16:332:544: Communication Networks II

Final Project Report

K out of N Multicast Protocol Implementation

using Click Framework

Nithin Raju Chandy (158007346)

Thara Philipson (167004251)

Nirali Vinit Shah (168000050)

INTRODUCTION

In computer networking, **multicast** (one-to-many or many-to-many distribution) is group communication where information is addressed to a group of destination computers simultaneously. Multicast Routing protocols are used to distribute data like audio/video streaming broadcasts to multiple recipients with the aim of saving network bandwidth. If you unicast the same data to multiple recipients, it will results in bandwidth wastage. In order to broadcast messages to multiple recipients using minimum bandwidth, the paths which are common between the recipients should be recognized and utilized. Today, using multicast, a source sends a single copy of data to a single multicast address, which is then distributed to an entire group of recipients which belong to the group. Different Multicast protocols are there which takes advantage of common paths and duplicate packets whenever there's no further common path. In this project, we've implemented a k-out-of-n protocol using "Click" modular router software package.

ABSTRACT

As mentioned earlier, in this project, a k-out-of-n multicast protocol is implemented. It makes use of some concepts from existing multicast protocols. In this k-out-of-n multicast protocol, sender will specify a K value and will provide N destinations. K can be any value from 0 to N-1. For simplicity, we're considering a N value of 3 and K can be either 1, 2, or 3. The decision about final destinations will be made by routers. This decision depends on the total cost to the destination and the common next hops. Packets will be duplicated and updated whenever a particular router is not able to find common next hops for the packet destinations.

IMPLEMENTATION

We've implemented the following click elements for this project.

1. Packet Classifier
2. Client
3. Topology
4. Basic Router
5. Forwarder
6. Switch

1. PACKET CLASSIFIER

Packet Classifier element has 1 input and 4 outputs. It simply accepts packets from router ports and classifies it to HELLO, UPDATE, ACK and DATA Packets. We used classifier element at both router and hosts.

Fig. 1. Packet Classifier Element

A classifier object can be instantiated as follows

```
classifier :: PacketClassifier();
```

At the host, packet classifier's HELLO/UPDATE outputs are discarded as the host doesn't require HELLO/UPDATE messages coming from a router. But, at the router all packets will be sent to appropriate router elements. No packets will be discarded at the router's classifier.

2. CLIENT

Client is the element that basically implements hosts. A client object can be created using the following format.

```
client1:: Client( PAYLOAD "STRAWBERRY", K_VALUE 3,
                  MY_ADDRESS 101,
                  DST1 105, DST2 106, DST3 107,
                  DELAY 10, DATA_SEND_ENABLE 1);
```

We've an argument **DATA_SEND_ENABLE** which basically enables/disables the capability of a client to send data packets. This is required because there'll be only one sender and all other hosts will simply send HELLO and ACK packets. You can specify your own address using **MY_ADDRESS** argument. **DST1, DST2, DST3** Specifies three user specified destinations and **K_VALUE** defines the value of k. **DELAY** is just to introduce a delay for data packet generation. We've an argument **DATA_SEND_ENABLE** which basically enables/disables the capability of a client to send data packets. This is required because there'll be only one sender and

all other hosts will simply send HELLO and ACK packets. You can specify your own address using **MY_ADDRESS** argument. **DST1, DST2, DST3** Specifies three user specified destinations and **K_VALUE** defines the value of k. **DELAY** is just to introduce a delay for data packet generation.

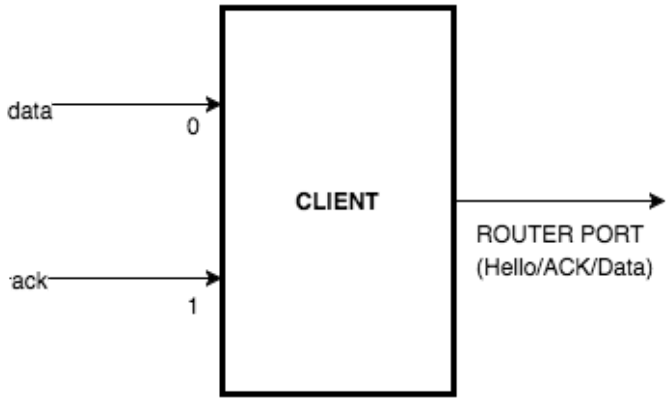

Fig. 2. Client Element

3. TOPOLOGY

Topology is one of the core elements of our router implementation. Topology element receives hello messages from classifiers and builds the ports table. It builds a table of (Destination, port) which has the details of immediate neighbours and the port to which they're connected. This also sends ACK messages for HELLO's received from neighbours. Topology element creates ACK's for received HELLO messages and paint it with proper output port before sending it to Switch element.

