MSc Computer and Communication Networks (2019)

Coursework Assignment

Instructor:	Dr John Easton
Written by:	James J Nkhata
Student ID:	
Assignment Date:	14/03/2019
Report Delivery Date:	25/04/2019

FLOOD ROUTING

1.1 Purpose

This section of the assignment aimed at writing a programme in Python to simulate Flood Routing in a network with a minimum of 15 nodes and investigate the congestion caused by duplication of packets. The results obtained at each node were to be represented graphically, analysed, discussed and a proposal for a solution to reduce the duplication causing the congestion made.

The flood routing simulation was to work using discrete time intervals, where transmissions to a neighbour node were done on each time increment.

The nodes needed a buffer that could queue additional packets generated. The nodes were to be connected to each other through links and although the links were bi-directional, only one-way transmission could be made by each link during any time interval.

Each packet was to have a counter that represented its hop-count (the amount of times the packet could be copied over before being terminated). The counter was initialised at 10 (n = 10) and decremented by '1' on each hop the packet made from one node to another. If the counter reached '0', the packet was to be discarded.

1.2 Procedure

1.2.1 Flooding

Flooding worked by having a node forward its incoming packets to all its communication ports except for the one it arrived from. To have it redistributed, the packet had to be multiplied at the node and then forwarded to its next nodes. The exponential growth of these packets as they travelled from node to node and the restrictions of each node's packet forwarding ability led to duplication and congestion of the network. If left uncontrolled the packets would multiply continuously and render the network unusable.

Controls could be applied to the network to minimise the effect of packet exponential growth by giving each packet a time-to-live (TTL) or hop count. Another method to make sure the packets did not multiply indefinitely was for the packet to keep track of the nodes it had visited as stated by Leon-Garcia and Widjaja (2003).

1.2.2 Specifications

- Transmissions between nodes were to be done during time intervals and each link could only be used one way.
- The nodes were to copy the packets to their neighbour nodes except the one they received the packet from.
- The nodes were to have buffers to store any additional packets. The buffers were to work on a first in first out queue basis (FIFO queue).
- Each packet needed a counter for its time-to-live (n = 10), and this number had to be related to the network diameter. Network diameter was the minimum hop

number between two furthest nodes according to Leon-Garcia and Widjaja (2003).

- Packets were to be discarded once the counter reached zero.
- A packet was to be initialised from a node (inserted) to start the flooding algorithm.

1.2.3 Flood Routing Simulation Algorithm

The network topology had to take the initial hop count of 10 into consideration to fulfil the network diameter needed (minimum hop number between two furthest nodes). This meant the furthest nodes had to be about 10 connections away. To fulfil this requirement, 18 nodes were used and connected as shown in Figure 1.

