)
                if not bytes_read:
                    # nothing is received
                    # file transmitting is done
                    break
                # write to the file the bytes we just received
                f.write(bytes_read)
                # update the progress bar
                progress.update(len(bytes_read))
        print(f"[+] Message received at [{datetime.utcnow()}]")
        for dest in destinations:
            q.put((conn, addr, filename, filesize, dest))
        recovered = False
        conn.close()
    accept_conn.close()
def forward():
    global recovered
    print("Forward connections listening...")
    while True:
        if q.empty()==True and recovered==False:
            break
        curr = q.get()
        forward_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            print(f"Forwarding to {curr[4]}:{port}")
            forward_conn.connect((curr[4], port))
            # send the filename and filesize
            filename = curr[2]
            filesize = curr[3]
            forward_conn.send(f"{filename}{SEPARATOR}{filesize}{SEPARATOR}".encode())
            # start sending the file
            progress = tqdm.tqdm(range(filesize), f"Sending {filename}", unit="B", unit_scale=True, 
unit_divis$
            with open(filename, "rb") as f:
                while True:
                    # read the bytes from the file
                    bytes_read = f.read(BUFFER_SIZE)
                    if not bytes_read:
                        # file transmitting is done
                        break
                    # we use sendall to assure transimission in
                    # busy networks
                forward_conn.sendall(bytes_read)
                # update the progress bar
                progress.update(len(bytes_read))
        except:
            print(f"{curr[4]} already received message")
            continue
        finally:
            forward_conn.close()
if __name__ == "__main__":
    t1 = threading.Thread(target=accept)
    t2 = threading.Thread(target=forward)
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

6. TESTING, RESULTS, DISCUSSION ON RESULTS

6.1. Testing

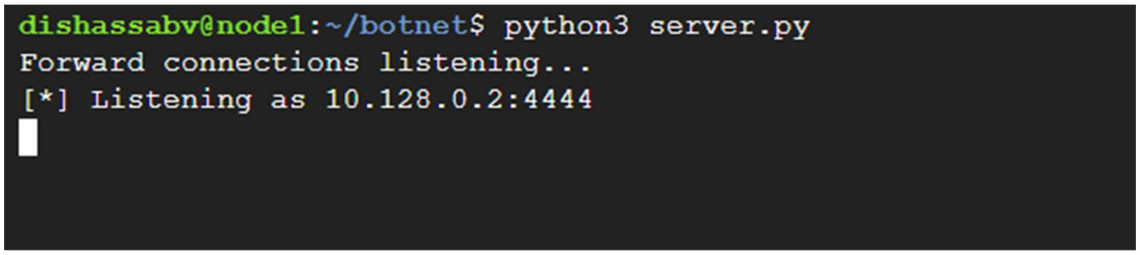
We will distribute a text file "sample.txt" throughout the entire network. Initially it was present in start-node only and its contents are:



```
dishassabv@start-node:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@start-node:~/botnet$ cat sample.txt
Sample Resource.
For testing....
dishassabv@start-node:~/botnet$ 
```

Fig 8. SampleResource

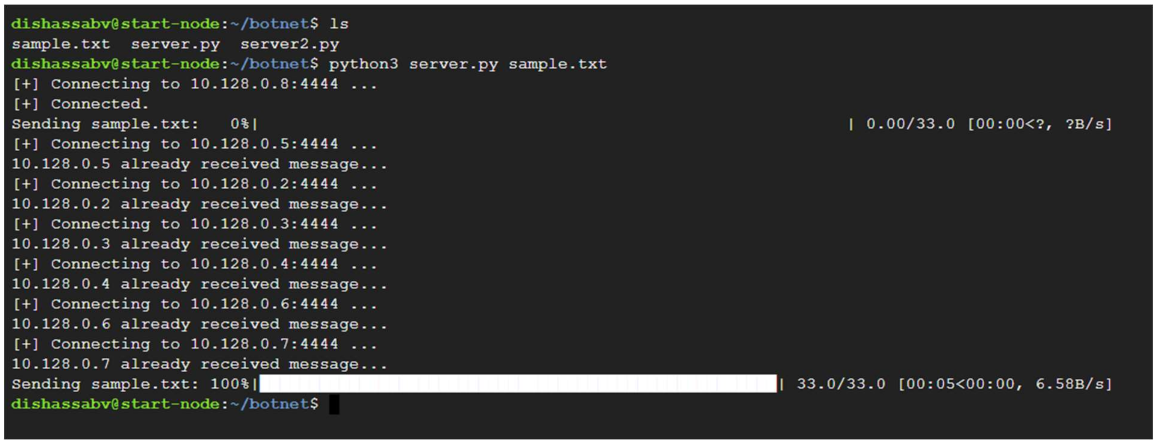
- Now first we need to run the python script (server.py) in all the nodes from node 1 to 7 and they will be in a waiting or listening state waiting for the client to connect(Fig listening-socket).



```
dishassabv@node1:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.2:4444
```