An object of topology element can be created as follows

```
topology :: Topology(MY_ADDRESS 500);
```

where **MY_ADDRESS** argument can be used to specify the router address.

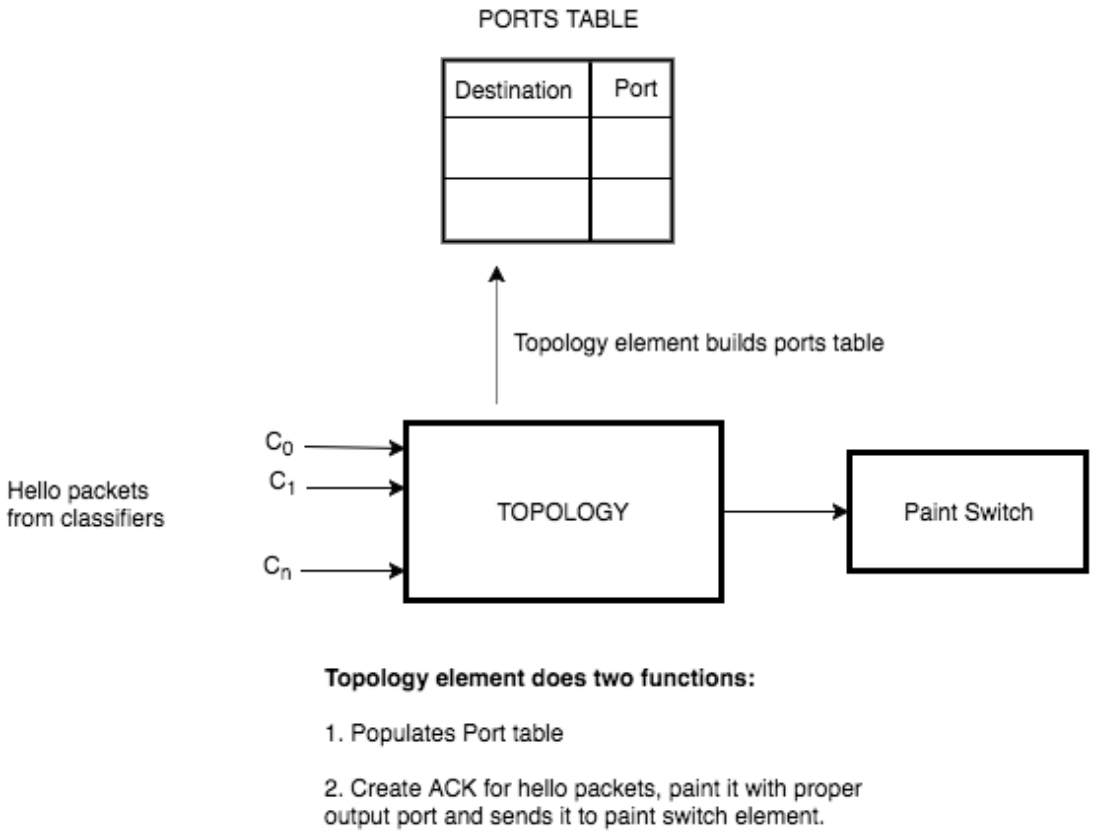

Fig. 3. Topology Element

4. BASIC ROUTER

Basic Router is another core element of our router. It receives routing UPDATE messages from classifiers and builds the routing table (destination, cost, next hops). This also sends ACK messages for UPDATEs received from neighbours. It creates ACK's for UPDATE messages, paint it with proper output port before sending it to Switch element.

A basic router element object can be created as follows

```
router :: BasicRouter(TP_ELEMENT topology, MY_ADDRESS 500);
```

It accepts two arguments. **`TP_ELEMENT`** allows you to pass the topology element object to the basic router element so that the basic router can access the ports table created by topology element.