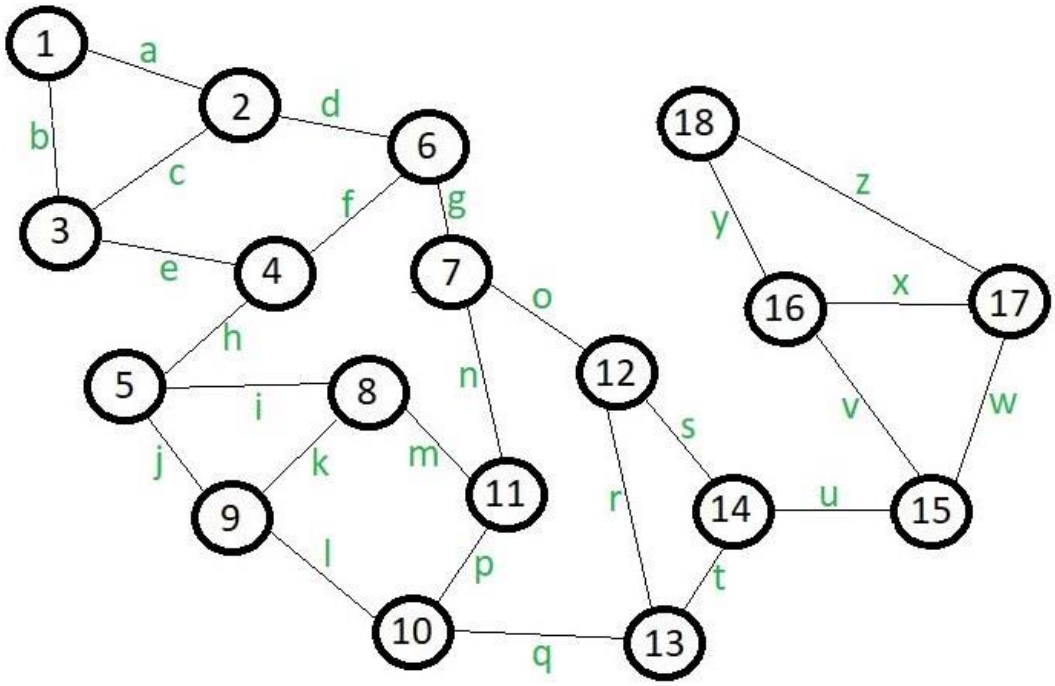

Figure 1: Proposed Network Topology of the Flood Routing Simulation

The nodes were represented by numbered circles and the links connecting them by the lines with characters “a” to “z” as depicted in Figure 1. To keep the network neat and uniformly structured a decision was made to have a maximum of 3 links per node.

Upon initializing the first packet within any node in Figure 1, its neighbours had to be identified to determine where the packet was to be copied to. Once this copy list was established, the packet was duplicated to be transmitted to the nodes in its copy list. The link connecting the current node to the next node was examined to see if it was being used by other nodes (since a link could be used for one-way transfer only during any time interval). If the link was in use, the copy process to that next node was suspended and that packet was modified (removed that copy list element from the current packet) and appended to the end of the current node’s buffer as a new packet until the next time interval.

When a packet was successfully transferred to the next node it was deleted from its current node and the hop count was decremented by 1 in the next node. The process was then repeated to copy the packet from its subsequent node, to its neighbour nodes (except for the node it came from) during the next time interval.

1.2.4 Implementation of Flood Routing Simulator in Python

1.2.4.1 State Chart for Flood Routing Simulation

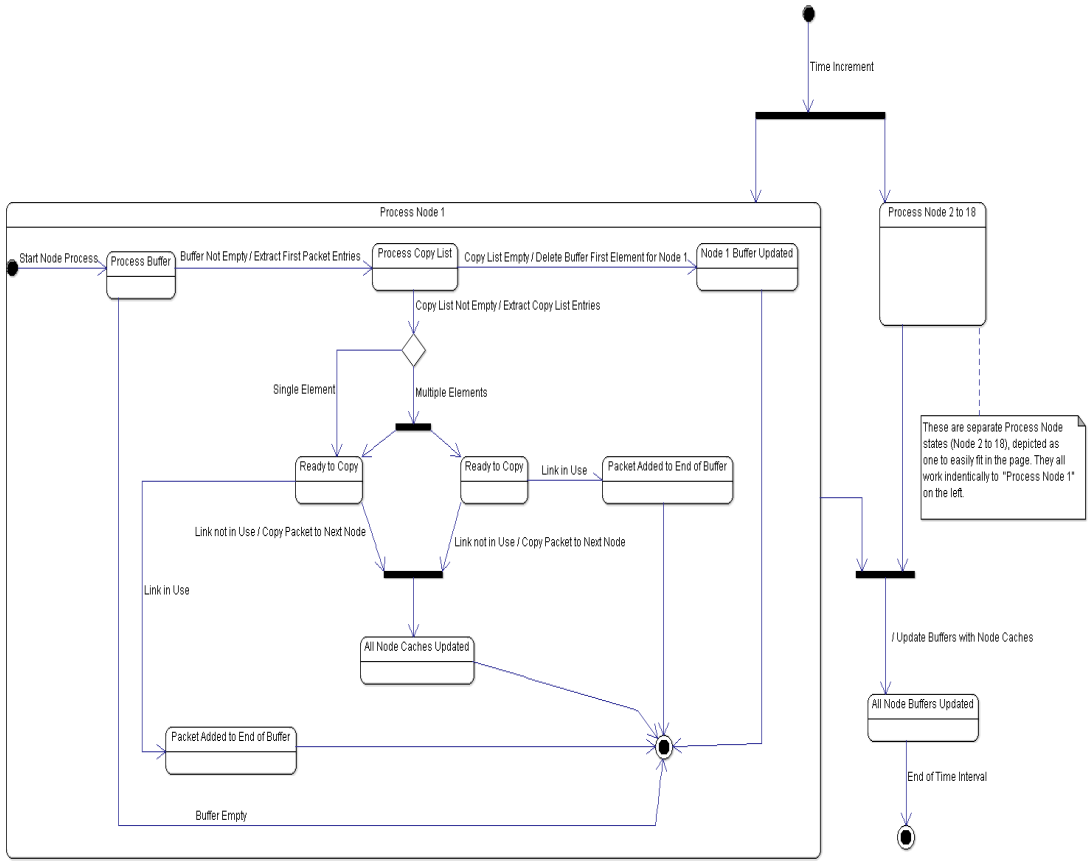

Figure 2: State Chart Diagram Depicting Concurrent Processes in Nodes 1 to 18 (Per Time Interval)