Fig 9. Listening-socket

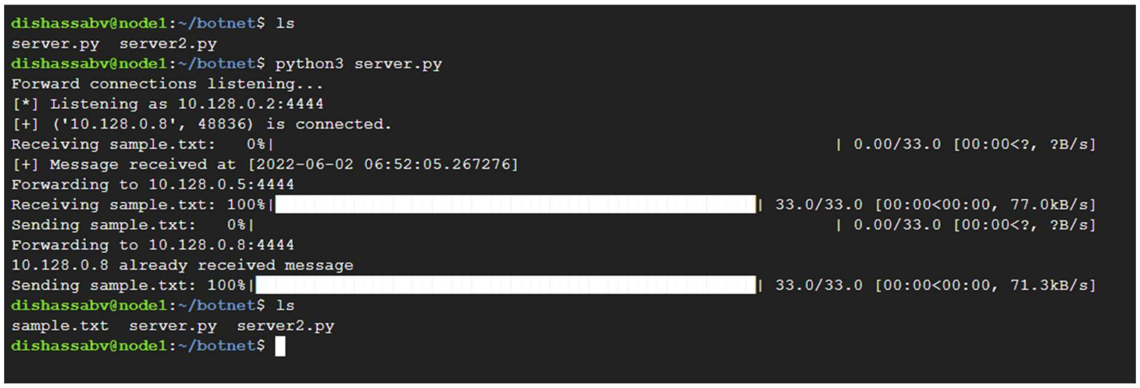
- Finally run the python script in start-node specifying the resource to be distributed.



```
dishassabv@start-node:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@start-node:~/botnet$ python3 server.py sample.txt
[+] Connecting to 10.128.0.8:4444 ...
[+] Connected.
Sending sample.txt:   0%|                                                    | 0.00/33.0 [00:00<?, ?B/s]
[+] Connecting to 10.128.0.5:4444 ...
10.128.0.5 already received message...
[+] Connecting to 10.128.0.2:4444 ...
10.128.0.2 already received message...
[+] Connecting to 10.128.0.3:4444 ...
10.128.0.3 already received message...
[+] Connecting to 10.128.0.4:4444 ...
10.128.0.4 already received message...
[+] Connecting to 10.128.0.6:4444 ...
10.128.0.6 already received message...
[+] Connecting to 10.128.0.7:4444 ...
10.128.0.7 already received message...
Sending sample.txt: 100%|████████████████████████████████████████| 33.0/33.0 [00:05<00:00, 6.58B/s]
dishassabv@start-node:~/botnet$ 
```

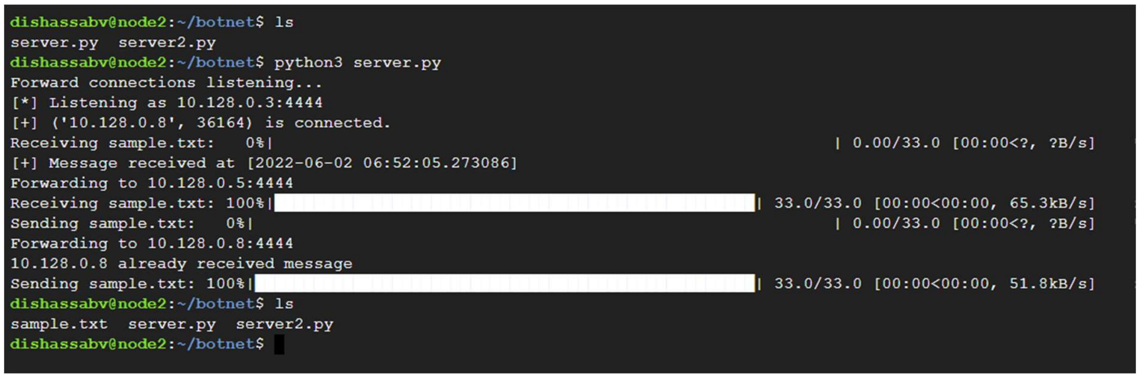
Fig 10. Start-node-log

This will distribute “sample.txt” throughout the entire network and the output in each node is as follows:



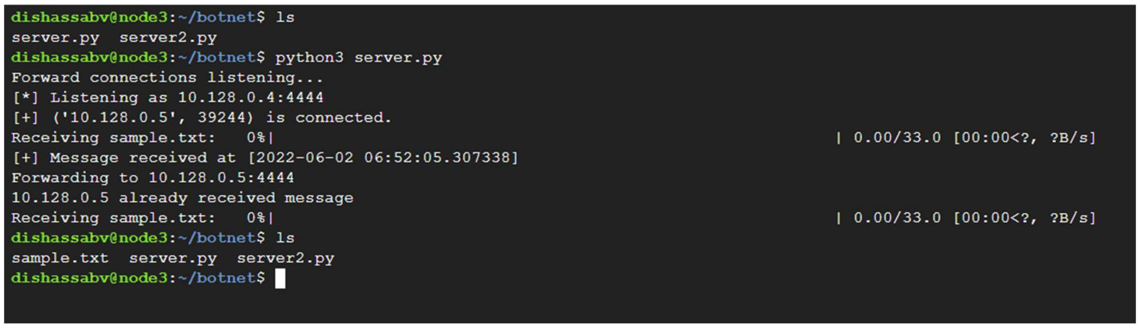
```
dishassabv@node1:~/botnet$ ls
server.py  server2.py
dishassabv@node1:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.2:4444
[+] ('10.128.0.8', 48836) is connected.
Receiving sample.txt:   0%|                                                   | 0.00/33.0 [00:00<?, ?B/s]
[+] Message received at [2022-06-02 06:52:05.267276]
Forwarding to 10.128.0.5:4444
Receiving sample.txt: 100%|███████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 77.0kB/s]
Sending sample.txt:   0%|                                                     | 0.00/33.0 [00:00<?, ?B/s]
Forwarding to 10.128.0.8:4444
10.128.0.8 already received message
Sending sample.txt: 100%|█████████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 71.3kB/s]
dishassabv@node1:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node1:~/botnet$
```

Fig 11. Node1-log



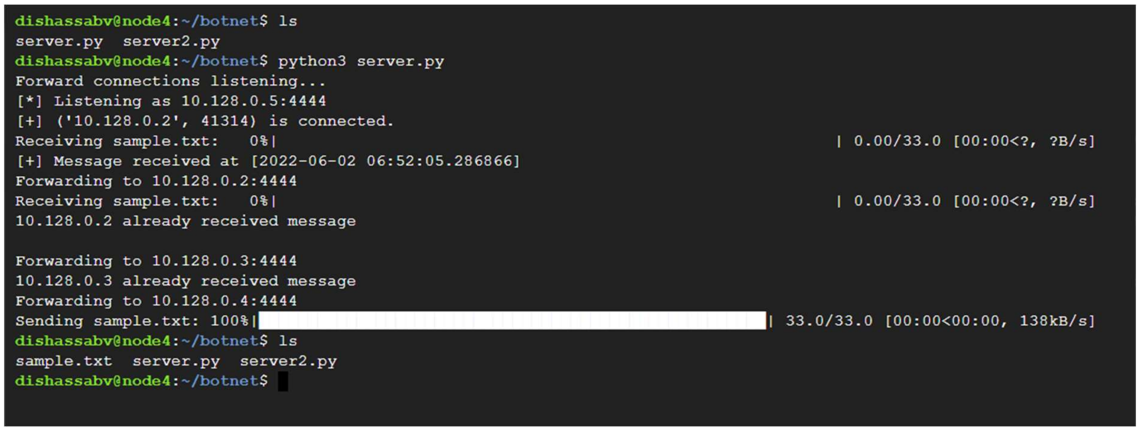
```
dishassabv@node2:~/botnet$ ls
server.py  server2.py
dishassabv@node2:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.3:4444
[+] ('10.128.0.8', 36164) is connected.
Receiving sample.txt:   0%|                                                   | 0.00/33.0 [00:00<?, ?B/s]
[+] Message received at [2022-06-02 06:52:05.273086]
Forwarding to 10.128.0.5:4444
Receiving sample.txt: 100%|███████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 65.3kB/s]
Sending sample.txt:   0%|                                                     | 0.00/33.0 [00:00<?, ?B/s]
Forwarding to 10.128.0.8:4444
10.128.0.8 already received message
Sending sample.txt: 100%|█████████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 51.8kB/s]
dishassabv@node2:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node2:~/botnet$
```

