**ELC 2080**

**FreeRTOS Network Switch Simulator**

|  |  |  |  |
| --- | --- | --- | --- |
| BN | Section | Student ID | اسم الطالب |
| **37** | **1** | **9230242** | **أمير اشرف لولي عباس** |
| **38** | **1** | **9230243** | **أمير سامح فانوس عطية** |

# System Design

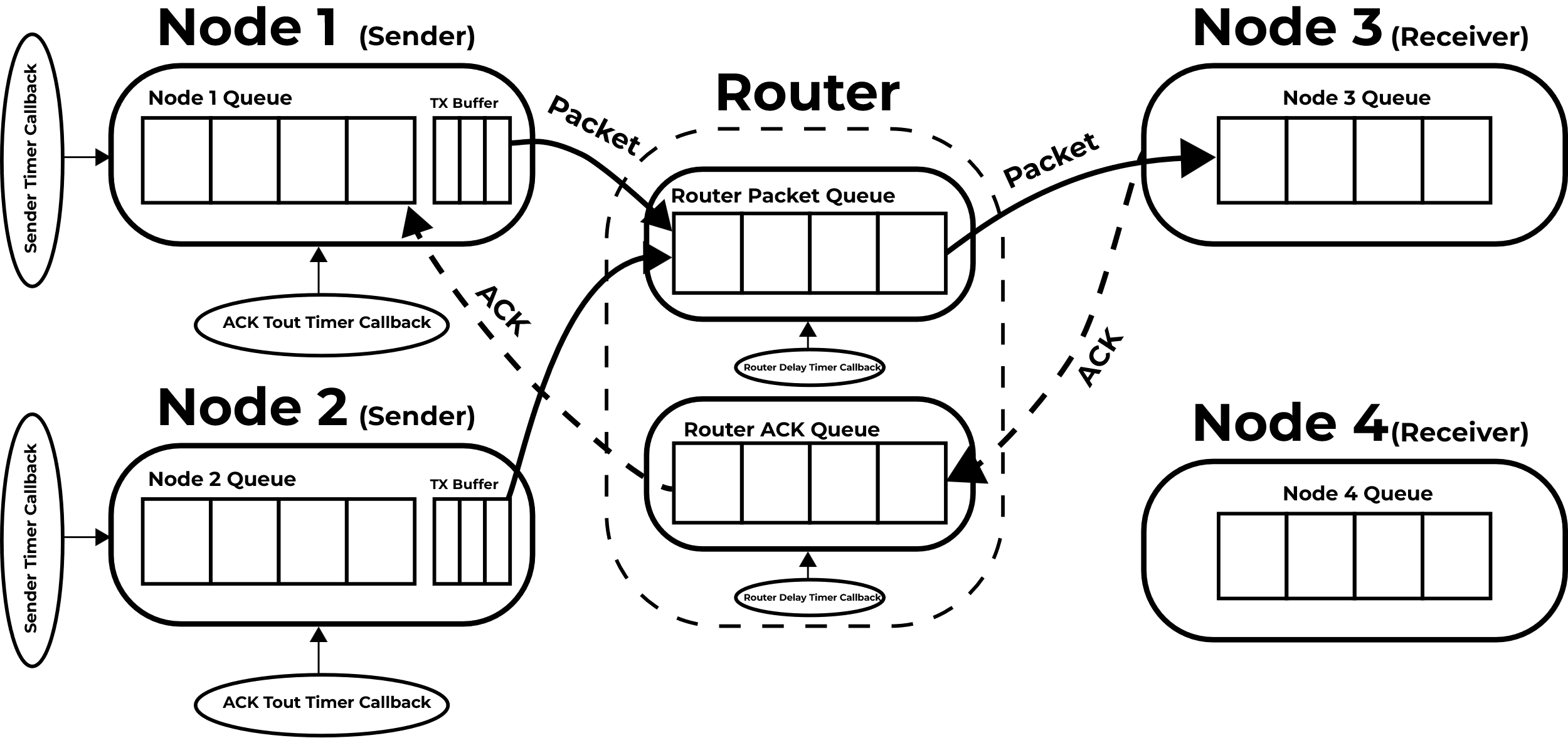


Figure 1 System Design

## Main Components:

**Sender Nodes:** The 2 Nodes Generate Packets according to a random period between [100, 200] msec to a random node (3 or 4) and a sequence number incrementing with each new generated packet corresponding to each receiver nodes. They send the Generated Packet to the router queue and store it in a Buffer while they await an ACK from the receiver. If the ACK is late for more than period Tout, the sender attempts to send the Packet again. This is repeated 4 times before the sender clears the buffer and moves on to the next packet.

**Receiver Nodes:** The 2 Receiver Nodes check if the packets they received are meant to the current node, if so, they check the sequence number and compare it to the last sequence node received from its corresponding node to calculate lost packets. It then sends an ACK regardless notifying the Sender of Successful receiving of the packet.

**Router (Switch) Node:** The Router handles transmitting packets between Senders and Receivers on its single queue to handle packets one by one; it has a probability Pdrop to drop Packets and P\_ACK to drop ACKs. Additionally for realism. The Router employs a delay with each packet it receives before transmitting it to its subsequent receiver according to L \* 8 / C.

**Timer Callbacks:** Each delay or period is simulated in the Program using a timer that unblocks the task by releasing its corresponding semaphore (Semaphore used as a signaling mechanism), The Task starts the timer and attempts to take a taken semaphore causing it to be blocked and staying in a blocked state till the timer callback is executed releasing the semaphore and unblocking the task.

A diagram of data

AI-generated content may be incorrect.**Packets:** All packets are dynamically allocated and use the same structure as seen in Figure 2, The payload part is allocated independently to account for the variable length of the packet generated, ACKs are the same structure but have a shorter length of 40 but are treated the same way. (Code Snippet 1)