1.2.4.2 Main storage structures used

Decisions had to be made for the storage structures to be used for the system. The requirements were defined as follows:

- A storage structure had to hold information for node connections and the links between them.
- A storage structure had to hold packet information on a first in first out queue basis (FIFO queue) for the node buffers.

```
# ------------------------------ Defining Structure for Network Topology ------------------------------
# Nested dictionary holding the routing_table (map of connections between nodes and link names)
routing_table = {
    1: {2: "a", 3: "b"},  # - key "1" refers to current node, internal key "2" to next node and value "a" to link
    2: {1: "a", 3: "c", 6: "d"},
    3: {1: "b", 2: "c", 4: "e"},
    4: {3: "e", 5: "h", 6: "f"},
    5: {4: "h", 8: "i", 9: "j"},
    6: {2: "d", 4: "f", 7: "g"},
    7: {6: "g", 11: "n", 12: "o"},
    8: {5: "i", 9: "k", 11: "m"},
    9: {5: "j", 8: "k", 10: "l"},
    10: {9: "l", 11: "p", 13: "q"},
    11: {7: "n", 8: "m", 10: "p"},
    12: {7: "o", 13: "r", 14: "s"},
    13: {10: "q", 12: "r", 14: "t"},
    14: {12: "s", 13: "t", 15: "u"},
    15: {14: "u", 16: "v", 17: "w"},
    16: {15: "v", 17: "x", 18: "y"},
    17: {15: "w", 16: "x", 18: "z"},
    18: {16: "y", 17: "z"}
}
# Dictionary holding the status of links
link_status = {
    "a": False, "b": False, "c": False, "d": False, "e": False, "f": False, "g": False, "h": False, "i": False,
    "j": False, "k": False, "l": False, "m": False, "n": False, "o": False, "p": False, "q": False, "r": False,
    "s": False, "t": False, "u": False, "v": False, "w": False, "x": False, "y": False, "z": False
}

# Dictionary to hold packet information as they are added to to the Buffer
# N_Buffer = dict("name": {"packet_status_flag": FALSE, "hop_count": 0, "copy_list": [0], "packet_history": [0]})
N_Buffer = dict()

# Ordered List to simplify indexing and key information of N_Buffer
# Initialise an empty list of 18 lists to store the order of each nodes N_Buffer's entries
N_Buffer_OrderList = [[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
```

Figure 3: Choice of Main storage structures used to implement the Flood Routing Simulation Network

To fulfil the requirements for buffer storage two structures were used; “N_Buffer” and “N_Buffer_OrderList”. They were both made into class variables, so each Node class object could access them and change their entries.

“N_Buffer” was a nested dictionary that used a packet’s name string as its key and stored keys; “packet_status_flag”, “hop_count”, “copy_list” and “packet_history” as its entries. “packet_status_flag” was a key with the value that was used as a flag to indicate if a packet was processed to be copied to the next node or just newly copied to a node. “packet_history” held the list of nodes a packet had already visited and was used to prevent a packet from being copied back to a node it had already come from (by using the “remove_similar_nodes” static function). “copy_list” was used to hold the node numbers that a packet was to be copied to, this was determined by a function that extracted the information from the node objects (“node_process”).

```
# ---------------------------------- Creating Node objects to define Network ----------------------------------------
# Create Node objects using Tuples to define the connection structure between the nodes
# The values will not be changeable
Node_1 = Node(1, 2, 3)  # - 1st element in the Tuple defines the node number, the rest define its neighbour Nodes
Node_2 = Node(2, 1, 3, 6)
Node_3 = Node(3, 1, 2, 4)
Node_4 = Node(4, 3, 5, 6)
Node_5 = Node(5, 4, 8, 9)
Node_6 = Node(6, 2, 4, 7)
Node_7 = Node(7, 6, 11, 12)
Node_8 = Node(8, 5, 9, 11)
Node_9 = Node(9, 5, 8, 10)
Node_10 = Node(10, 9, 11, 13)
Node_11 = Node(11, 7, 8, 10)
Node_12 = Node(12, 7, 13, 14)
Node_13 = Node(13, 10, 12, 14)
Node_14 = Node(14, 12, 13, 15)
Node_15 = Node(15, 14, 16, 17)
Node_16 = Node(16, 15, 17, 18)
Node_17 = Node(17, 15, 16, 18)
Node_18 = Node(18, 16, 17)
#
```