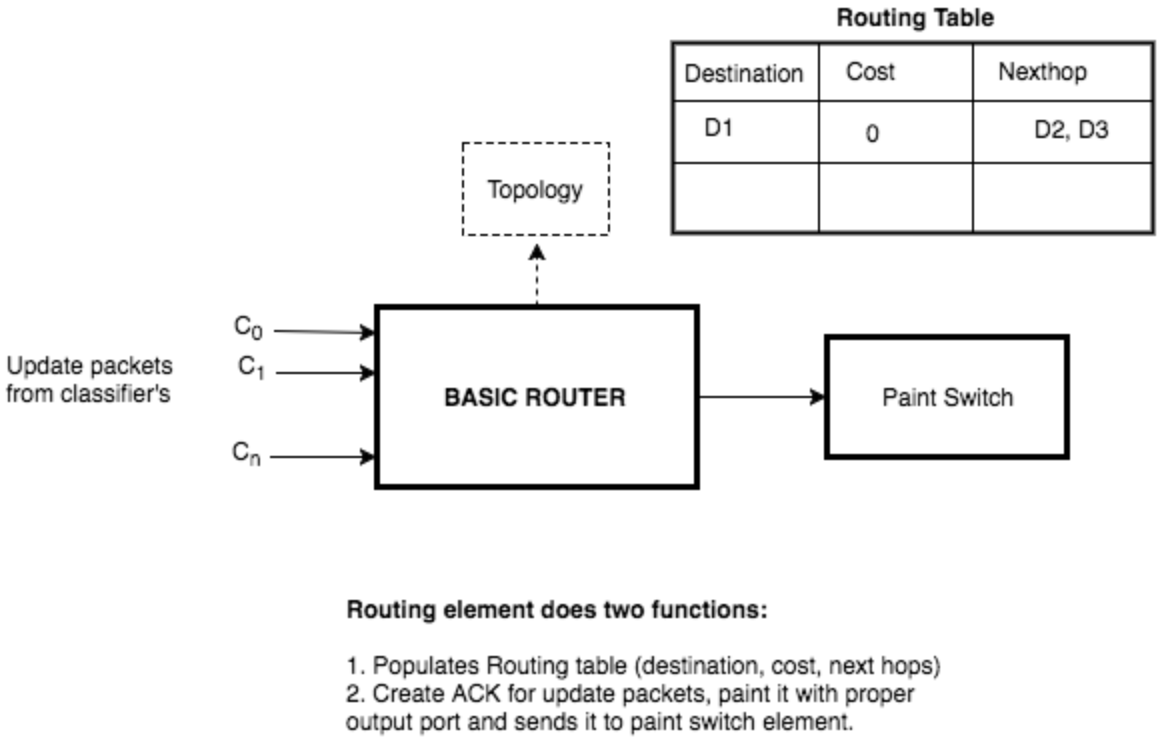

Fig. 4. Basic Router Element

5. FORWARDER

Forwarder is the third and final core element in the router. It receives DATA/ACK packets from the classifiers. It then checks the routing table to find the common next hop if any and forwards it to corresponding port by painting the packet.

Forwarder element object can be created as follows

```
forwarder :: Forwarder(TP_ELEMENT topology, ROUTER_ELEMENT router,SWITCH_ELEMENT switch, MY_ADDRESS 500);
```

TP_ELEMENT, ROUTER_ELEMENT, SWITCH_ELEMENT arguments passes those objects to forwarder element so that it can access other element's data structures. Access to **SWITCH_ELEMENT** is required as whenever you receive an ACK for a data packet, you need to remove it from corresponding output port queue which is implemented at switch element.