Fig 12. Node2-log



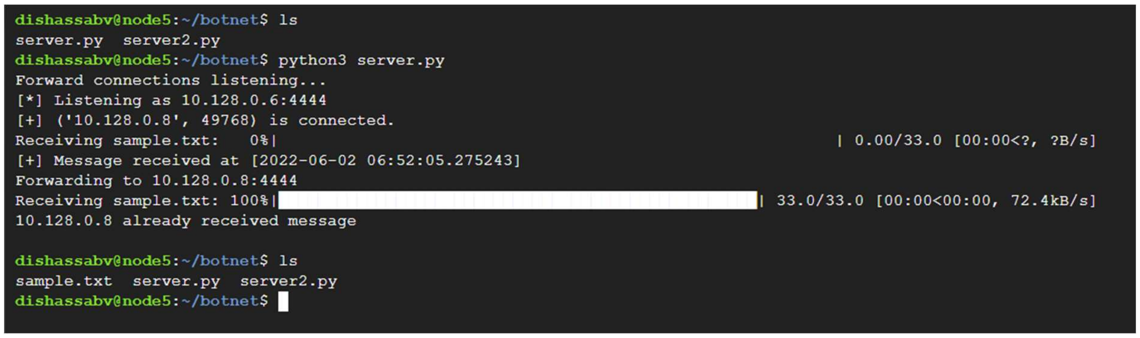
```
dishassabv@node3:~/botnet$ ls
server.py  server2.py
dishassabv@node3:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.4:4444
[+] ('10.128.0.5', 39244) is connected.
Receiving sample.txt:   0%|                                                   | 0.00/33.0 [00:00<?, ?B/s]
[+] Message received at [2022-06-02 06:52:05.307338]
Forwarding to 10.128.0.5:4444
10.128.0.5 already received message
Receiving sample.txt:   0%|                                                   | 0.00/33.0 [00:00<?, ?B/s]
dishassabv@node3:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node3:~/botnet$
```

Fig 13. Node3-log



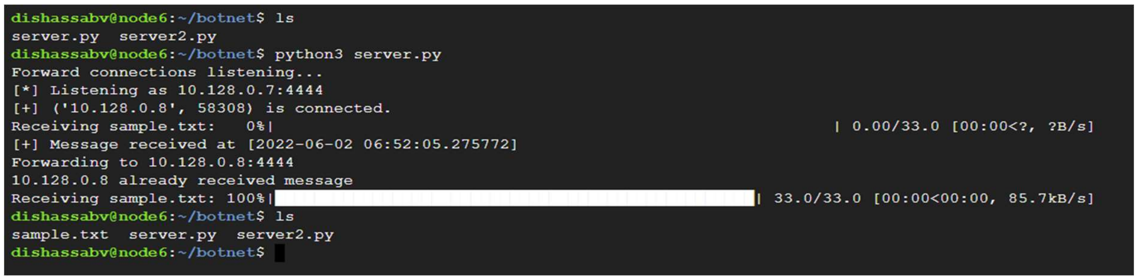
```
dishassabv@node4:~/botnet$ ls
server.py  server2.py
dishassabv@node4:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.5:4444
[+] ('10.128.0.2', 41314) is connected.
Receiving sample.txt:   0%|                                                         | 0.00/33.0 [00:00<?, ?B/s]
[+] Message received at [2022-06-02 06:52:05.286866]
Forwarding to 10.128.0.2:4444
Receiving sample.txt:   0%|                                                         | 0.00/33.0 [00:00<?, ?B/s]
10.128.0.2 already received message

Forwarding to 10.128.0.3:4444
10.128.0.3 already received message
Forwarding to 10.128.0.4:4444
Sending sample.txt: 100%|████████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 138kB/s]
dishassabv@node4:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node4:~/botnet$
```