Figure 4: Node objects created by passing neighbour node numbers to the Node Class __init__ function

“N_Buffer_OrderList” was a list containing 18 lists (multidimensional). Each internal list belonged to node 1 to 18 (indexed 0 to 17). The “N_Buffer_OrderList” was used to store the packet name strings and the order they were added to the node’s buffer (list).

Put together “N_Buffer” held the packet name string key with its entries (more keys and their corresponding values), while the “N_Buffer_OrderList” held the same packet name string in the order it was added to a node’s buffer. This gave a solution to storing packet information, the order it was added to node’s buffer and a way to access the information by way of packet name string. Python’s OrderedDict module could have been used to store dictionary entries in the order of being added but it was going to complicate intermediate node processes.

The packet name string adopted a naming convention that used the string prefix “pkt” concatenated to the “initiated node number” followed by “the nodes the packet had visited”. This guaranteed that packet name strings were unique and did not overwrite entries in the “N_Buffer” nested dictionary.

The “routing_table” and “link_status” was also made into dictionaries. Nested dictionary “routing_table” held the node numbering as its keys (1 - 18), and its internal keys as references to the next node. This gave a method to map node numbers to their corresponding next nodes and their values as the name of the links between them (current node and next node). The “routing_table” map was as depicted by Figure 3.

Dictionary “link_status” held the links between the nodes as its keys and their status as its values in the network (“False” to indicate the link not in use and “True” to indicate that link was in use).

1.2.4.3 User Interface to represent node buffer information

Python's TkInter package was used to develop the user interface for the Flood Routing Simulation program.

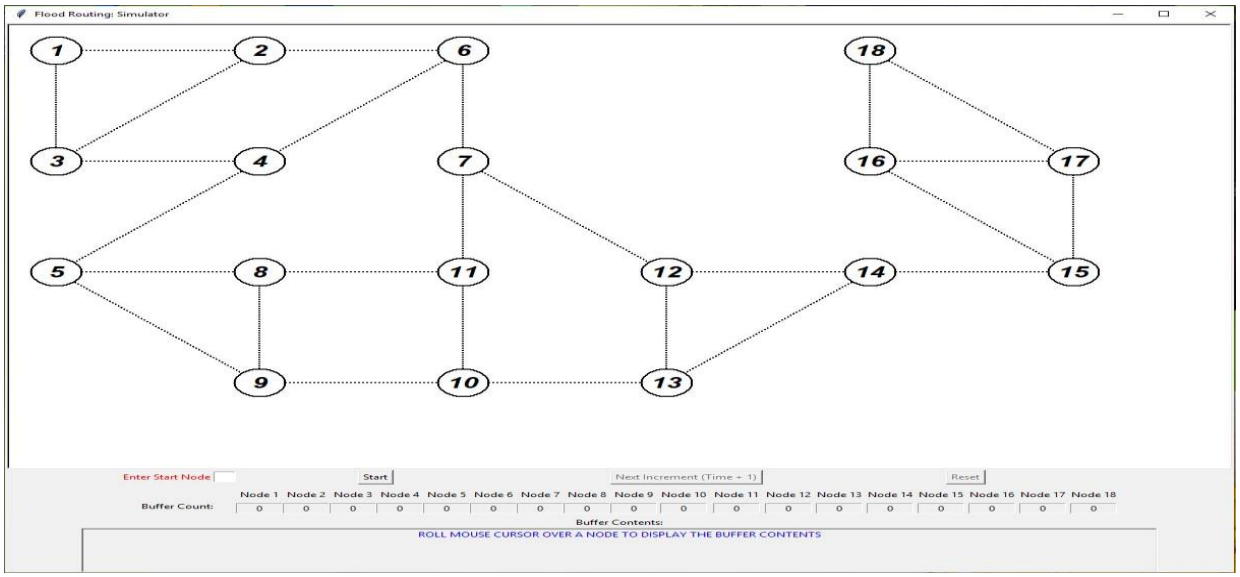

Figure 5: Screenshot of Python's User interface for Flood Routing Simulator using TkInter

When the "Main.py" script was running, the interface shown in Figure 5 was displayed to the user.