**Mutual Exclusion:** A ”GeneratePacket” semaphore is used to at every packet generation or freeing operation to protect memory from corruption during these operations ensuring thread safety. (Code Snippet 2)

**Random Process:** The random process used multiple times in the system for packet generation and probability is simply done using rand() function from stdlib.h and seeded using srand(time(NULL)) utilizing the present time the system ran at to ensure a different seed for each run.

Figure 2 Packet Shape

## Task Flow-Charts:

Figure 5 Receiver Node Flow chart

Figure 3 Sender Node Flow Chart

Figure 4 Router Node Flow Chart

# Results and Discussion:

**2.1 S&W Results:**

We ran the system for a little more than 2000 received packets as each Receiver Node to see the system behaviour at the following values for Pdrop and Tout. Pdrop = {0.01, 0.02, 0.04, 0.08}, Tout = {150, 175, 200, 225} msec.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P\_Drop** | **T\_OUT** | **Time Elapsed (Sec)** | Total Packets Received | Total Packets Successful | Total Packets Failed | Total Bytes Sent | Total Bytes Successful | Total Bytes Failed | ThroughPut (Bytes/Sec) |
| 0.01 | 150 | 793 | 4981 | 4862 | 119 | 4965330 | 4838395 | 126935 | **6101.38** |
| 175 | 757 | 4994 | 4990 | 4 | 5012577 | 5008257 | 4320 | **6615.93** |
| 200 | 729 | 4956 | 4956 | 0 | 4964384 | 4964384 | 0 | **6809.85** |
| 225 | 678 | 4981 | 4980 | 1 | 4976764 | 4975357 | 1407 | **7338.28** |
| 0.02 | 150 | 806 | 4980 | 4804 | 176 | 5001978 | 4814863 | 187115 | **5973.78** |
| 175 | 750 | 4965 | 4959 | 6 | 4973372 | 4966633 | 6739 | **6622.18** |
| 200 | 724 | 4895 | 4891 | 4 | 4916255 | 4910976 | 5279 | **6783.12** |
| 225 | 681 | 4972 | 4972 | 0 | 4970731 | 4970731 | 0 | **7299.16** |
| 0.04 | 150 | 793 | 4994 | 4866 | 128 | 4989264 | 4855459 | 133805 | **6122.90** |
| 175 | 756 | 4950 | 4934 | 16 | 4994878 | 4976951 | 17927 | **6583.27** |
| 200 | 724 | 4887 | 4886 | 1 | 4932804 | 4932241 | 563 | **6812.49** |
| 225 | 677 | 4982 | 4982 | 0 | 4980553 | 4980553 | 0 | **7356.80** |
| 0.08 | 150 | 789 | 4950 | 4835 | 115 | 2700844 | 4823329 | 121255 | **6113.22** |
| 175 | 761 | 4999 | 4985 | 14 | 4971654 | 4955330 | 16324 | **6511.60** |
| 200 | 749 | 4953 | 4943 | 10 | 4957398 | 4944979 | 12419 | **6602.11** |
| 225 | 697 | 4944 | 4994 | 0 | 4921169 | 4921169 | 0 | **7060.50** |