Fig 14. Node4-log



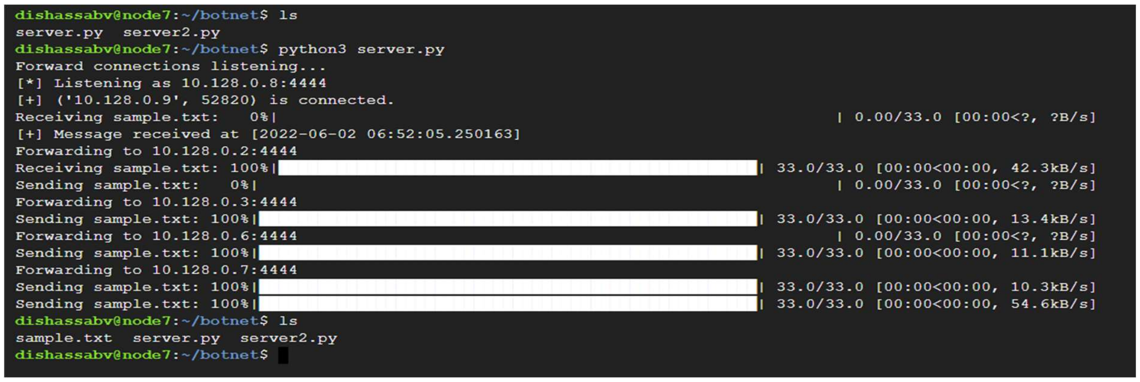
```
dishassabv@node5:~/botnet$ ls
server.py  server2.py
dishassabv@node5:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.6:4444
[+] ('10.128.0.8', 49768) is connected.
Receiving sample.txt:   0%|                                                         | 0.00/33.0 [00:00<?, ?B/s]
[+] Message received at [2022-06-02 06:52:05.275243]
Forwarding to 10.128.0.8:4444
Receiving sample.txt: 100%|███████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 72.4kB/s]
10.128.0.8 already received message

dishassabv@node5:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node5:~/botnet$
```

Fig 15. Node5-log



```
dishassabv@node6:~/botnet$ ls
server.py  server2.py
dishassabv@node6:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.7:4444
[+] ('10.128.0.8', 58308) is connected.
Receiving sample.txt:   0%|                                                         | 0.00/33.0 [00:00<?, ?B/s]
[+] Message received at [2022-06-02 06:52:05.275772]
Forwarding to 10.128.0.8:4444
10.128.0.8 already received message
Receiving sample.txt: 100%|███████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 85.7kB/s]
dishassabv@node6:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node6:~/botnet$
```

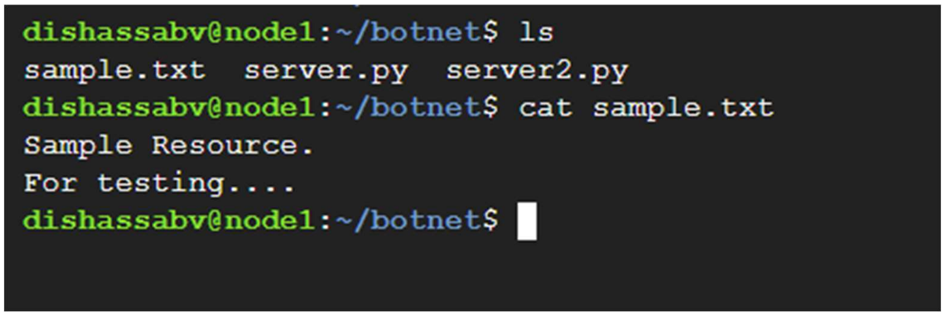
Fig 16. Node6-log



```
dishassabv@node7:~/botnet$ ls
server.py  server2.py
dishassabv@node7:~/botnet$ python3 server.py
Forward connections listening...
[*] Listening as 10.128.0.8:4444
[+] ('10.128.0.9', 52820) is connected.
Receiving sample.txt:   0%|                                                         | 0.00/33.0 [00:00<?, ?B/s]
[+] Message received at [2022-06-02 06:52:05.250163]
Forwarding to 10.128.0.2:4444
Receiving sample.txt: 100%|███████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 42.3kB/s]
Sending sample.txt:   0%|                                                           | 0.00/33.0 [00:00<?, ?B/s]
Forwarding to 10.128.0.3:4444
Sending sample.txt: 100%|█████████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 13.4kB/s]
Forwarding to 10.128.0.6:4444                                                        | 0.00/33.0 [00:00<?, ?B/s]
Sending sample.txt: 100%|█████████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 11.1kB/s]
Forwarding to 10.128.0.7:4444
Sending sample.txt: 100%|█████████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 10.3kB/s]
Sending sample.txt: 100%|█████████████████████████████████████████████| 33.0/33.0 [00:00<00:00, 54.6kB/s]
dishassabv@node7:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node7:~/botnet$
```