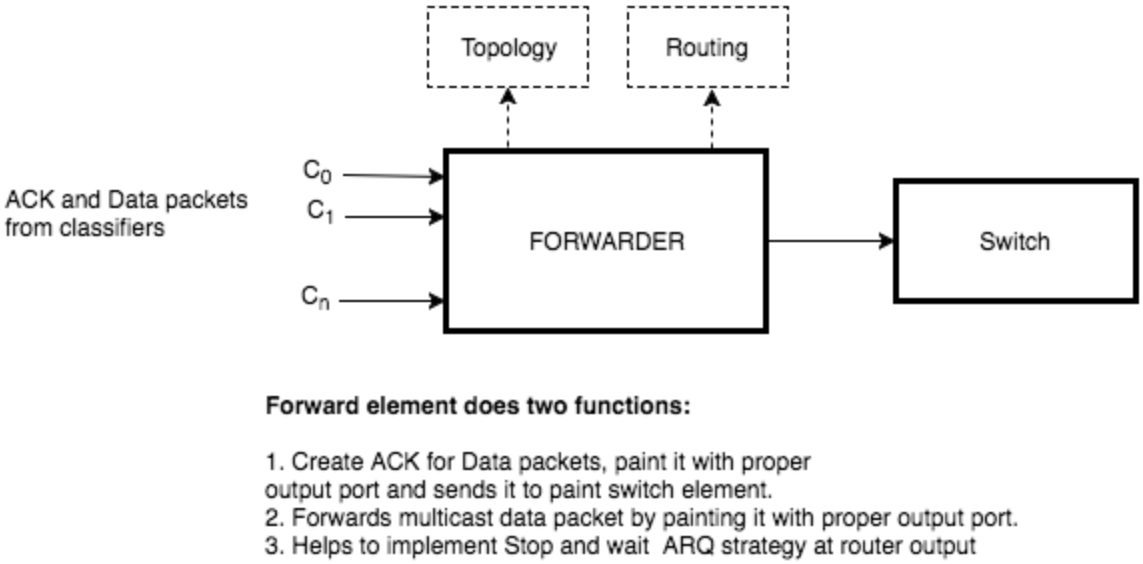

Fig. 5. Forwarding Element

6. SWITCH

Switch element simply checks the port values in the packet annotations and forwards packets to corresponding router ports. We also implemented queues at switch for each router output port to implement reliability at the router.

