Content

1 Introduction

1.1 Purpose

The purpose of this protocol is to implement a k-out-of-n packet datagram multicast network. In a network topology consisting of n sources, the algorithm must cast to k hosts. The Protocol uses a Link State routing algorithm using Open Shortest Path First (OSPF) and Dijkstra's algorithm to construct routing tables. Our data will be forwarded using our designed algorithm

1.2 Definitions

1.2.1 Router

A node that is connected to two or more networks, it forwards messages from one network to another. A packet is typically forwarded from one router to another router through the networks that constitute an internetwork until it reaches its destination node. As an intermediary network device, it will receive a datagram packet, read the destination field, and use its internal set of routing tables to decide which port to forward the packet to.

1.2.2 Node

A connection point inside a network that can receive, send, create, or store data. Each node requires some form of identification to receive access, like an IP address. A few examples of nodes include computers, printers, modems, bridges, and switches.

1.2.3 Multi-casting

Transmission of data from a single source to multiple recipients. Multicasting is similar to broadcasting, but only transmits information to specific users. It is used to efficiently transmit streaming media and other types of data to multiple users simultaneously. Transmission can range from a single destination all the way to every destination. The advantage that multicast has to broadcast is that only one packet is unciasted to a common router before it is distributed/broadcasted amongst the other multicast recipients.

1.2.4 Reliability

We define R(t), the network reliability at time t, as the probability that all the nodes are operational and can communicate with each other over the entire time interval [0, t]. As a communication protocol, it notifies the sender whether or not the delivery of data to intended recipients was successful.

1.2.5 Routing Table

A data structure that will be stored in a router that lists the routes to particular network destinations and in some cases distances (costs) associated with stored routes. To decide how to forward a packet, the router will perform a lookup on the routing table.

1.2.6 Routing Protocol

A routing protocol uses software and routing algorithms to determine optimal network data transfer and communication paths between network nodes. Routing protocols facilitate router communication and overall network topology understanding. It allows the network to dynamically adjust to changing conditions, otherwise, all routing decisions have to be predetermined and remain static.

1.3 Assumptions

- Unreliable network with packet loss prob p
- Each end node attached to only one router
- All links have the same characteristics(i.e hop cost 1, MTU 1500 bytes)
- A maximum number of nodes = 50.
- Link latency = 20ms.
- Link capacity = 10Mbps.
- N max is three

2 BOOTSTRAP AND DISCOVERY

2.1 Addressing Scheme

For our addressing scheme, we decided to follow a simple 1-byte address for each node. The addresses will start at 0000 0000 and will be incremented by one for each new address.

For example: The first node will be 0000 0000, the second address will be 0000 0001 etc

2.2 Procedure

For our procedure, the way we would go about it is:

- Discovering Neighbors: Periodic HELLO messages in links, where the neighbors would respond to these HELLO messages, identifying the neighbor where the addresses of the neighbors are also included
- Neighbor Link Cost: link state would have to estimate the cost of the links by having tests performed to measure delay or bandwidth. For simplicity, we will start with hop cost = 1 for all links.
- Creating and sending the link packets: Packets with neighbors would send updates on discovering and learning about links and routers in the network.
- Finding the shortest path: We would use the Dijkstra algorithm to find the shortest paths through the nodes.

2.3 Packet Format

2.3.1 Packet Types

- HELLO Packet - enables routers to share information with each other. This packet is sent out periodically from each router to confirm adjacent connections. In a Multicast transmission such as the one in our proposal, a HELLO packet can be simultaneously sent to multiple routers.
    - Type, Seq, TTL, Src
- Update Packet
    - Type, Seq, TTL, Src, LS
- Data Packet - Unit of data made into a single package to travel along the network
    - Type, Seq, Len, Src, K-Val, Dest (1-N), N-val, Data, Remaining K
- Data Ack Packet - a packet containing data of acknowledgment
    - Type, Seq, Src, Dest

2.3.2 Semantics

- Type
  - 4 packets in the protocol, size is 2 bits. For implementation, we can use one byte.
- Seq
  - Used in the multi-packet chain. Indicates the number in a sequence. Crucial to ARQ and packet ordering. Size is 2 bits. For implementation, we can use one byte.
- TTL - Time to Live
  - An arbitrary amount of time[2] a packet may spend on the network until it is taken out of circulation. Size is 1 byte.
- Src / Dest - Source and destination
  - Working in an environment with less than 50 nodes, only 8 bits are needed.
- Len - length of data
  - based on $\log_2 n$ where n is max data per packet + length of headers. Size is 16 bits.
- Data
  - content of packet. Up to 1500 bytes.
- K-val
  - Number of nodes the algorithm must reach, In this specification, we will use 8 bits due to size limitations of the network
- K-remaining
  - Number of nodes the packet must reach, we will use 8 bits
- N-Value
  - Total number of receivers in the network, we will use 8 bits
- LS
  - routing table of nodes sending a particular packet. The size of this can scale from 6 bits to n*24 bits. The structure of the routing table will be, for each other destination in the network, a nexthop to get to it, a total cost to get to it, and a second to last hop to get to the destination.

