Network Flood Routing Simulator (18 Nodes)

## Purpose

The assignment aimed at code 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.

## Procedure

### **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).

### **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.

### **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.

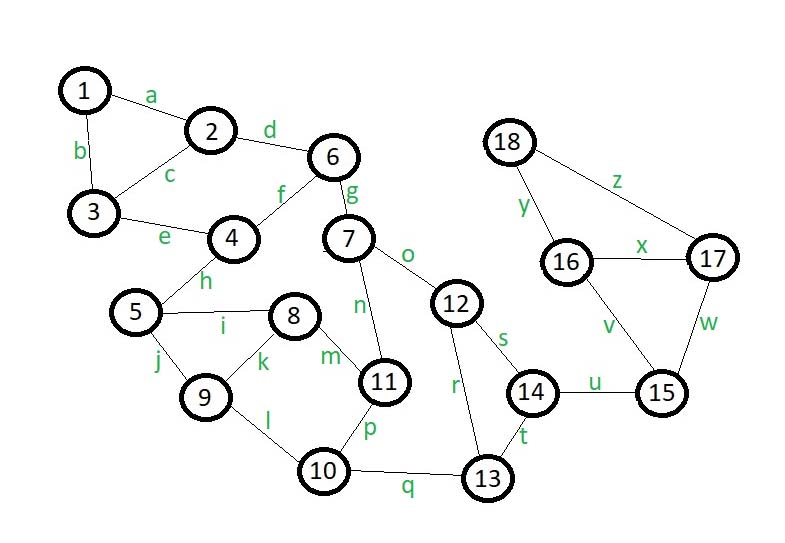


Figure 1: Proposed Network Topology of the Flood Routing Simulation

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.