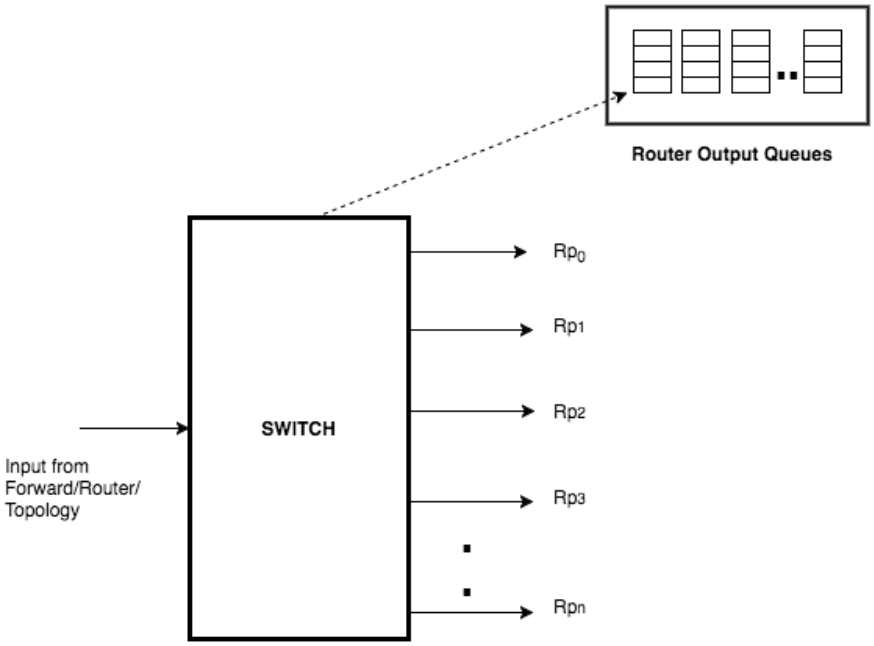

Fig. 6. Switch Element

Switch element object can be created as follows

```
Switch :: Pswitch();
```

ROUTER INTERFACE

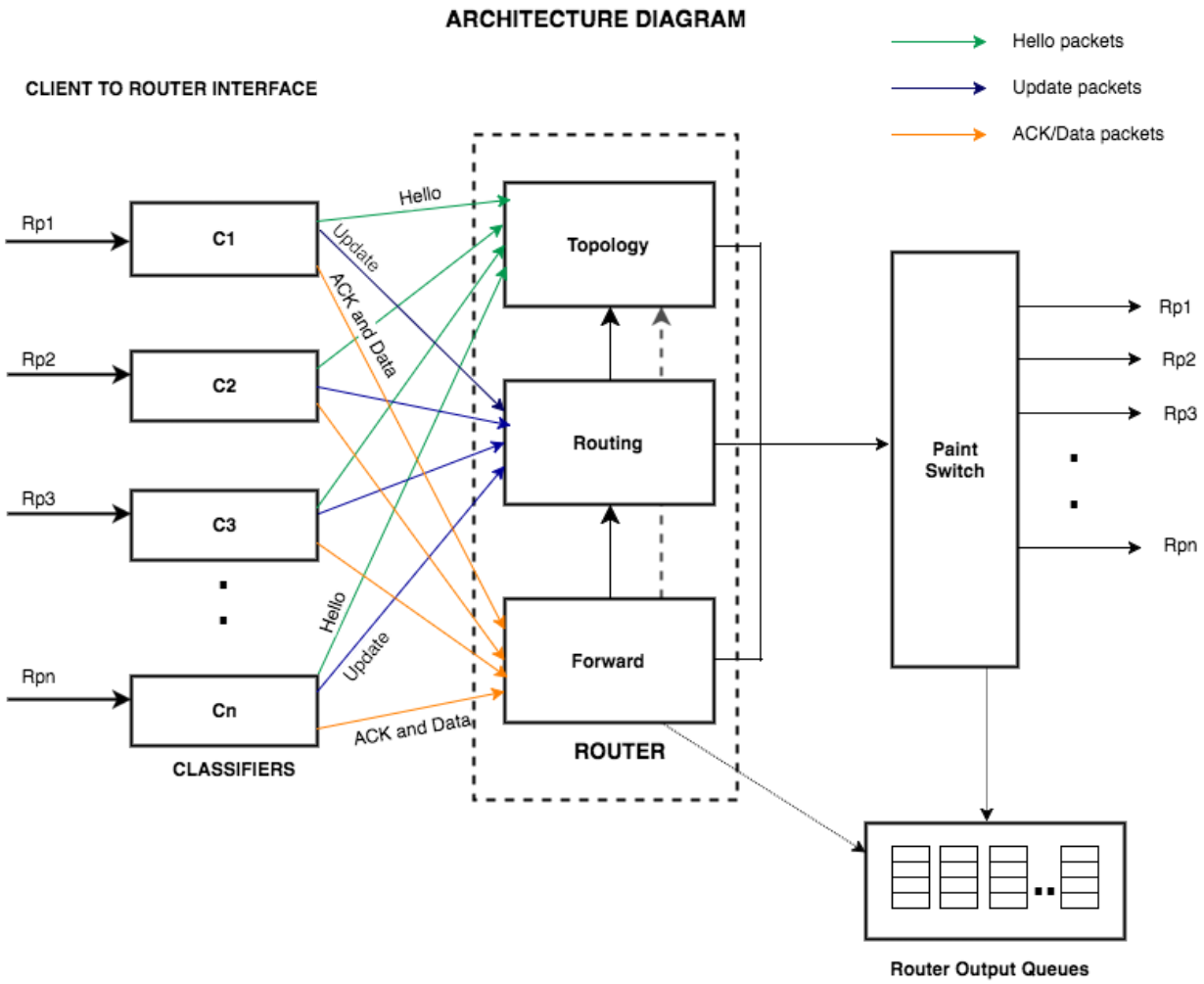

Fig. 7. Router Graph of Elements

All the packets arriving at router ports will first go through the classifier. Each router port will have its own classifier. Router has three core elements : Topology, Routing, Forwarder. Topology element gets all the HELLO packets and builds the ports table. Routing element gets all the UPDATE packets and builds the routing table which has destination, cost and next hops information. Forward Element receives

DATA/ACK packets and then it will check routing and topology element to properly forward the packet. HELLO Ack's, UPDATE Ack's, and DATA/DATA Ack's will be sent to switch after inserting the output port information in the packet annotation area. Switch element has output queues built in to for implementing ARQ scheme. Before any packet is sent, it will be queued at its corresponding output port queue. Switch checks the output port annotation and properly forwards the packet.

