

MINOR PROJECT 2

END-TERM REPORT

ON

Data Communication Using Software-defined Wireless Network

Submitted By

Prakash Tiwari

Amrit Kumar

Akshit Chauhan

Gaurav Singh

500062611

500062268

500062444

500062611

Under the guidance of

Amit Singh

Assistant Professor, SoCSE



**Department of Cybernetics,
School of Computer Science
UNIVERSITY OF PETROLEUM AND ENERGY STUDIES
Dehradun-248007
April-2020**

Abstract

Software Defined Network is an emerging paradigm that promises to change this state of affairs, by breaking vertical integration, separating the network's control logic from the underlying routers and switches, promoting (logical) centralization of network control, and introducing the ability to program the network. SDN brings many well-known benefits such as manageability and adaptability, it also poses some challenges. Scalability becomes an issue in large scale networks, where the forwarding rules of single elements must be updated at a high pace by central controller. This problem can be solved using Dynamic Flow rule. Dynamic Flow rules enables network elements to change their forwarding behaviour locally according to pre-defined instruction set up by central controller.

Introduction:

1. SDN:

Software-Defined Networking (SDN) is a network architecture approach that enables the network to be intelligently and centrally controlled, or 'programmed,' using software applications. This helps operators manage the entire network consistently and holistically, regardless of the underlying network technology. SDN enables the programming of network behaviour in a centrally controlled manner through software applications. By opening up traditionally closed network platforms and implementing a common SDN control layer, operators can manage the entire network and its devices consistently, regardless of the complexity of the underlying network technology.

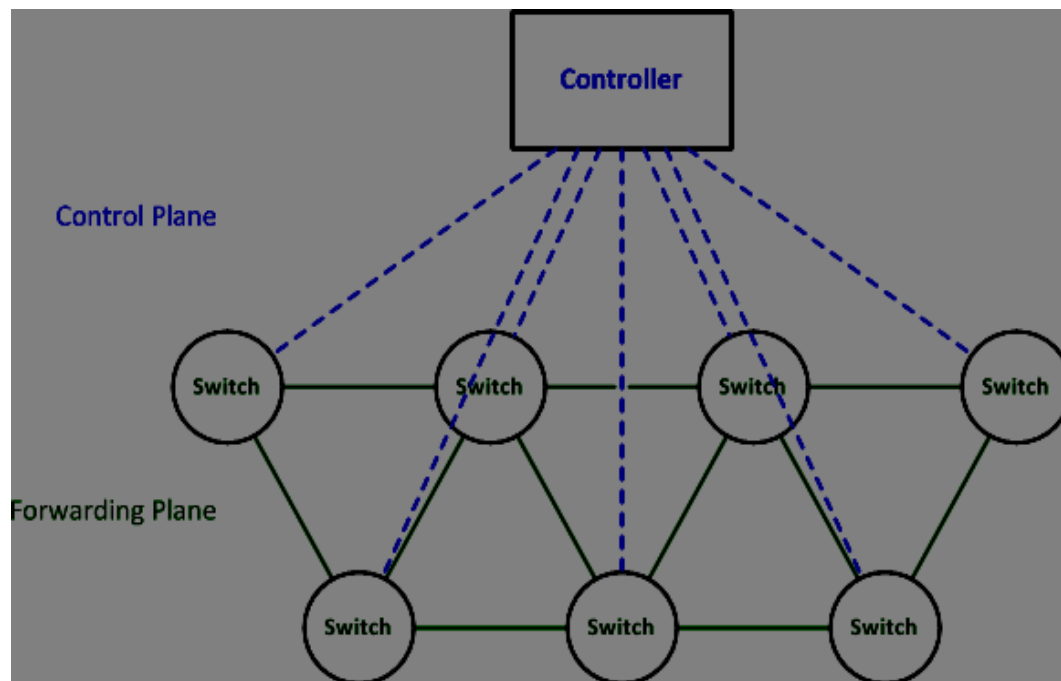
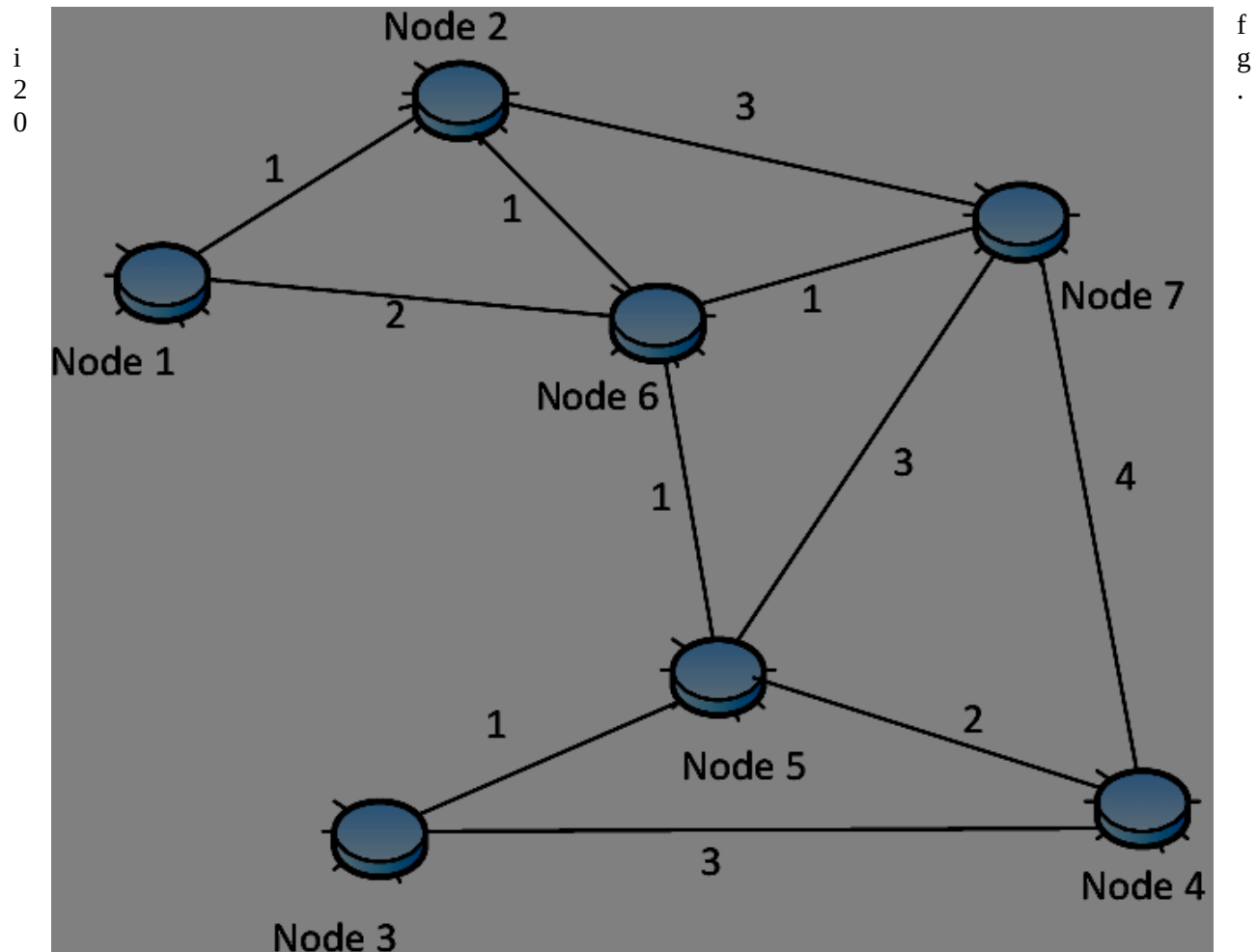