3 ROUTING AND MULTICASTING

3.1 Routing

The protocol uses a typical Link State OSPF functionality to construct routing tables. When the protocol is initialized, each router sends a Hello packet in order to discover each of its neighbors. Once every router has discovered and established communication with its neighbors, the exchange of Update packets will begin. Each router will send its neighbor's packets informing them of what they know about their own distance from each neighbor. After the first exchange, each node will only know its distance to each of its neighbors. After the second iteration, each node will know its distance to every node reachable with two hops. When an iteration occurs and no Node learns anything new, each node has all the information it needs to construct its routing table.

In order to construct a routing table, each node will perform a Dijkstra operation to find the shortest path to any node. The routing table will have an entry for every other node (both routers and sources) in the topology, and the entry will include: the distance to

that node, the next-hop required to get to that node. In other words, if to get to Node E from Node A we must follow the path A, B, C, D, E, the entry for Node E in A's routing table will consist of a distance, 4, a next hop, B.

The Nodes will periodically[1] exchange Hello packets to verify that their understanding of the network topology, given by their routing table, is still accurate. If at any point a hello packet exchange yields that a new node has appeared, or an old node has left the network (or stopped working) the process of exchanging Update packets until there are no updates will occur again. When a node enters the network it will send hello packets to discover its neighbors, also starting the Update process.

3.1.1 Routing Packet Format

We will be working with 3 types of routing packets, the HELLO packet, UPDATE packet, and Data Ack packet.

The HELLO packet consists of:

Type	Seq	TTL	Src

Table for Type semantics:

Packet Type	Bit Structure
HELLO	0000 0000
UPDATE	0000 0001
DATA	0000 0010
ACK	0000 0011

1 byte	0000	00000000	00000000

The UPDATE packet consists of:

Type	Seq	TTL	Src	LS

00	0000	00000000	00000000	...

Expansion of LS

Neighbor$_1$	Distance to Neighbor$_1$	Next Hop to Neighbor$_1$	...	Neighbor$_N$	Distance to Neighbor$_N$	Next Hop to Neighbor$_N$

00 00 00 00	0000000 0	000000 00	..	0000 0000	00000000	00000000

The Data Ack packet consists of:

Type	Seq

00	0000

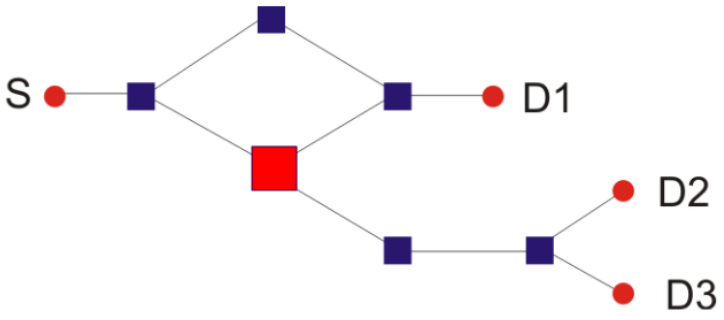
The type will require two bits, for the four possible packets in the protocol. Seq indicates the number in a sequence (important for ARQ and packet ordering).. Src/Dest will take another eight bits, given the assumption that we have less than 50 nodes. TTL, or Time-to-live (the amount of time until forwarding planes take the packet out of circulation) is eight bits, given the time was set to 255. Lastly, LS, the routing table of the node sending the packet, takes number of nodes * 24 bits. It is also important to note that the Seq value in the Data Ack packet must correspond to the Seq value of the data packet it is acknowledging.

3.2 Multicast

3.2.1 Algorithms for Multicast Routing

The Multicast Routing Algorithm for the protocol is based on using an algorithm to calculate the most ideal path to take. Here is a rundown of the steps the algorithm takes when arriving at a node.