The nodes were represented by the circles with the numbers in the middle on the canvas (the rectangle with the white background). The lines connecting the circles represented the links between the nodes. The nodes could only copy to and receive packets from the nodes they were immediately connected with.

Below the canvas (bottom rectangle) showed the "Enter Start Node" entry field, "Start" button, "Next Increment (Time + 1)" button, "Reset" button, Node 1 to Node 18 headers with label fields directly below ("Buffer Packet Count:" row label) and a "Buffer Contents:" label with a field label below it that span the page.

Table 1: Items found on the Flood Routing Simulator User Interface and their purpose

Item	Purpose
"Enter Start Node" entry field	The field was used to select the node to insert the initial packet by specifying it with an integer value (1-18)
"Start" button	The button used to start the flood routing process once a start node was entered
"Next Increment (Time + 1)" button	The button used to increment time by the value of 1
"Reset" button	The button used to restart the program to insert a new packet in a different start node
"Buffer Packet Count:" labels below Node 1 to Node 18 headers	Displayed the number of packets contained within the node's buffer at that time
"Buffer Contents:" (bottom label field)	Displayed the packet name strings of the packets contained within the node that had been highlighted (by moving the mouse cursor over its corresponding node on the canvas)

1.3 Experimental data

1.3.1 A packet inserted at Node 1

The Flood Routing “Main.py” script was executed to bring up the user interface. The integer “1” was entered in the “Enter Start Node” entry field (to insert the initial packet in Node 1’s buffer) and the “Start” button was pressed.

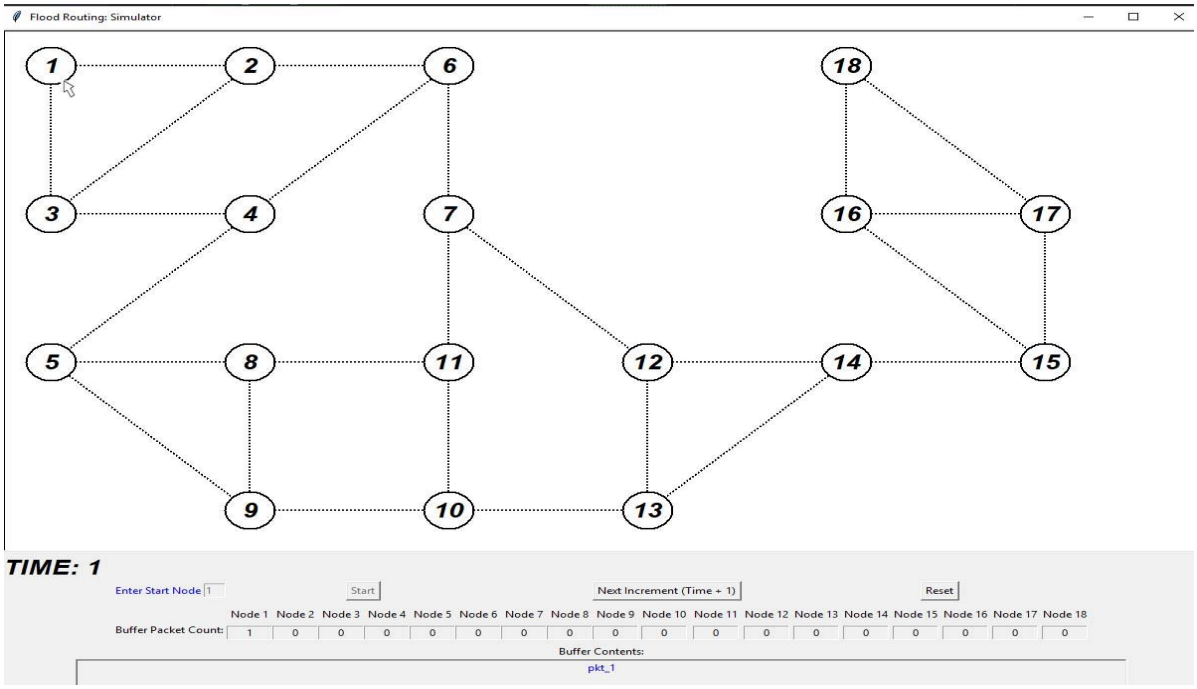

Figure 6: Screenshot of the user interface with a Packet inserted into Node 1 Buffer with Buffer Contents

The results obtained were as shown in Figure 6. The interface displayed “TIME: 1”, “Buffer Packet Count:” for Node 1 was “1” with “0” for Nodes 2 to 18 and the “Buffer Contents:” for node 1 was “pkt_1” (by having the cursor over the circle depicting Node 1 on the canvas). This was an indication that the flooding algorithm was correctly initiated as no packets were distributed yet at “Time: 1”.