Fig1.0

2. WSN:

In this project Sensor network is assumed as a graph in which nodes and edges are present. Graph is implemented using Adjacency list, Adjacency list uses two types of Data Structure first is Hash Map other is Linked List. Graph has attributes named Nodes, Dynamic Array. Dynamic Array contains every node so that we can retrieve data from each node.

The design complexity of a WSN depends on the specific application requirements such as the number of nodes, the power consumption, and the life span of the sensors, information to be sensed and its timing.



Objective:

1. To generate wireless sensor networks.
2. To establish data communications through flow rules using SDN.

Objective Achieved:

1. We have achieved our primary objective of generating wireless sensor network.
2. We have established data communication through flow rules using SDN.
3. Additionally we have also achieved parallel transmission of packets.
4. Based on number of packets transmitted we have also calculated throughput and Packet delivery ratio (PDR).

Problem Statement:

Flow rule-based data transmission from source to destination with minimum cost in software-defined wireless network.

Literature Review:

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today applications. Software-Defined Networking (SDN) plays an important role in paving the way for effectively virtualizing and managing the network resources in an on demand manner. [1]The control plane is heart of the SDN architecture, so it is very important to give proper concern towards the design parameter of controller. Controller provides a programming interface to the network. Multiple controllers are being used to hold the backup of data of controller that controls the whole network. [2]

In recent years an efficient design of a Wireless Sensor Network has become a leading area of research, A Wireless sensor network can be defined as a network of devices that can communicate the information gathered from a monitored field through wireless links. [3]The data is forwarded through multiple nodes, and with a gateway, the data is connected to other networks. WSN is a wireless network that consists of base stations and numbers of nodes (wireless sensors) WSNs are highly useful in several fields of research, overall in computer science and telecommunications.[4] Such networks provide a great applicability from the point of view of monitoring to obtain important information in a system or an environment. The union between WSN and SDN architecture allows for multitasking in WSN. Architecture comprised of WSN and SDN allowing multitasking sharing the same network resources. Therefore, each sensor contains several programs running within them, which belongs to these applications based on user requirements. In order to support multitasking in an efficient way, it offers an optimization (in terms of load and energy) in all applications managed by a global controller with a scheduling process. [4]

Methodology:

- In first phase, we study information from various research paper, article, blogs related to Software Defined Network (SDN) and Wireless Sensor Network (WSN) and also we identifies the hardware and software requirements of the project, software requirement like operating system(Ubuntu), Programming language,
- Sensor network is assumed as a graph in which nodes and edges are present. Graph is implemented using Adjacency list, Adjacency list uses two types of Data Structure first is Hash Map other is Linked List. Graph has attributes named Vertex, Dynamic Array. Dynamic Array contains every node so that we can retrieve data from each node.
Each Node contains its name, Output Queue which contains forwarding packets and it is circular queue and Flow table decides the flow rules. Information is send from source node to destination node with minimum cost. Every node contains information of packets to be sent to which node.
The network (graph) connection is given randomly means the network is dynamic, connection between one node to another is not static. This is done by using rand () function. And the neighbours of any node is stored in list.

Implementation:

Sensor network is assumed as a graph in which nodes and edges are present. The following section contains details of implementation of our project.

Directory Structure and Files:

Project have following directory structure and file:

Directory	Files
src/cpp/	Driver.cpp,output
src/cpp/Network	graph.hpp,nodes.hpp,packets.hpp
src/cpp/Routing	Dijkstra.hpp, RoutingProtocol.hpp, TableStruct.hpp, Topology.hpp, Transmission.hpp
src/cpp/Controle_Plane	controller.hpp
src/cpp/Logs	log.hpp,_outputlog.log,_routingTable .txt
src/cpp/Message	message.hpp

[src/cpp/Driver.cpp:](#)

This is the driver file responsible for running the project and includes all the headers of the project. First object of Graph is created by taking input of vertex from the user and then creates the object of Topology class to create a network(dynamic) for **that create_Network()** and **view_Network()** is invoked to view the topology of the network. Then taking simulation time from the user simulation of the data communication is performed. In which messages are divided into packets by using **split_into_packet()** method of Message class. Then SDN applied, it generates a routing table for each message (according to message id) and sets flow rules to the nodes. Then Transmission started for each packet parallelly using Thread by invoking **startTransmission()** method of Transmission class. After completion of simulation the mathematical calculations take place.

Packet Delivery Ratio (pdr) = (total received packet)/(total sent packet)

*Success rate = (pdr * 100) %*

Throughput = (total received packet/time) packet per sec

[src/cpp/routing/topology.hpp:](#)

Having class Topology and it is responsible for creating any random network. Topology class has methods **create_Network(Graph & g)** that take Graph as an argument and using rand() function creates a network as a graph in the form of adjacency list. To calculate vertex u and vertex v formula is used $v \text{ or } u = (\text{rand()} \% \text{vertex})$ and to calculate weight is $\text{wt} = (u+v)\%20$ is used. **view_Network(Graph & g)** is used to print the network along with edge weight.