First, the algorithm checks which destinations can be reached. The algorithm checks if all immediately neighboring nodes can reach the same set of destinations as itself. If all of the nodes evaluated can reach the same set of destinations, the algorithm will pass the packet to the next most common node. At the red square, there are two neighbor nodes reachable, but there is a split path. Our algorithm checks what the ideal longest path is in order to reduce the number of hops the packet takes for reaching a destination



If instead, the current node's neighbors have differing sets of reachable destinations, the most ideal path or paths must be identified and this is done using Dijkstras.

Pseudocode: (Dijkstra's Algorithm)

```
if (src/dst is not in graph):
        Return
if (src=dst):
        Build a list for the shortest path
        while (dst is reachable):
                add dst to path
                update the dst
        return the path array and destination value
else:
        initialize the cost
        visit the neighbor, if is not,visited, update the distance and mark as visited
        add src to visit list
        select non-visited node with shortest distance x
Dijkstra's algorithm works with src = x
```

3.2.2 Algorithms for Message Forwarding

When the data forwarding plane sends a packet down a path, as determined by the routing algorithm, it uses a fairly simple copy and forward method, but with one key difference.

As discussed in the routing algorithm section, when the protocol must send packets down multiple paths, these paths will have different expectations for how many packets are needed. In some cases, all the packets in a particular path will be needed, and in some fewer will be. Thus, when the algorithm reaches a diverging point, the packets sent down the different paths will vary slightly. Consider the situation from the second sample network in section 3.2.1. Using Dijkstra's we will be able to determine the shortest path to each of the three destinations. We will then compare these paths to see if they have any nodes in common. For paths that have nodes in common we will only send one data packet along that route. If the path diverges we will need to send one packet to each of the branches.

3.2.3 Data Packet Format

Type	Seq	Len	Src	Dest	K-Val	N-Val	Remaining K	Data
00	0000	000000	00000000	00000000	000000	000000	000000	...

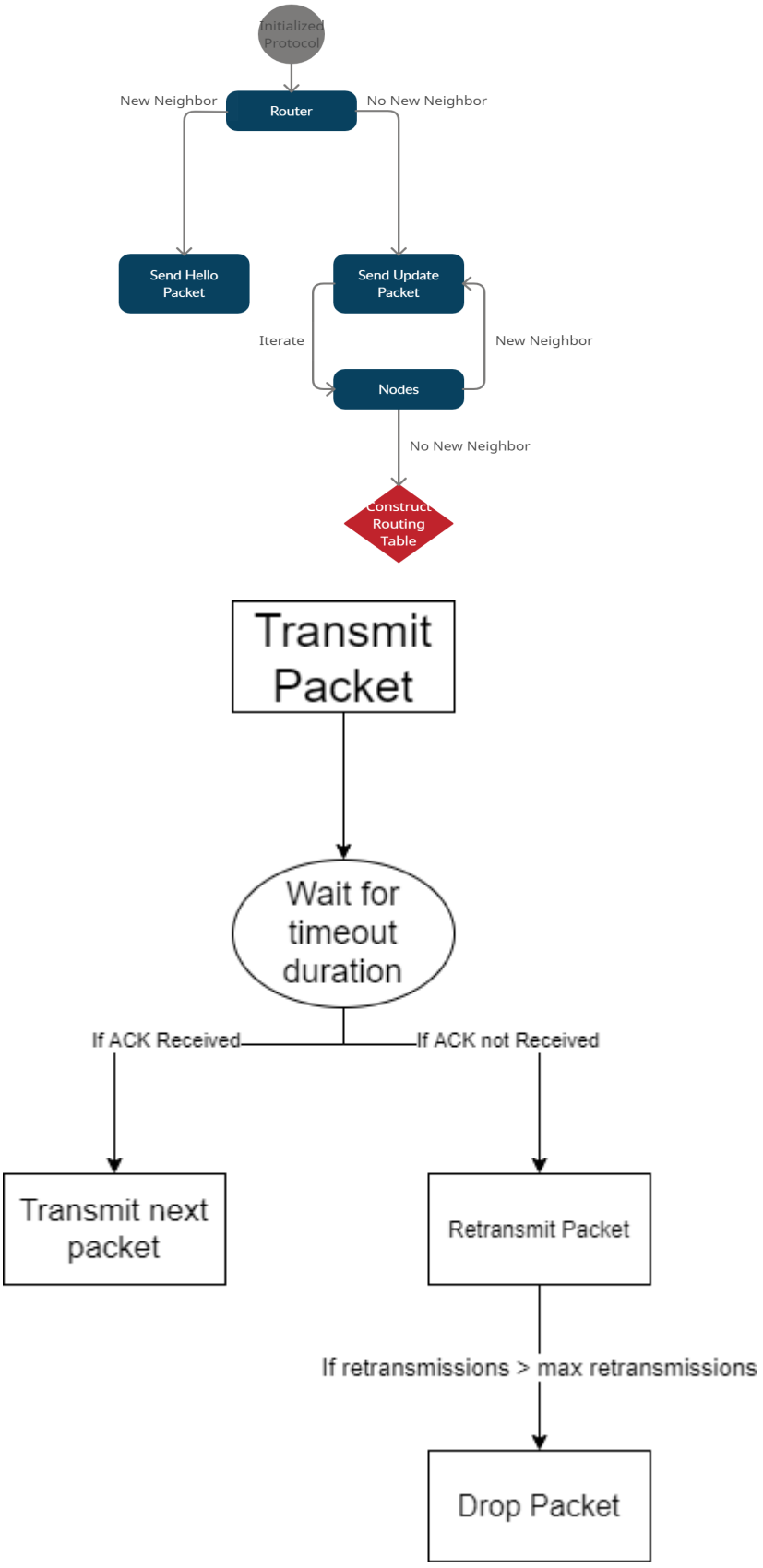
The type will require one byte, for the four possible packets in the protocol. Seq indicates the number in a sequence (important for ARQ and packet ordering), needing one byte. Len indicates the length of the headers. Src/Dest will take another eight bits,