1.3.2 Packets at “Time: 2”

The “Next Increment (Time + 1)” button was pressed to simulate a time increment.

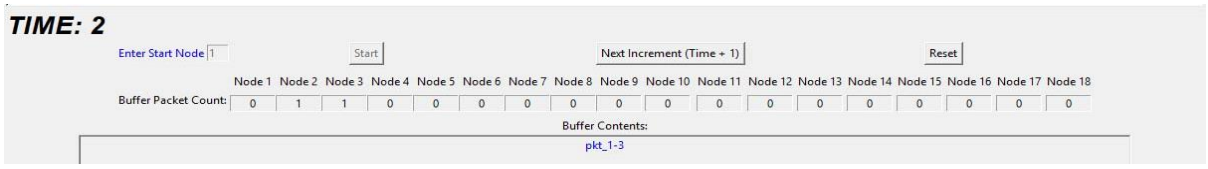

Figure 7: Screenshot of the bottom half of the user interface at Time: 2 with Node 3 Buffer Contents

At “TIME: 2” the packet in Node 1 was deleted while Node 2 and Node 3 had “Buffer Packet Count:” at “1” packet each as shown in Figure 7. Upon moving the cursor over Node 2 the “Buffer Contents:” had “pkt_1-2” and Node 3 had “pkt_1-3”. The rest of the Nodes (4 to 18) had “0” indicating that there were still empty.

This showed that Node 1 had multiplied the packet in its buffer successfully and sent it to Node 2 and Node 3. The results validated that the simulation behaved as required at this point.

1.3.3 Packets at “Time: 3”

“Next Increment (Time + 1)” button was pressed to increment time from 2 to 3.

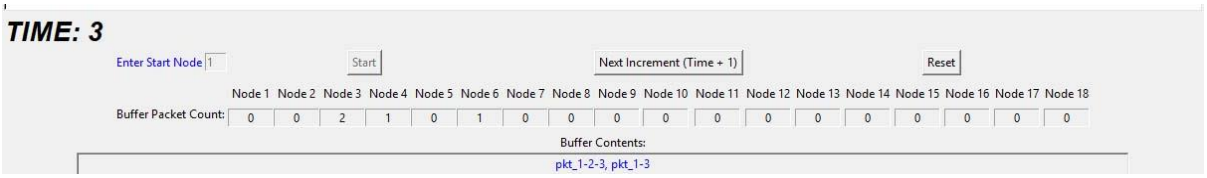

Figure 8: Screenshot of the bottom half of the user interface at Time: 3 with Node 3 Buffer Contents

At “TIME: 3” the “Buffer Packet Count:” in Node 2 was “0” (packet deleted), “Buffer Packet Count:” in Node 3 went up to “2”, Node 6 was “1“ and that in Node 4 was “1”. Upon inspecting the packets in Node 3 by moving the mouse cursor over the circle item with the number 3 on the canvas, “Buffer Contents:” was “pkt_1-2-3, pkt_1-3”. The same inspection was done for Node 4 with its “Buffer Contents:” displaying “pkt_1-3-4” and Node 6’s “Buffer Contents being:” displaying “pkt_1-2-6”.

The results obtained showed that Node 2 had deleted its “pkt_1-2” from “TIME: 2” and managed to copy it to its neighbour nodes 3 and 6 (“pkt_1-2-3” in Node 3 and “pkt_1-2-6” in Node 6). Node 3 managed to send one packet to Node 4 (“pkt_1-3-4”) but failed to send the packet to Node 2 as shown by the packet (“pkt_1-3”) still in Node 3. This was because Node 2 was using the link between 2 and 3 to send its packet (“pkt_1-2-3”) to Node 3 (a link between two nodes could only be used one way at any time interval).

These results verified that the program was obeying the one-way link usage and working as intended by the implemented algorithm for flood routing.