[src/cpp/network/graph.hpp:](#)

Class Graph has attributes vertex (Integer) which contains number of vertices, adjList used to represent graph which is implemented using hashmap and the structure is `map<int ,map<int,int>>` and Node* individual_Nodes which contains information about each node. It has a parameterized constructor (vertex as parameter). **addEdge()** method is used to add edges into the graph.

[src/cpp/network/nodes.hpp:](#)

This header file contains Node class, which defines the structure of nodes. It has attributes Node id (integer type and identifier of node) , packet (Object of Packet class and contains if any packet reaches to this node), Flow rule (copy of flow rule), output_buffer(only temp packet stored), output_queue (if node is destination then store packets) and range (node can communicate with in the range only).

[src/cpp/network/Packets.hpp:](#)

This header file contains two classes Header and Packets. Header contains information about the packet. It has attributes source (integer type and identifier of src node), destination (integer type and identifier of des node),length (length of payload), id (id of current node). The Packet class has attributes Header and Payload (string type and contains message) and methods are **setMessage()**, **setHeaderInformation()**, **getSource()**,**getDestination()** and **getMessage()**.

[src/cpp/routing/Dijkstra.hpp:](#)

This header file contains Dijkstra class which has a method shortest_path() it takes src,des and adjList as arguments. It runs the Dijkstra algorithm and returns the shortest path between src to dest in a stack.

[src/cpp/routing/RoutingProtocol.hpp:](#)

In this header file Routing class and Structure of table takes place. Structure of table has attributes next_hop which contains id of next node, action (if node is destination then action is receive, if medium_reliability is <0.2 then action is drop else action is forward and next field is medium_reliability which is decided by a rand() function.

The Routing class has an attribute Routing_table its structure is map<int ,vector<table_attributes>>; where int represents the first field of the routing table that is the current hope. **generate_Table()** method of Routing class is defined. It takes arguments as graph,src,des. In which object of Dijkstra class is created and **shortest_path()** method is called to calculate shortest path between src to destination and path is stored in a stack. Using this path routing table is filled. And **setTable()** method of Routing class forwards table to path nodes.

[src/cpp/controlPlane/Controller.hpp](#)

This header file is an SDN controller. Which have attribute name routing table map<string,map<int,vector<table_attributes>>> in which calculated routing table is stored for each source and destination using unique message id. To calculate routing table link-state routing protocol is used because we are doing intra-domain communication. Which is modified according to our project .Method generateTable(int src,int des,Graph &g,string msg_id) generates a routing table using method **generate_Table()** of RoutingProtocol class and stores it into a routing table. Method **generateFlowRule(Node *individual_Nodes)** generates flow rules for nodes and method **generateLog()** stores routing table into log file. Structure of routing table:

Structure of routing table:

current_hop	next_hop	tll	cost	medium_reliability
-------------	----------	-----	------	--------------------

<src/cpp/routing/Transmission.hpp>:

This header file contains Transmission class which is responsible for transmitting messages between nodes (from src to destination). It contains the method **startTransmission()** which takes Graph and Packet as arguments. It forwards packets according to the routing table. When a packet is reached at any node then the status of the packet is printed. If the value of the medium_ reliability field is <2.0 then the packet is dropped. If the node id is equal to destination then transmission ends and prints details of packet and successful message of acknowledgement

Version Control With Git:

To develop our project we are using the Version control system to keep track of previous versions and changes. We are developing our project under the development branch of GitHub

[repository of our project](#). All the user guide as well as developer guide is also maintained in README.md file of GitHub repository.

CLASS DIAGRAM:

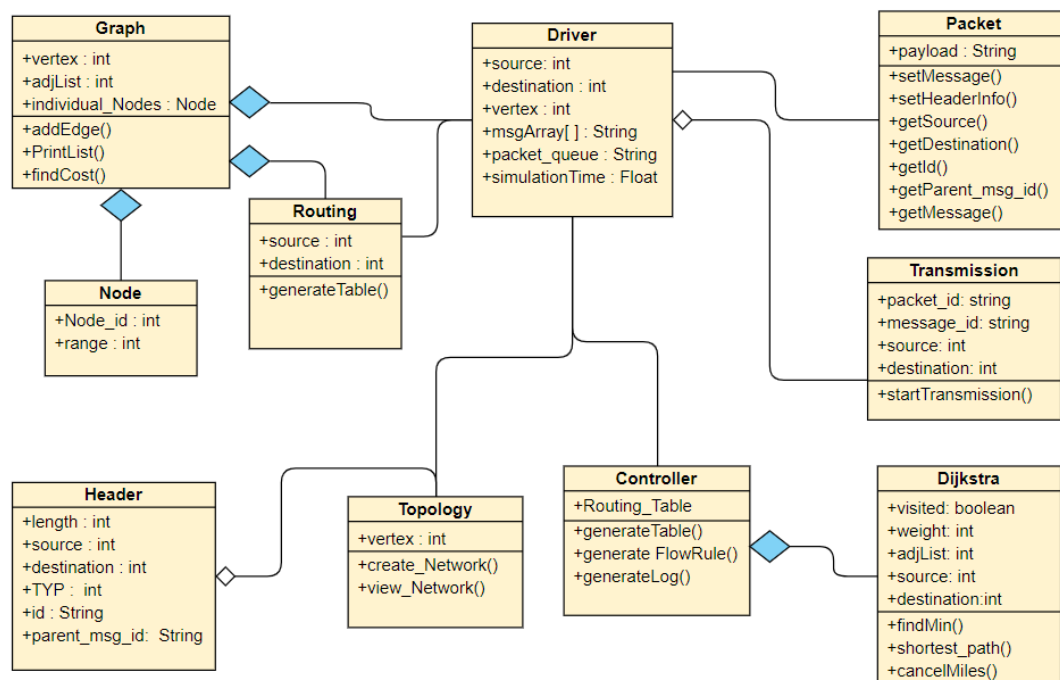


Fig3.0

FLOW CHART:

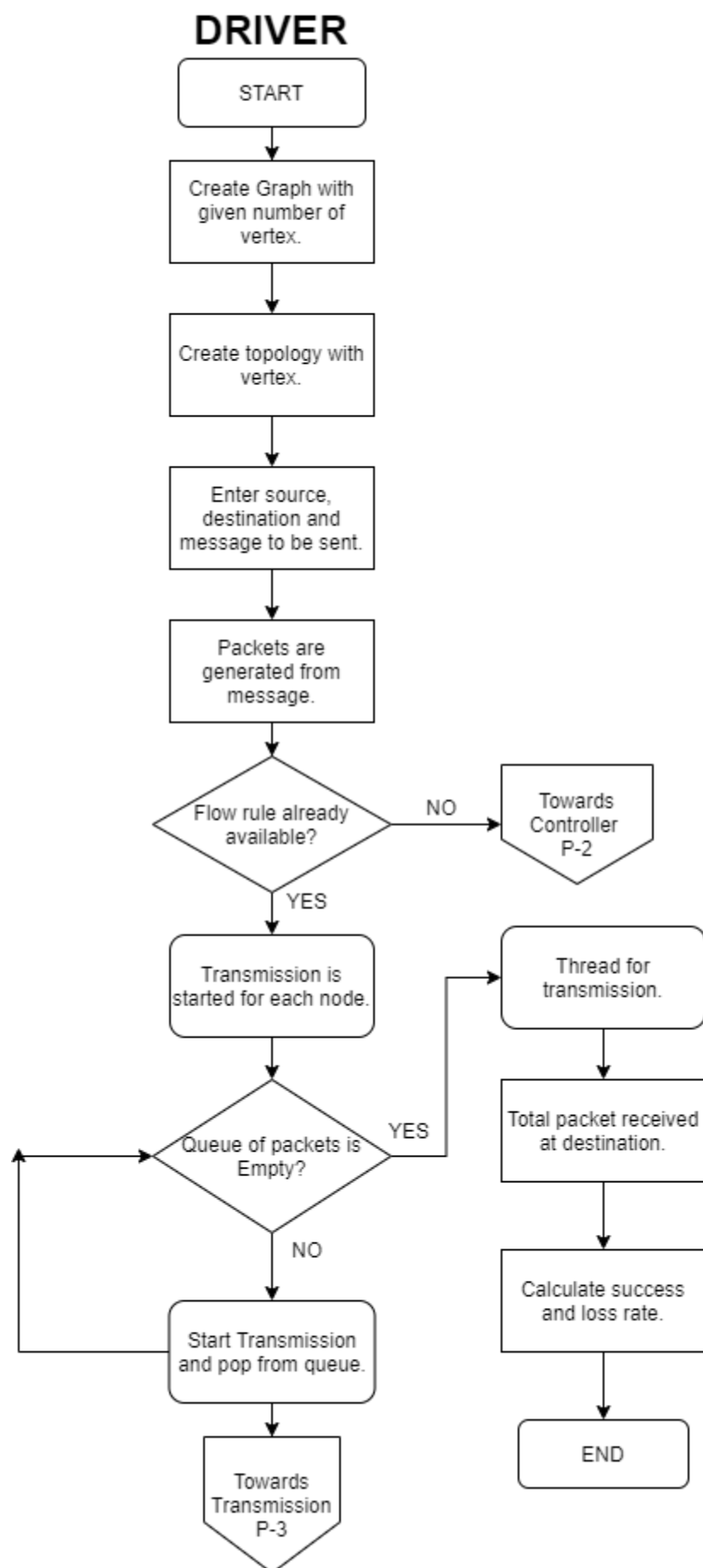


Fig4.0

CONTROLLER

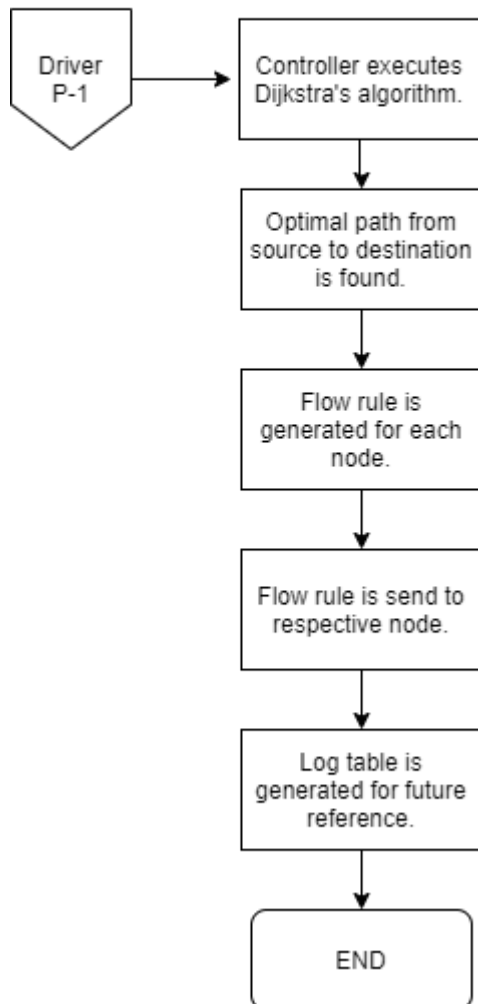


Fig5.0

TRANSMISSION

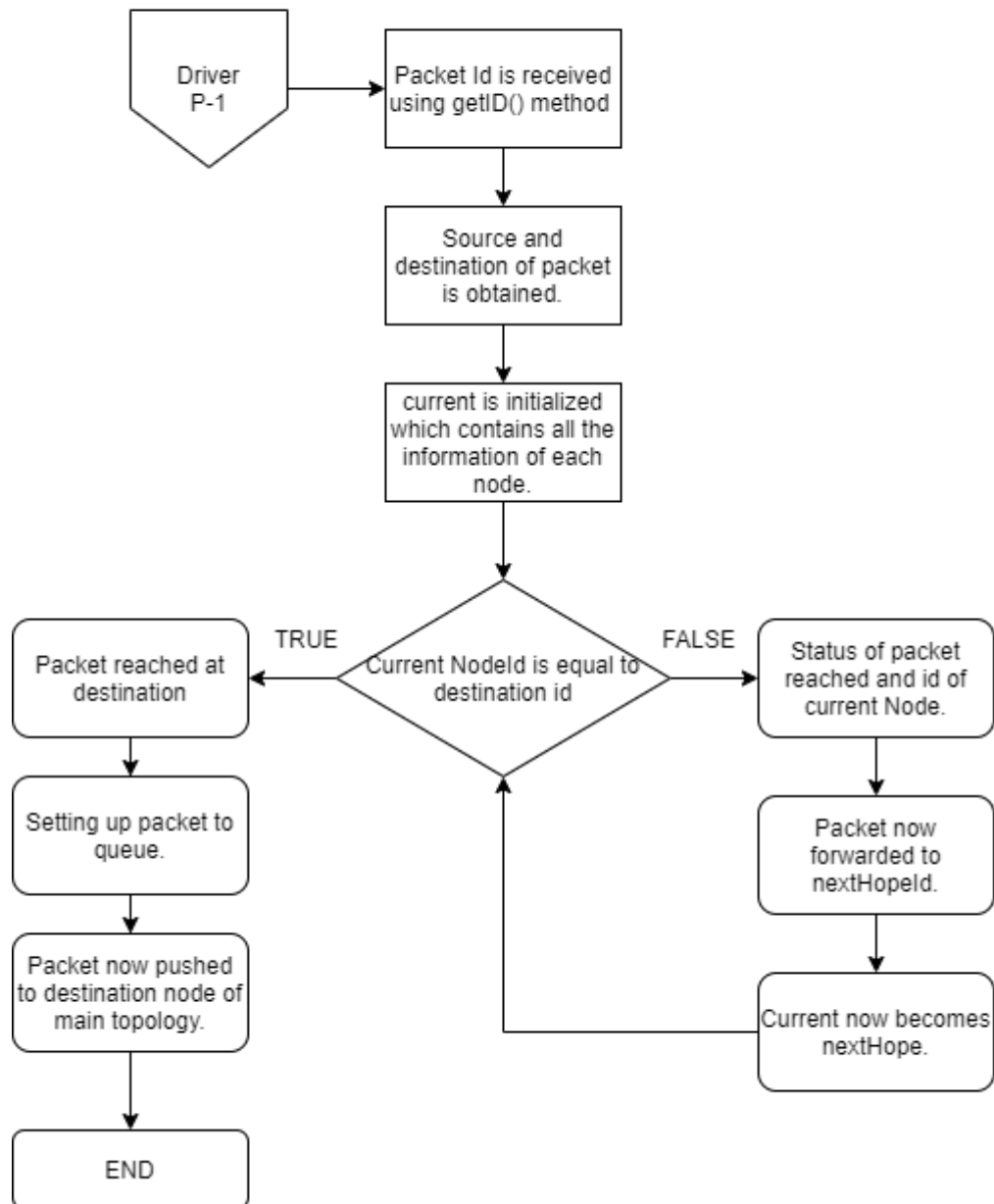


Fig6.0

PSUEDOCODE:

Dijkstra:

//Pseudo code for module Dijkstra

This module find the shortest distance from source to destination using Dijkstra Algorithm (Single source shortest path algo)

```
class Dijkstra{
```

```
function finMin(args visited, args weight, args v){
```

```
    initialize minWtNode to -1
```

```
    for loop from i = 0 to i less than v
```

```
    {
```

```
        if not visited i
```

```
            And minWtNode is equal to -1 oR weight of i less than minWtNode  
            than
```

```
                minWtNode = i;
```

```
    #end of for loop
```

```
    }
```

```
    return minWtNode;
```

```
end
```

```
}
```

```
function shortest_path(args adjList, args src args des, args v){
```

```
    for loop from i = 0 to i less than v
```

```
    {
```

```
        visited[i]=false;
```

```
        weigth[i]=inf;
```

```
    #end of for loop
```

```
    }
```

```
    initialize src weight to 0
```

```
    for loop from i = 0 to i less than v
```

```
    {
```

```
        call findMin function and initialize it to minWtNode
```

```
        for each loop neigh to adjList
```

```
        {
```

```
            if not visited neigh.first
```

```
            than
```

```
                if weight of neigh.first is grather than neigh.second+weight[minWtNode]
```

```
                than
```

```
                    weigth[neigh.first]=neigh.second+weigth[minWtNode];
```

```
                    parent[neigh.first]=minWtNode;
```

```
                #end of if statements
```

```
            #end of for each loop
```

```
        }
```

```
    #end of for loop
```

```
    }
```

```
    Print the Total Cost Required For this Transmission
```

```
    return;
```

```
end
```

```
}_____
```

Graph:

//Pseudo code for module Graph

This module print the bidirectional graph in adjacency list representation along with its weight.

```
function addEdge(args src, args des ,args wt){  