Table 1 S&W Results

### Result Analysis:

We can see the throughput of the system isn’t affected much by changing Pdrop. This is due to the ACK system and packet retransmission mitigating router drops and drops due to other reasons.

Throughput is heavily affected by changing Tout though and increases by increasing Tout. This is due to less re-transmission sent and ensuring only re-transmissions happen when actual drops occur from the router or otherwise. Giving us the highest throughput at the highest Tout of 150 msec.

**Question 1:** Avg Number of Transmissions of a Packet as a function of Pdrop.

To Calculate this, we did another run on all Pdrop values at an average Tout of 200ms to calculate the number of retransmissions which wasn’t accounted for in the original 16 Runs.

As we can see the Average number transmissions increase with increasing Pdrop albeit they are all very small values and landing closer to the minimum of the expected range, this is due to the efficiency of the S&W system as it is not need of many re-transmissions, it is expected to be higher though at higher values at lower Values of Tout.

**Question 2:** Packets dropped due to being transmitted more than 4 times can be seen in Table 1.

## Go-Back N results:

Our Implementation for the Go-Back N Protocol did not yeild

# Future Work:

Attempting this project again in the future we would like to experiment with using multiple Queues at the router nodes specifically a queue for each node that sends to the router a timer for each queue. This would allow the Queue to handle multiple packets during the same cycle and implement delays on each packet parallel to each other. This would increase the throughput of the system and better simulate a real-life network switch.

# Design demonstrations:

1. Packet Structs Decleration:

typedef uint8\_t Payload\_t;

typedef struct {

QueueHandle\_t sender;

QueueHandle\_t reciever;

SequenceNumber\_t sequenceNumber;

uint16\_t length;

uint16\_t padding; // To make sure the header 16 bytes

} header\_t;

typedef struct {

header\_t header;

Payload\_t\* data;

} packet;

1. Packet Generation:

xSemaphoreTake(GeneratePacket, portMAX\_DELAY);

/\* Generate and Send Packet when Semaphore is Taken \*/

PacketToSend = pvPortMalloc(sizeof(packet));

if(PacketToSend == NULL)

{

// Failed to Generate Packet, Trying Again

trace\_puts("Failed to Allocate Packet");

xSemaphoreGive(GeneratePacket);

continue;

}

PacketToSend->header.sender = CurrentNode->CurrentQueue;

switch(RandomNum(3, 4))

{

case 3:

PacketToSend->header.reciever = Node3Queue;

PacketToSend->header.sequenceNumber = ++SequenceToNode3;

break;

case 4:

PacketToSend->header.reciever = Node4Queue;

PacketToSend->header.sequenceNumber = ++SequenceToNode4;

break;

}

PacketToSend->header.length = RandomNum(L1, L2);

PacketToSend->data = pvPortMalloc((PacketToSend->header.length - sizeof(header\_t)) \* sizeof(Payload\_t));

if(PacketToSend->data == NULL)

{

trace\_puts("Failed to allocate data");

vPortFree(PacketToSend);

xSemaphoreGive(GeneratePacket);

continue;

}

// Semaphore is given at the end after packet is stored in buffer