Fig 17. Node7-log

The content of “sample.txt” after distribution is the same, meaning the implementation of the algorithm is properly distributing the resource throughout the entire network.



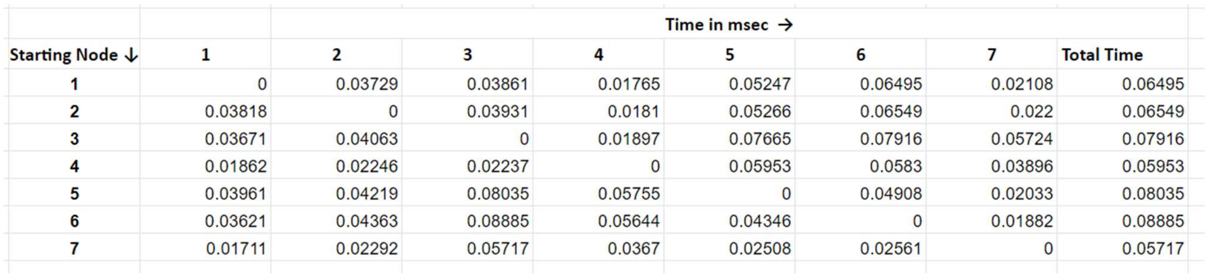
```
dishassabv@node1:~/botnet$ ls
sample.txt  server.py  server2.py
dishassabv@node1:~/botnet$ cat sample.txt
Sample Resource.
For testing....
dishassabv@node1:~/botnet$ 
```

Fig 18. Received-resource

6.2. Results & Discussion

Each node logs the timestamp at when it received the resource. From this time, we can reduce some useful observations.

We ran the routing procedure, starting from each node separately to distribute the same resource and gathered the reading of the time delay for each node to receive the resource starting from various nodes.



					Time in msec →			
Starting Node ↓	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total Time**
1	0	0.03729	0.03861	0.01765	0.05247	0.06495	0.02108	0.06495
2	0.03818	0	0.03931	0.0181	0.05266	0.06549	0.022	0.06549
3	0.03671	0.04063	0	0.01897	0.07665	0.07916	0.05724	0.07916
4	0.01862	0.02246	0.02237	0	0.05953	0.0583	0.03896	0.05953
5	0.03961	0.04219	0.08035	0.05755	0	0.04908	0.02033	0.08035
6	0.03621	0.04363	0.08885	0.05644	0.04346	0	0.01882	0.08885
7	0.01711	0.02292	0.05717	0.0367	0.02508	0.02561	0	0.05717

Fig 19. Delay-report

From this delay report (Fig delay-report) we can evaluate the total time required to complete the distribution for each starting node and can compare the relationship between total delay and betweenness centrality by plotting a scatter-plot.