    //add edge in each node in bidirectional graph  
  
    adjList[src][des]=wt;  
    adjList[des][src]=wt;  
end  
}  
function printList(){  
  
    // this method print the garph in adjacency list manner  
  
    print the graph with nodes and edges  
    and also generate the log  
    end of line  
end  
}  
function findCost(args u, args v){  
  
    return adjList[u][v];  
end  
}  
#end
```

Packet:

// Pseudo code for Module Packet

This module create the pocket and send the meassage form src to des

```
class Pocket{  
    Initialize header  
    Initialize paylode  
  
    function setMessage(args msg){  
  
        return paylode = msg;  
  
    end  
}  
  
function setHeaderInfo(args id, args src, args des){  
  
    This method will set the information of header
```

```

    if header.src is less than 0
        oR header.src greather than 9
    then
        print invalid source: please check your network

        generate log
    exit()
    #end of if statements

    if header.des is less than 0
        oR header.des greater than 9
    then
        print the same message and generate the log
    exit()
    #end of if statements
end
}
function getSource(){
    return header.src;
}
function getDestination(){
    return header.des;
}
function getId(){
    return header.id;
}
function getMessage(){
    return payload;
}

end of class
};

```

Controller:

//Psudo code for module Controller

This module generate flow rule and routing table using routing protocol

```

function genrateTable(args src,args des,args msg_id){
    //generate routing table

    inalisize r;
    r.genrateTable(src,des,g);
    Routing_Table[msg_id]=r.Routing_Table;
end
}
function genrateFlowRule(args individual_Nodes){

    //this function generate flow rule

    for loop from x = 0 to size-1
    {
        for loop from y = 0 to x.second
        {

```

```

        auto v=(y.second);
        individual_Nodes[y.first].flow_rule[msg_id]=v;
    #end of neasted for loop
#end of outer for loop
}
end
}
function generatelog(){
    // this method print log
    print the log table
end
}