### **Implementation of Flood Routing Simulator in Python**

#### State Chart for Flood Routing Simulation

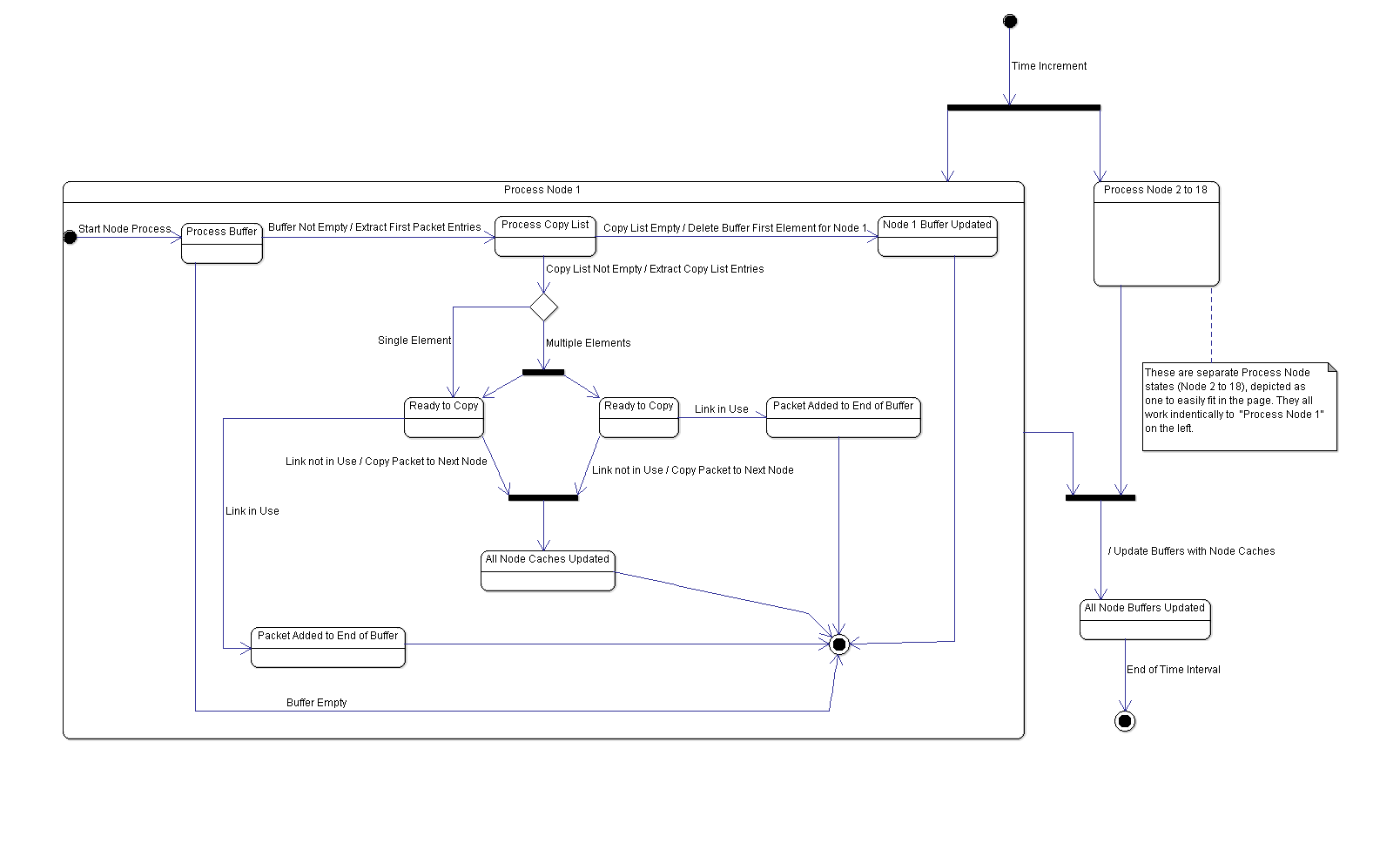


Figure 2: State Chart Diagram Depicting Concurrent Processes in Nodes 1 to 18 (Per Time Interval)

#### 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.

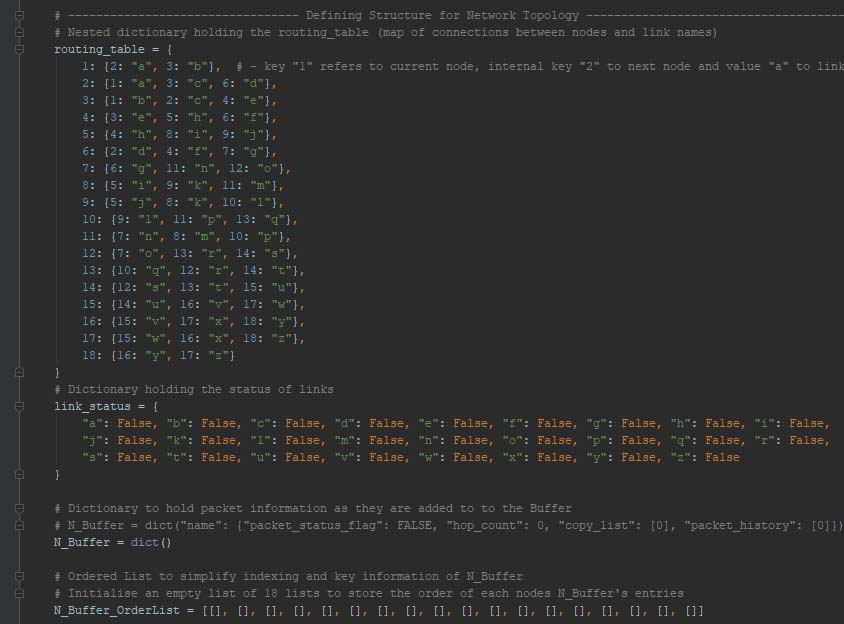


Figure 3: Choice of Main storage structures used to implement the Flood Routing Simulation Network

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”).

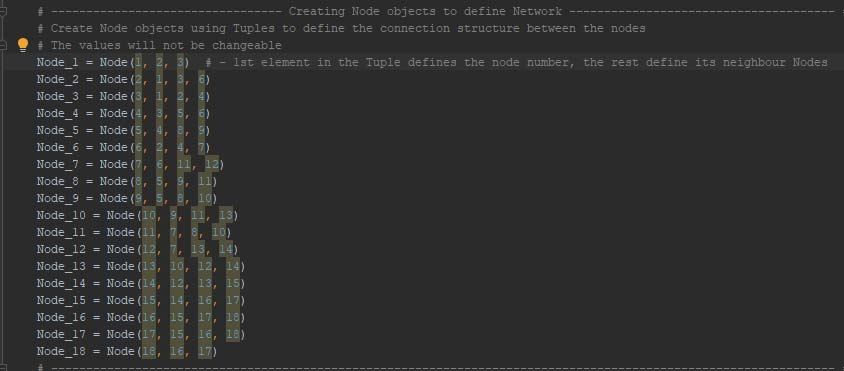


Figure 4: Node objects created by passing neighbour node numbers to the Node Class \_\_init\_\_ function

“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).