given the assumption that we have less than 50 nodes. K-val, the number of nodes the algorithm reaches, will take eight bits as well. Lastly, remaining-K , the number of nodes a particular packet will reach, is also eight bits.
Note: if header size is not static, add an h-len field.

Pseudocode for Packet Handling

```
read packet type
if (packet is hello packet):
        find a destination which is an immediate neighbor and populate the graph to this host
elseif(packet is update packet):
        populate graph with the rest of routers
elseif(data packets):
        if (destinations 2 and 3 are invaild):
                search for the first destination (dst1)
                add the destination to Thread
        elseif (dst 2 is valid but dst 3 is invaild):
                identify Dijisktra distance to closest destination
                save the information of destination (dst)
                if (k=1 for one server)):
                        server searches for the dst
                        Thread start to route
                else (two servers):
                        if (two paths are the same):
                                search for dst 1 and route for both
                        else:
                                server 1 to dst 1 and server 2 to dst 2
        else (both destinations are valid):
                identify shortest Dijkstra distance for 3 paths
                save the information of destination (dst)
                if (k=1):
                        the server will be routed to dst
                elseif (k=2):
                        compare three paths for two servers
                        if (there exists same path length):
                                route the packet to dst1/dst2
                        else:
                                sort the distance of three paths and route to the shortest one
(server 1 and server 2 to dst)
                else (k=3):
                        compare three paths for three servers
                        if (there exists same path length):
                                route the packet to dst1/dst2/dst3 regarding the equality of path
distance
                        else:
                                route the packet from server 1/2/3 to dst 1/2/3
        else:
                read and send ACK packet
        RETURN
```

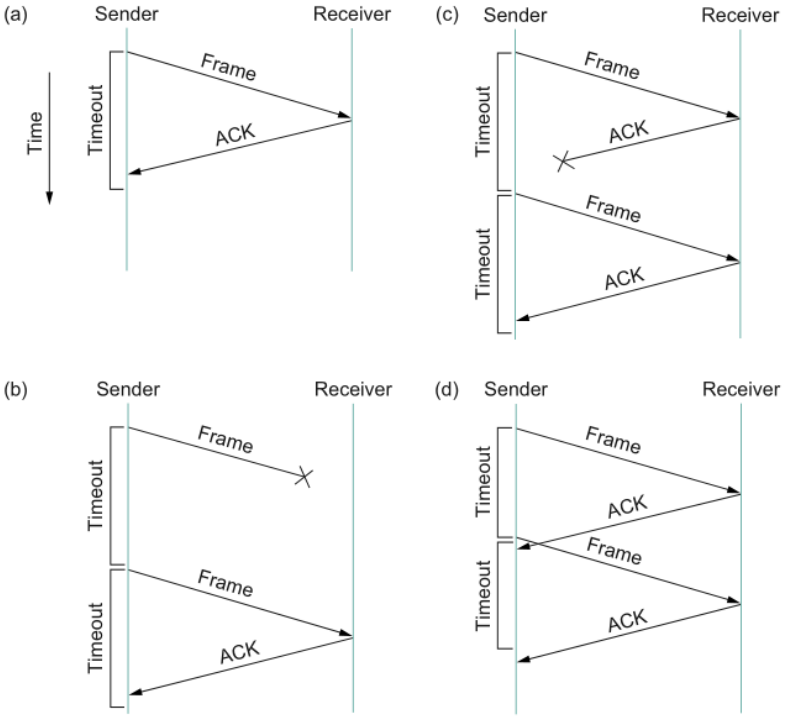
3.3 State Diagram



Stop-and-Wait ARQ State Diagram

4 RELIABILITY SCHEME

Our reliability scheme of choice will be Stop-and-wait ARQ. This is the simplest approach that will also be effective for this problem. Stop-and-wait will ensure that no packets are lost and that packets are received in the correct order. Packets are sent one at a time and this method will make use of our Ack packet. Furthermore, the reliability scheme will function on a hop-by-hop basis.