```

Transmission:

// psudo cose for Transmission module

This module responsible for transmitting messages between nodes (from src to destination).

```

function startTransmission(args Graph,args Packet){
    initialize packet_id = packet.getId();
    initialize msg_id = packet_id.substr(0,4);

    print Tranmission started for (packet_id)
    aNd generate log
    initialize src=packet.getSource();
    initialize des=packet.getDesti();
    // while loop start with the condition (current.Node_id!=des)
    while loop with the condition current.Node_id!=des
    {
        generate staus of log
        genaete log status of packat reached at

        if reliability > (0.8)
        {
            than print Packat could not be sent to next hope...
            print Medium reliability is greater then (0.8)...
            aNd
            generate log report Packat could not be sent to next hope...
            generate log report of Medium reliability is greater then (0.8)...

            print Packet lost between node
        }
        #end of if statements
    }
    else
    {
        print Packat forwarded to next node...

        Node nextHope=g.individual_Nodes[nextHopeId];
        current=nextHope;
    }
    print Tranmission completed for packet id
    #end of while loop
end
}

```

Algorithm:

- 1). START
- 2). Driver class runs the project. Graph is created by taking number of vertices as input.
- 3) Then, the object of Topology class created and to create a network(dynamic) `create_Network()` is invoked and `view_Network()` is invoked to view the topology of the network.
- 4) Simulation of data communication started. Step 5 to 7 repeated until `simulation_time < 0`.
- 5) Source and destination assigned using `rand()` function. Message is splitted into packets by using `split_into_packet()` method.
- 6) SDN applied through the controller which generates a Routing table using `generateTable(src, des, g, m.message_id)` method and set flow rule to path node using `generateFlowRule(g.individual_Nodes)` method.
- 7). Parallel transmission started for all packets stored in queue using `std::thread th(&Transmission::startTransmission, std::ref(g), std::ref(Packet))`.

Where Transmission is a class.

- 8) Finally, Mathematical calculation takes place to analyse the transmission.

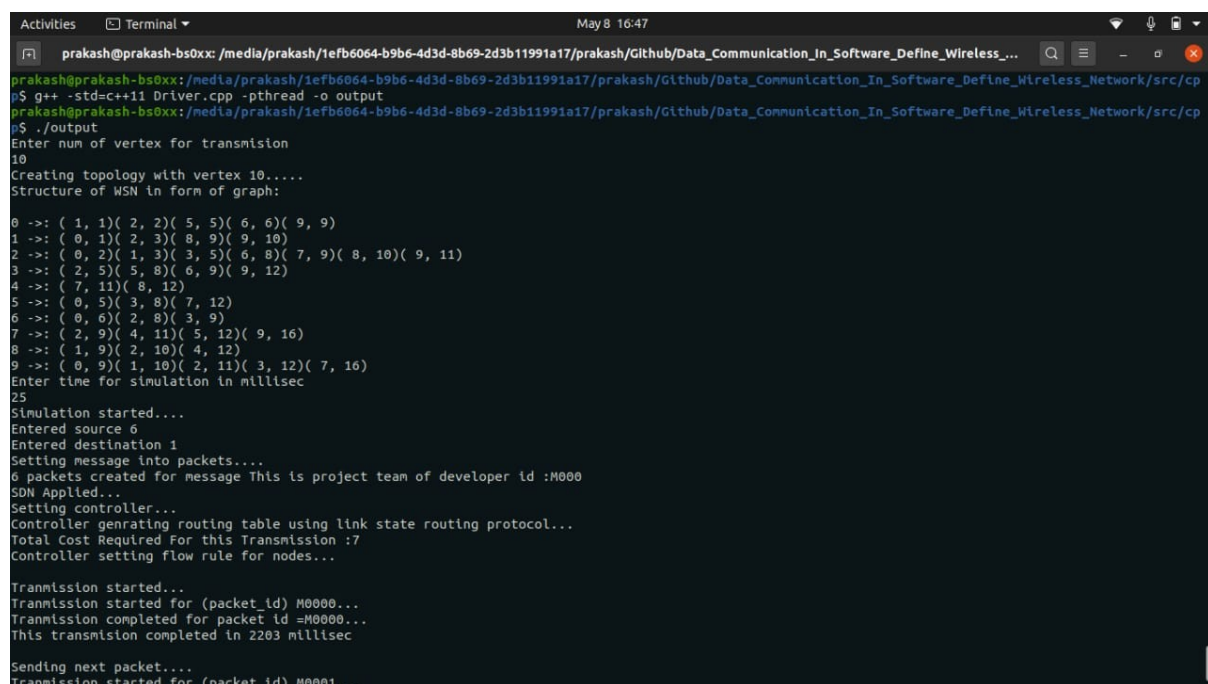
$$\text{Packet Delivery Ratio (pdr)} = (\text{total received packet}) / (\text{total sent packet})$$

$$\text{Success rate} = (\text{pdr} * 100) \%$$

$$\text{Throughput} = (\text{total received packet} / \text{time}) \text{ packet per sec}$$

- 9) END

OUTPUT:



```
prakash@prakash-bs0xx: /media/prakash/1efb6064-b9b6-4d3d-8b69-2d3b11991a17/prakash/Github/Data_Communication_In_Software_Define_Wireless_Network/src/cp
p$ g++ -std=c++11 Driver.cpp -pthread -o output
prakash@prakash-bs0xx: /media/prakash/1efb6064-b9b6-4d3d-8b69-2d3b11991a17/prakash/Github/Data_Communication_In_Software_Define_Wireless_Network/src/cp
p$ ./output
Enter num of vertex for transmission
10
Creating topology with vertex 10.....
Structure of WSN in form of graph:
0 ->: ( 1, 1)( 2, 2)( 5, 5)( 6, 6)( 9, 9)
1 ->: ( 0, 1)( 2, 3)( 8, 9)( 9, 10)
2 ->: ( 0, 2)( 1, 3)( 3, 5)( 6, 8)( 7, 9)( 8, 10)( 9, 11)
3 ->: ( 2, 5)( 5, 8)( 6, 9)( 9, 12)
4 ->: ( 7, 11)( 8, 12)
5 ->: ( 0, 5)( 3, 8)( 7, 12)
6 ->: ( 0, 6)( 2, 8)( 3, 9)
7 ->: ( 2, 9)( 4, 11)( 5, 12)( 9, 16)
8 ->: ( 1, 9)( 2, 10)( 4, 12)
9 ->: ( 0, 9)( 1, 10)( 2, 11)( 3, 12)( 7, 16)
Enter time for simulation in millisecc
25
Simulation started....
Entered source 0
Entered destination 1
Setting message into packets....
0 packets created for message This is project team of developer id :M000
SDN Applied...
Setting controller...
Controller generating routing table using link state routing protocol...
Total Cost Required For this Transmission :7
Controller setting flow rule for nodes...

Transmission started...
Transmission started for (packet id) M0000...
Transmission completed for packet id =M0000...
This transmission completed in 2203 millisecc

Sending next packet....
Transmission started for (packet id) M0001...
```