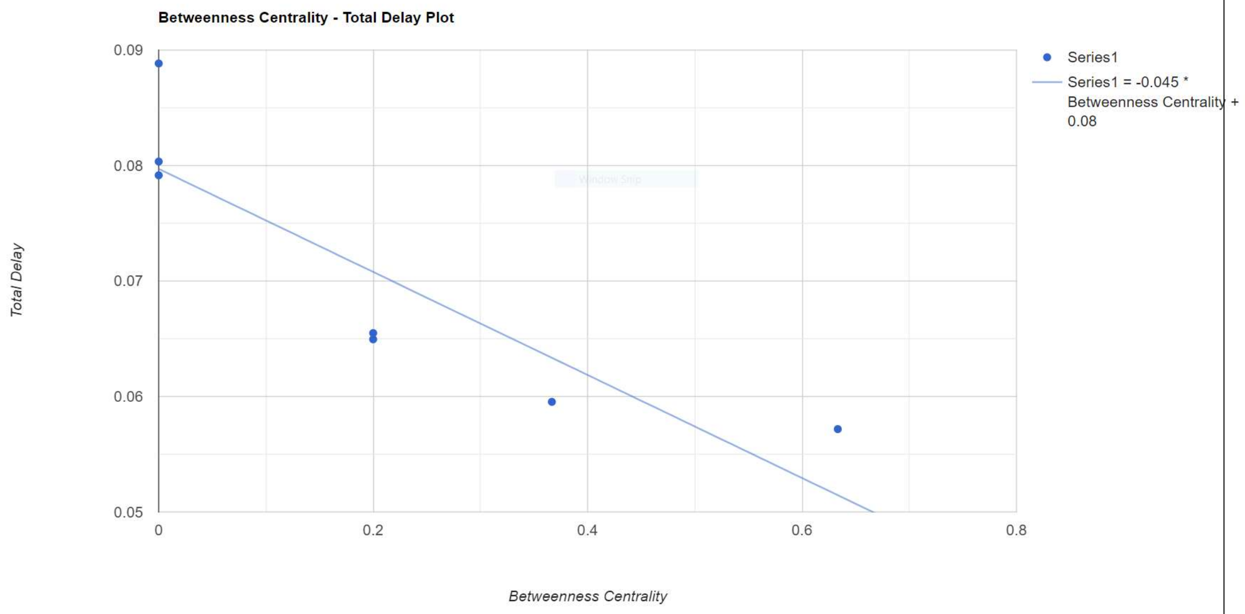


Fig 20. Scatter-plot

From the above scatter-plot we can clearly conclude that, the total time required to distribute the resource in the entire network decreases as the betweenness centrality of the starting node increases.

Hence, the enhancement we made to the controlled flooding algorithm, will definitely help us to optimize the time required to distribute a resource throughout the entire network.

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

Flooding is the most obvious technique for achieving broadcast. Source node sends a copy of the packet to all of its neighbours. When a node receives a broadcast packet, it duplicates the packet and forwards it to all of its neighbours (except the neighbour from which it received the packet). But this uncontrolled way of flooding can lead to broadcast storms if the network has cycles in it. To avoid this broadcast storm, we use a controlled flooding mechanism where there is a control logic that prevents a node from accepting the broadcast packet if it has already received it once. To boost this controlled flooding mechanism and optimize the algorithm in such a way that it will deliver a resource in the entire network in the least amount of time, we introduced the property betweenness centrality of a node in any graph. Higher values of PC or BC indicate that from a given node with the highest value of PC or BC a procedure/mobile code/rumor/disease will spread or percolate faster to the entire network through its different connections than through a node with low values of PC or BC.

From the testing and observations of our project, we can clearly conclude that the total time required to distribute the resource in the entire network decreases as the betweenness centrality of the starting node increases.

Hence, the enhancement we made to the controlled flooding algorithm, will definitely help us to optimize the time required to distribute a resource throughout the entire network.

6.2. Future Scope

There can be many enhancements made to our project and some of those future scopes can be:

- Introduction of automation to minimize the number of manual steps to run the scripts for each routing.
- Distributing resources with complex format

BIBLIOGRAPHY

[1] SamTeachYourself, Joe Casad, Pearson Education INC.,5th Edition.

[2] https://azure.microsoft.com/en-in/overview/future-of-cloud/

[3] De, S., Barik, M. S., & Banerjee, I. (2019, January). A percolation-based recovery mechanism for bot infected P2P cloud. In *Proceedings of the 20th International Conference on Distributed Computing and Networking* (pp. 474-479)

[4] Iu, M.Y. (2002). Selective flooding in ad hoc networks (Master's thesis, University of Waterloo)

[5] http://manaskhatua.github.io/courses/CS348/CN_4.6_Broadcast_Multicast_Routing.pdf

[6] Behrouz A Forouzan, Data Communications and Networking, 4th Edn, Ch.20 (pp. 579-602) , Ch.22 (pp. 647), Tata McGraw-Hill, 2011

[7] https://www.techopedia.com/definition/16190/flooding-networking