■ FIGURE 2.17 Timeline showing four different scenarios for the stop-and-wait algorithm. (a) The ACK is received before the timer expires; (b) the original frame is lost; (c) the ACK is lost; (d) the timeout fires too soon.

Computer Networks, a systems approach 5th edition, Larry Peterson, Bruce Davie

4.1 Procedure

1. Sender transmits a single packet.
2. Sender waits for an acknowledgement from the receiver.
   2.a. If the sender receives an ack packet with the correct seq number, transmit the next packet.
   2.b. If the sender does not receive the acknowledgement packet within time frame t, the sender retransmits the packet and goes to step 1.

4.2 Estimated Efficiency

The maximum sending rate of a Stop-and-wait ARQ system is

$$\frac{Bits/Frame}{Time/Frame}$$

Given our assumptions, this will come out to a maximum sending rate of 192kbps.

4.3 Pseudocode

```
t = time
p = packet
x = maxNumTransmissions
numberTransmissions = 0
transmit(packet, receiver)
wait(time)
if(ackReceived)
   transmit(nextPacket, receiver)
else if(numberTransmissions > x)
   drop packet
else
   transmit(p, receiver)
   numberTransmissions++
```

5 Implementation

In this section, we will show the exact implementation of our network: packet and router

5.1 Packet

For the packet.py, we firstly define Hello packet, Update packet, Data packet, and ACK packet with corresponding packet types and header, then we define how to read, send, and receive these packets. For the data packet part, we are introducing three destinations, even though some of them are not always valid destinations , and k values for servers that send data packets. For the receiving packet function, we set the timeout to 0.05 seconds, or 50 milliseconds which is provided in Appendix, and if timeout occurs, the loop will break. The function also determines the received packet type, as we defined, if the packet type equals to 2, it is a good packet and is recorded and forwarded; however, if the packet type is 3, the function will break. The multicast function is implemented as well, and creates and sends out our data packet to end up at our destinations based on our multicast algorithm.

5.2 Router

For the router,py, we are implementing multicast routing and Dijkstra's algorithm into it. First, we let each router know the information of other routers, define the function that can send packets to corresponding dst addresses. Next, we define the handle function to complete the multicasting: if packet type is 0, we populate graph with immediate neighbors; if packet type is 1, we populate graph with rest routers; if packet type is 2, we start to route by k-out-of-n. The exact code implementation is: if 2 of 3 dst are not reachable, the left one is the only dst. If 1 dst is not reachable, we compare the distances from the graph to left two dsts by Dijkstra's algorithm and pick up the dst that has the shortest distance. Under this condition, when there is only one server (k=1), the routing is straightforward, but if we have two servers (k=2), we route packets to the same dst unless the distances are not the same. Finally, if all dsts are reachable, it is similar for k=1,2 that comparing distances and routing to what we have done for 2 dsts available; for k = 3, we sort the distances from servers to dsts, compare, and decide the dst for each server: if path 1,2,3 is the same, the routing is straightforward; if path 1 equals to

path 2, server 1 and 3 will send packets to dst1 and dst3; if path 1 equals to path 3, server 1 and 2 will send packets to dst1 and dst 2; if path 2 equals to path 3, server 2 and 1 will send packets to dst 2 and dst1; if path 1,2,3 is not the same at all, server 1,2,3 will send packets to dst 1,2,3 correspondingly. Second part is Dijstra's algorithm, we are implementing it by comparing src and dst: if they are equal, we build the shortest path and show it in an inverted array, or we record the path by visiting neighbors and updating the distance. Finally for nodes that are not visited, we mark and record the lowest distance to complete Dijkstra's algorithm.

6 APPENDIX

1) Hello Packets will be exchanged every 2 seconds.
2) TTL, the maximum amount of time a packet may spend in the network is 255 seconds. This uses 8 bits.
3) The Stop-and-Wait Timeout time will be 50 milliseconds.