Fig7.0

```

Activities Terminal May 8 16:48
prakash@prakash-bs0xx: /media/prakash/1efb6064-b9b6-4d3d-8b69-2d3b11991a17/prakash/Github/Data_Communication_In_Software_Define_Wireless_...

Sending next packet....
Transmission started for (packet_id) M0001...
Transmission completed for packet id =M0001...
This transmission completed in 3255 millisc

Sending next packet....
Transmission started for (packet_id) M0002...
Transmission completed for packet id =M0002...
This transmission completed in 4157 millisc

Sending next packet....
Transmission started for (packet_id) M0003...
Packet could not be sent to next hope
Medium reliability is greater then 0.8 ie 0.9
Packet lost between node 0 1
This transmission completed in 4734 millisc

Sending next packet....
Transmission started for (packet_id) M0004...
Transmission completed for packet id =M0004...
This transmission completed in 4920 millisc

Sending next packet....
Transmission started for (packet_id) M0005...
Transmission completed for packet id =M0005...
This transmission completed in 5188 millisc

Sending next packet....
time left 0.543 millisc

Next message processed
Entered source 2
Entered destination 5
Setting message into packets....
3 packets created for message Hello Everyone !! id :M001
SDN Applied...
Setting controller...
Controller generating routing table using link state routing protocol...
Total Cost Required For this Transmission :7

```

Fig8.0

```

Activities Terminal May 8 16:48
prakash@prakash-bs0xx: /media/prakash/1efb6064-b9b6-4d3d-8b69-2d3b11991a17/prakash/Github/Data_Communication_In_Software_Define_Wireless_...

Sending next packet....
Transmission started for (packet_id) M0011...
Transmission completed for packet id =M0011...
This transmission completed in 708 millisc
time left -0.697 millisc

Next message processed
Total transmission 7

For detailed transmission visit to log file stored at /temp/_output.log (Linux OS)
-----Mathematical analysis-----

Total successful transmitted packets 7
Packet delivery ratio (pdr) = 7/9 = 0.777778
success rate 77.7778%
loss rate 22.2222%
Throughput = 28 packets/sec

Log file for this transmission is stored at /temp/_output.log
If want to see log file here press 1 else 00
If want to see routing table here press 1 else 0 1

-----
id Next_hop reliability TTL Cost
Routing Table for msg_id M000
0 1 0.8 50 1
1 1 0.8 50 0
6 0 0.8 50 6
id Next_hop reliability TTL Cost
Routing Table for msg_id M000
0 1 0.8 50 1
1 1 0.8 50 0
6 0 0.8 50 6
Routing table for msg_id M001
0 5 0.8 50 5
2 0 0.8 50 2
5 5 0.8 50 0

```

Fig9.0

Evaluation:

Pocket Delivery Ratio (PDR) = (total received pocket / total sent)

Throughput = Packet transmitted / time

E.g. For the simulation time of 100 msec the following result is obtained

Nodes	Throughput	PDR
10	12	0.66
15	13	0.72
20	15	0.38
25	13	0.48
30	3	0.23
35	17	0.53
40	13	0.50
45	16	0.41
50	3	0.33

Graphical Representations:

1.) PDR vs Nodes

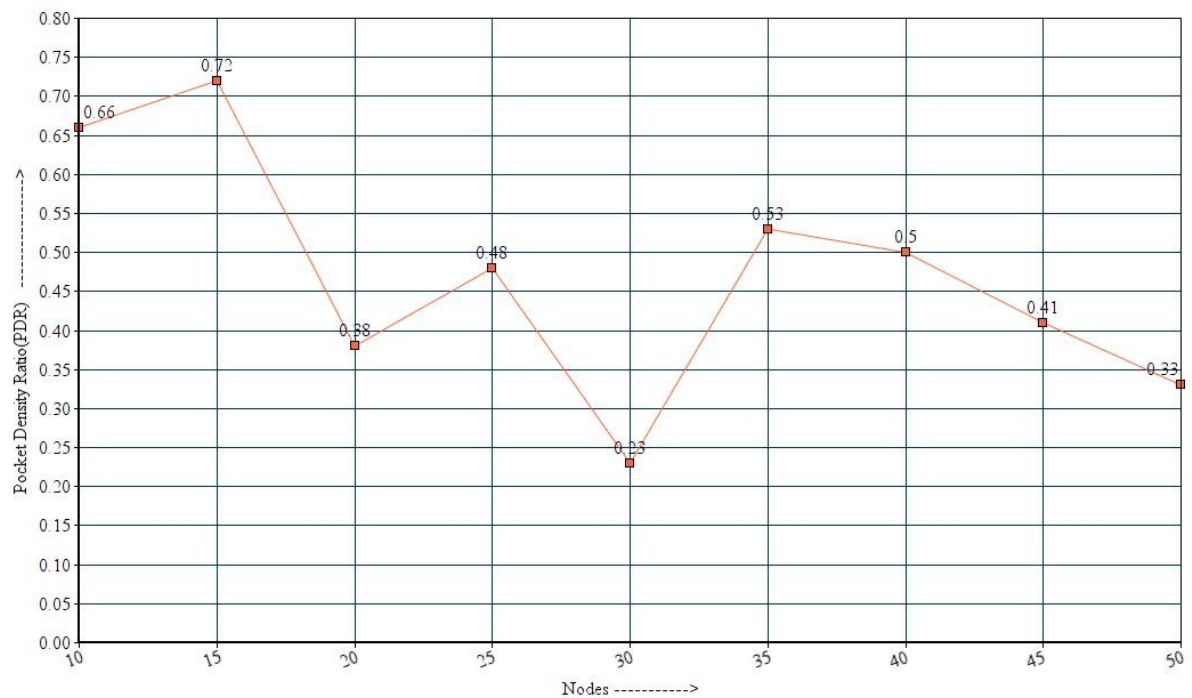


Fig10.0

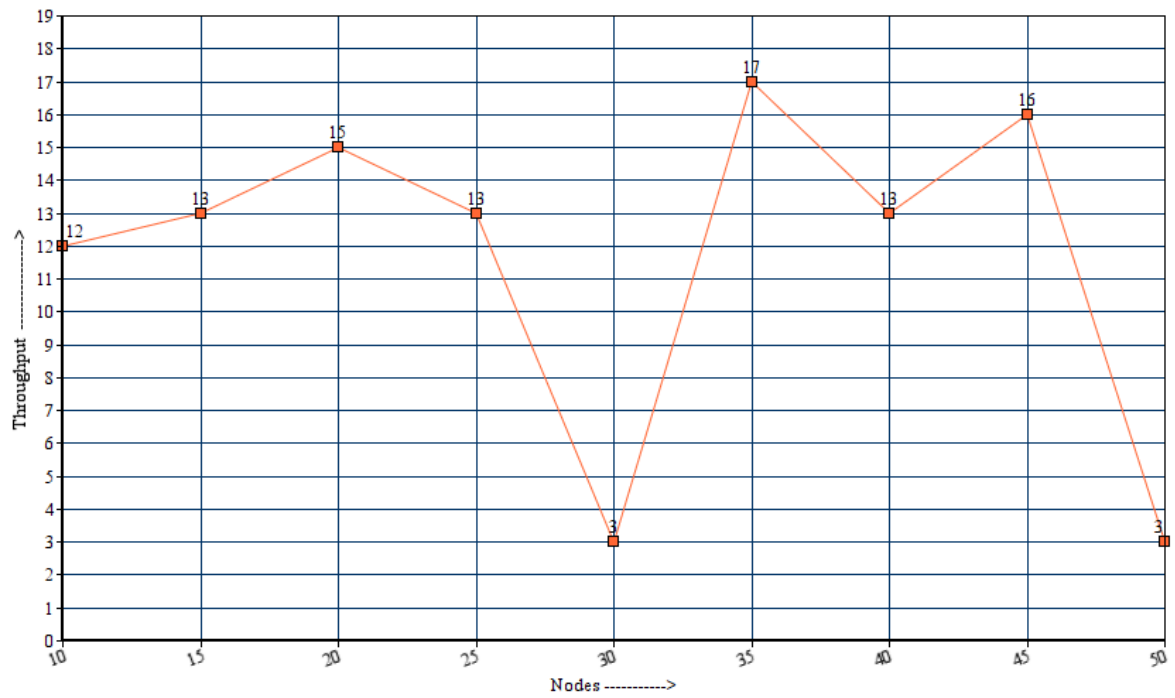


Fig 11.0

System Requirement:

Operating System	:	Ubuntu 18.04/Windows
Programming Language	:	C++
Running Environment	:	Terminal
Processor	:	Pentium IV
Disk Drive	:	Floppy or Hard Disk Drive
RAM	:	512 MB (min)

Schedule (PERT Chart):

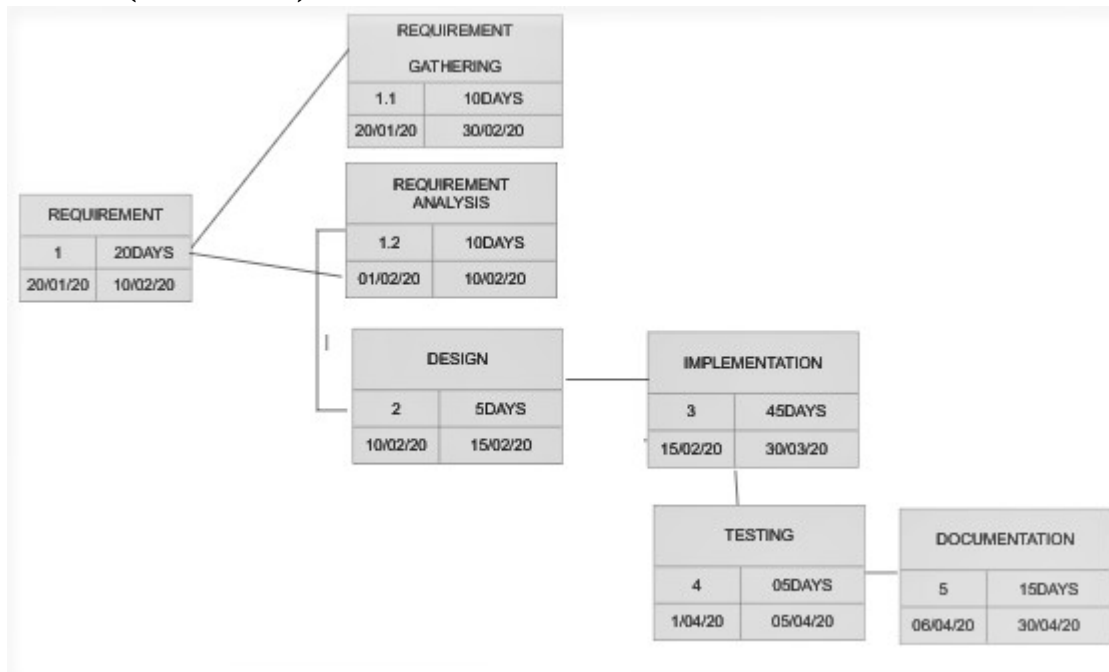


Fig12.0

References:

- [1]. Wei, Qing, & David, P. Dynamic Flow Rules in Software Defined Networks, Retrieved from Huawei Technologies - European Research Center, Riesstrasse 25, 80992 Munich, Germany.
- [2]. Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, Palo Alto, CA, USA, White paper, Apr. 2012.
- [3]. G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch," ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 44–51, 2014.
- [4]. Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A Hybrid Hierarchical Control Plane for Flow-Based Large-Scale Software-Defined Networks," Network and Service Management, IEEE Transactions on, vol. 12, no. 2, pp. 117–131, June 2015.

Approved By

Signature
Mr Amit Singh
Mentor

Signature
Dr. Monit Kapoor
Head of Department