1.3.4 Packets at “Time: 9” (Packets arrived at Node 18)

Time incremented until the all the nodes received a packet.

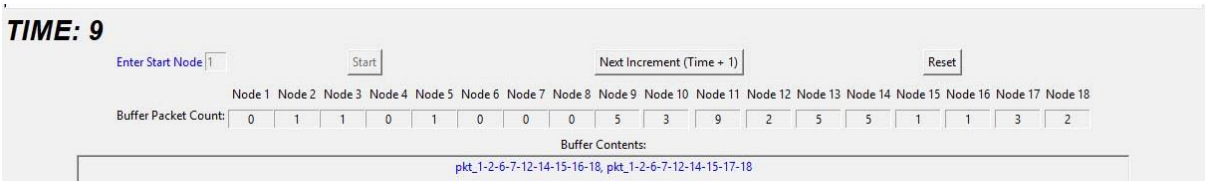

Figure 9: Screenshot of the bottom half of the user interface at Time: 9 with Node 18 Buffer Contents

At “TIME: 9” Node 18 had “2”, Node 2 had “1”, “1” was in Node 3, “1” was in Node 5, “5” was in Node 9, “3” was in Node 10, “9” was in Node 11, “2” was in Node 12, “5” was in Node 13, “5” was in Node 14, “1” was in Node 15, “1” was in Node 16 and “3” was in Node 17.

From Figure 9, the “Buffer Contents:” of Node 18 had two packets (“pkt_1-2-6-7-12-14-15-16-18, pkt_1-2-6-7-12-14-15-17-18”). The numbers in the the packet name indicated all the nodes each packet had visited from Node 1 to Node 18.

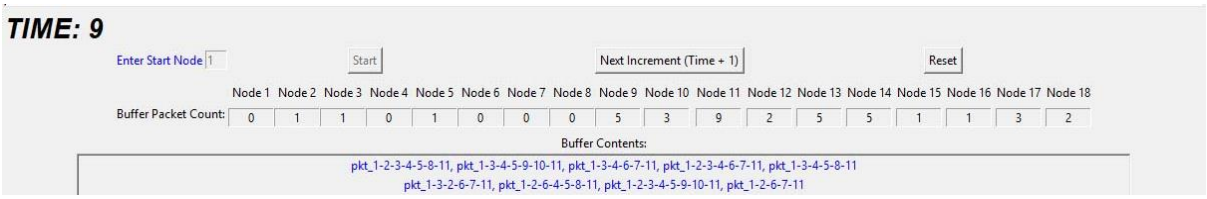

Figure 10: Screenshot of the bottom half of the user interface at Time: 9 with Node 11 Buffer Contents

Node 11's "Buffer Contents:" had 9 packets as shown in Figure 10, with each packet giving information to trace it's journey from the initial packet started at Node 1 to the current Node (by its packet name). Node 2, 3, 9, 10, 12, 13, 14, 15, 16 and 17 also contained packets that could be traced from the initial packet started at Node 1. Although Node 18 had received 2 packets by "TIME: 9", the network still had more packets across the it's buffers.

These results demonstrated the effects of flooding in the network and the duplicate packets it produced, even though the packet had been delivered to all nodes.

1.3.5 Packets at "Time: 68" (no more packets getting transferred)

Time was incremented until packets no longer copied between nodes.

Figure 11: Side-by-Side Comparison of Node 2 Buffer Contents Comparison at Time: 9 and Time: 68

The packet in Node 2's "Buffer Contents:" was inspected at "Time: 9" and "Time: 68". Both instances showed the same packet name ("pkt_1-3-4-6-2"). The packet started off from Node 1, moved to Node 3, then Node 4, then 6 and finally 2. Although time was incremented, the packet seized to move. This was because "pkt_1-3-4-6-2" had no more neighbour nodes it could copy to as it had exhausted its copy list, in this case Node 1, 3, 4 and 6.