CLIENT INTERFACE

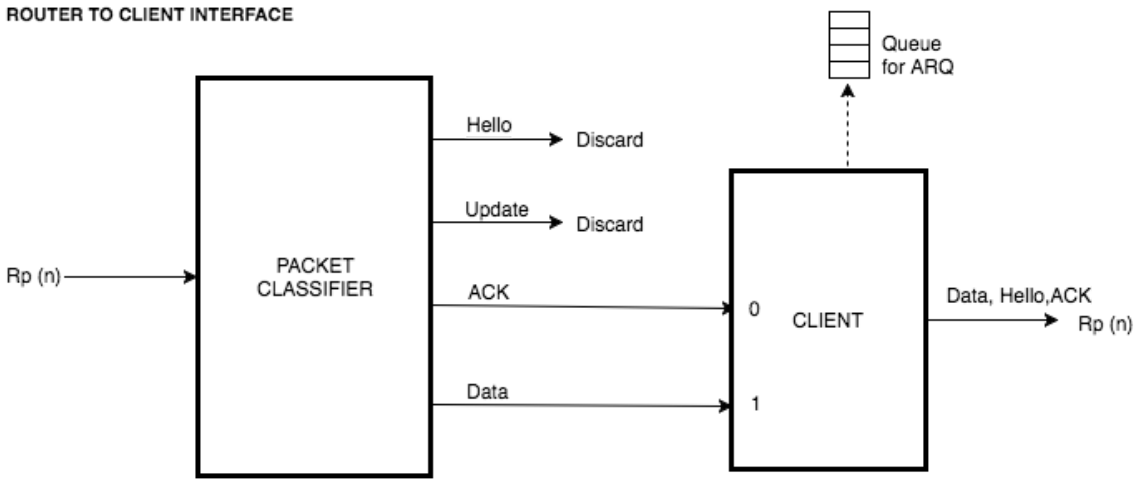

Fig. 8. Router - Client Interface

Packets from router ports will first go through packet classifiers. Hosts doesn't require HELLO and UPDATE packets from routers and hence the classifiers will discard those. ACK will be sent to client input port 1 and data will be sent to input port 0. We've also implemented queue at client for Stop and wait ARQ scheme. Only DATA, HELLO and ACK will be leaving the client.

DATA STRUCTURES AND ALGORITHMS USED

Ports Table	Map
Routing Table	Map Inside a Map
Multicasting	Set, Vector, Map, Set Intersect
ARQ	Queue

PACKETS USED

Hello packet - builds ports table

Type	Source Address	Sequence Number
1 byte	2 bytes	1 byte

Update packet syntax - builds routing table

Type	Source Address	Sequence Number	Length	Payload
1 byte	2 bytes	1 byte	2 bytes	

Acknowledgment packet syntax - for ARQ

Type	Source Address	Sequence Number	Destination Address
1 byte	2 bytes	1 byte	2 bytes

Multicast Data packet - Data to be sent

Type	Source Address	Seq Num	K	Destination 1 address	Destination 2 address	Destination 3 address	Length	Payload
1 byte	2 bytes	1 byte	1 byte	2 bytes	2 bytes	2 bytes	2 bytes	

Each router will send its own routing table in the update packet as an array of structs.

COLLECTED PERFORMANCE RESULTS

We've tested our implementation using the following topology (Figure 9).

When K=1,

- S sends the packet with user supplied values K=1 and 3 destinations to R1, its designated router .
- R1 checks its routing table and finds that D1 is at minimum hops away from it, so it designates D1 as destination. It makes other destination fields zero.
- R1 has R3, R2 as its next hop to reach D1 from which it randomly chooses one, so next hop could be either R1 or R3. Let's assume it takes the path R3.
- When the packet reaches R3, it finds next hop to reach D1 as R5.
- R5 then forwards it to D1 and thus D1 receives the packet.