#### User Interface to represent node buffer information

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

#### 

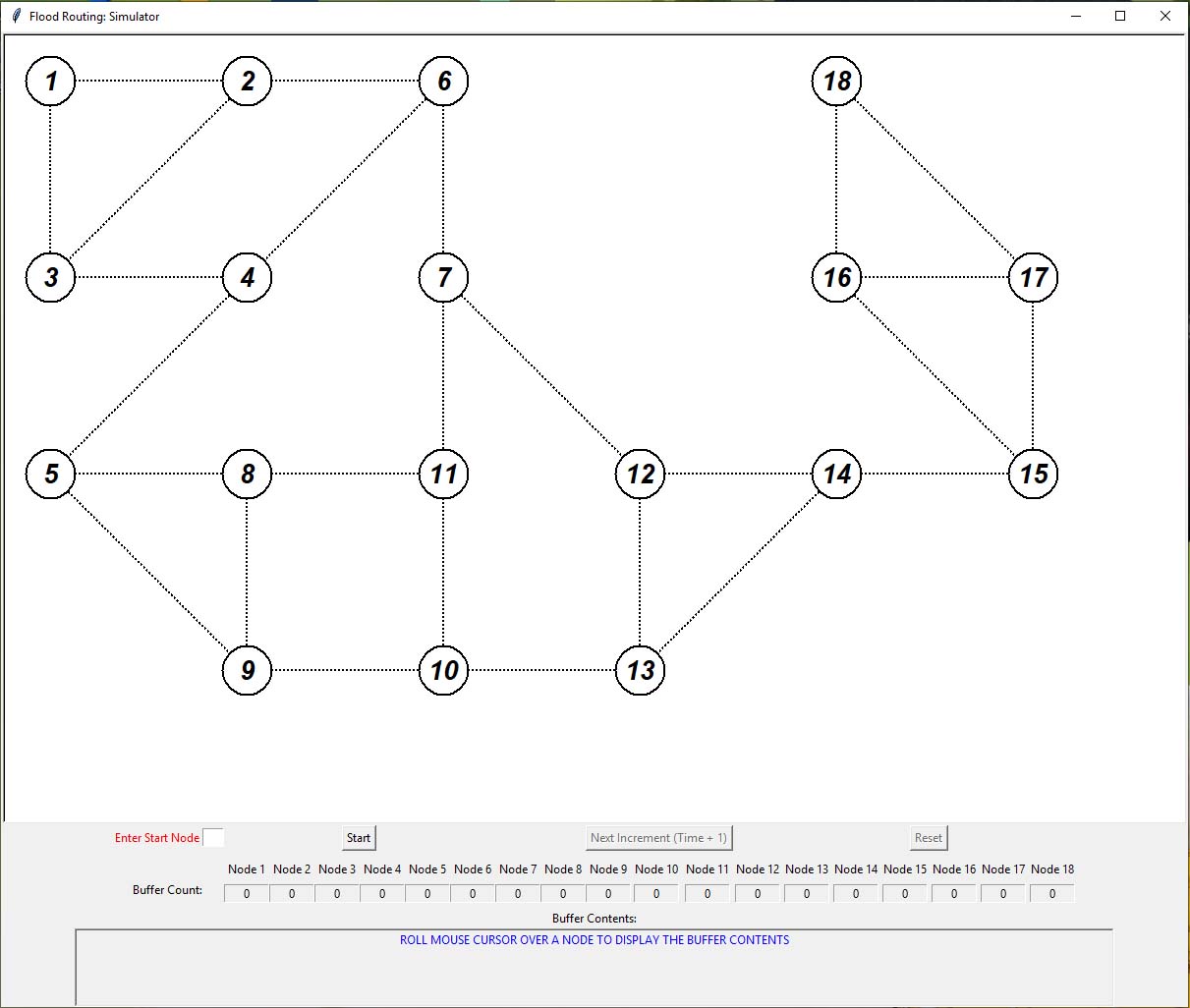


Figure 5: Screenshot of Python’s User interface for Flood Routing Simulator using TkInter

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) |

## Implementation

### **Software**

**\*Declarations\***

## 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.

Github link to repository for project