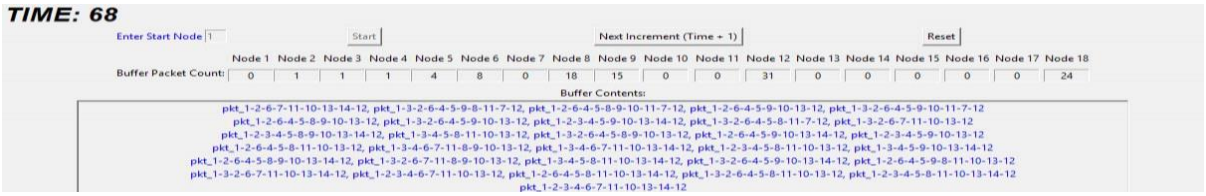

Figure 12: Screenshot of the bottom half of the user interface at Time: 68 with Node 12 Buffer Contents

The effects observed at Node 2 "Buffer Contents:" during "Time: 68" of packet's not moving from the buffer was also experienced in Nodes 3, 4, 5, 6, 8, 9, 12 and 18. This contributed to the duplicate packet numbers of "Buffer Packet Count:" across the nodes of the network.

Node 12 had the highest "Buffer Packet Count:" as shown in Figure 12, followed by Node 18, 8 and 9.

The other reason why the "Buffer Packet Count:" was high in some of the network nodes was because the algorithm implemented did not account for what to do with the packets once they had been delivered to a node. This led to the results with nodes having multiple packets stored in their buffers, as demonstrated in Figure 13.

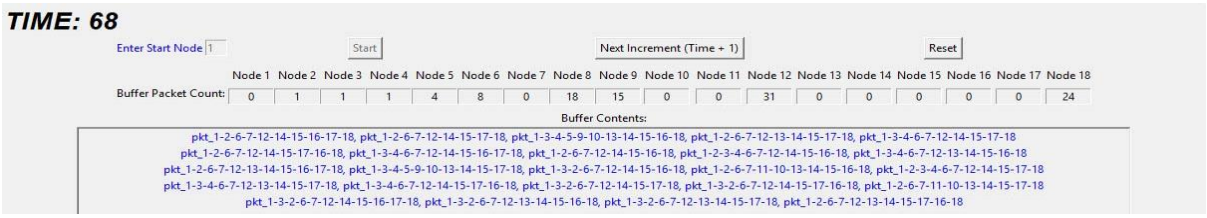

Figure 13: Screenshot of the bottom half of the user interface at Time: 68 with Node 18 Buffer Contents

1.4 Conclusion

In this report a programme was written in Python to simulate Flood Routing in a network with 18 nodes.

The Buffer Count and Buffer Contents for each node was represented graphically through the program's user interface. As gradual time increments were made the Buffer Count and Buffer Contents were inspected to verify the correct behaviour expected for packet copying, packet deletion and one-way use of links per time interval. The results from the experimental data showed that the program worked as intended by the proposed flood routing algorithm.

Despite one packet was inserted into the network (Node 1) during initialisation and it was able to reach all the nodes in the network by "Time: 9", the results obtained in the experimental data indicated several duplications of the initial packet were still in some node buffers across the network. These duplications were caused by the networks inability to remove packets once they had reached all nodes, the network holding on to packets that had exhausted all their copying destinations and finally the nature of the flood routing algorithm (even though some controls to minimise the duplication were implemented). This further verified the effects of flood routing and the duplication of packets it causes which would lead to network congestion.

A solution to the duplication of the packets caused by the algorithm could be to add an identifying marker to the initial packet before it is distributed. The nodes could be programmed to recognise the marker and distribute the packet as normal. On the chance a node encounters another packet with the same marker after distribution, it would discard it as it would be a duplicate of a previous packet according to Leon-Garcia and Widjaja (2003). Another way to address the duplicate packet issue would be to have a packet destination node specified before distributing a packet in the network. This node would not be allowed to distribute any packets once it received its packets.

A form of selective flooding could also be included in the solution. This way the distributing node would not send its incoming packet through all its outgoing links, instead only to the links that go approximately to the direction of the destination node according to Claypool and Kannan (2001).

References

Easton, J (2019) **Computer and Communications Networks** [Online]. Available from: https://canvas.bham.ac.uk/courses/33792 [Accessed 14 March 2019].

Leon-Garcia, A. and Widjaja, I. (2003) **Communication Networks: Fundamental Concepts and Key Architectures**.2nd ed. McGraw-Hill Education.

Videonations, ('no date') **Video Conferencing Network Requirements** [Online]. Available from: https://www.videonations.co.uk/resources/video-conferencing-news/video-conferencing-network-requirements [Accessed 20 April 2019].

Claypool, M. and Kannan, G. (2001) **Selective flooding for improved quality-of-service routing** [Online]. Available from: https://doi.org/10.1117/12.434367 [Accessed 23 April 2019].