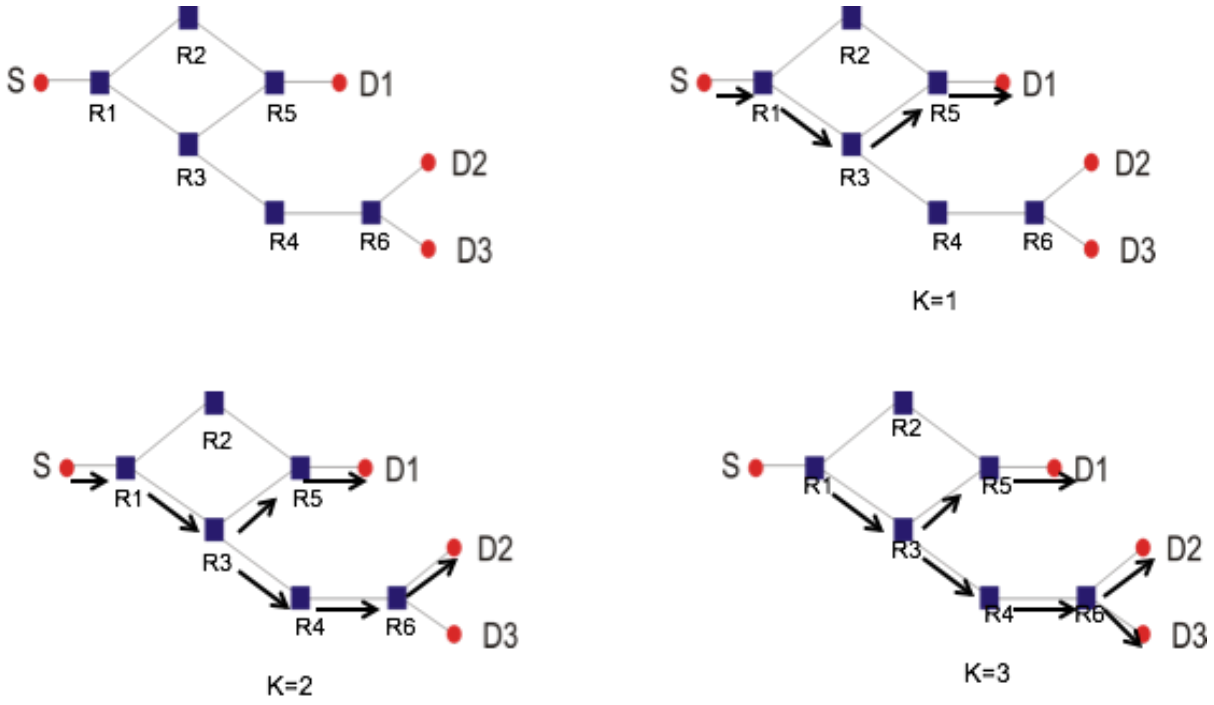

Figure 9. Test Topology

When K=2,

- S sends the packet to R1, its designated router.
- R1 checks its routing table and finds that D1 is at minimum hops and D2 and D3 are at equal distances, second nearest to the source, so it designates D1 and either D2 or D3 as the destination at random. eg. D2.
- R1 has R3 as its next hop to reach D1 and D2 evaluating a common path and so it sends the data packet to R3. Also, it fixes the destinations D1 and D2 in the packet and makes D3 field 0.
- When the packet reaches R3 it notices that K=2 and only two destination address exists and thus it copies and splits the packet as there's no common next hop from that point onwards. It creates two copies of packet each having K=1 and D2, D3 fields zero.
- R5 then forwards the first copy to D1 and R4 unicast the second copy to D2.

When K=3,

- S sends the packet to R1, its designated router.
- R1 checks its routing table and finds that D1, D2 and D3 are the three nearest destinations, so it designates D1,D2,D3 as the destinations.

- R1 has R3 as its next hop to reach D1,D2,D3 evaluating a common path and so it sends the data packet to R3. Also, it fixes the destinations D1,D2,D3 in the packet.
- When the packet reaches R3 it notices that K=3 and only three destination address exists. It looks for common path to reach the destinations and finds that R4 is the next hop to reach D2 and D3 and R5 is the next hop to reach D1.Thus it copies and splits the packet to R5 with K=1 and to R4 with K=2.
- R5 then forwards it to D1.
- R4 has R5 as the next hop to reach D2 and D3. Also its packet has K=2 and only two destination addresses exists. So, it splits the packet and unicast a copy to D2 and D3.
- Thus D1,D2 and D3 all receive the packet.

CONCLUSION

We were successfully able to implement k-out-of-N multicast protocol using click modular router software package. Several test topologies were used to verify the working.