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**

1 System Design

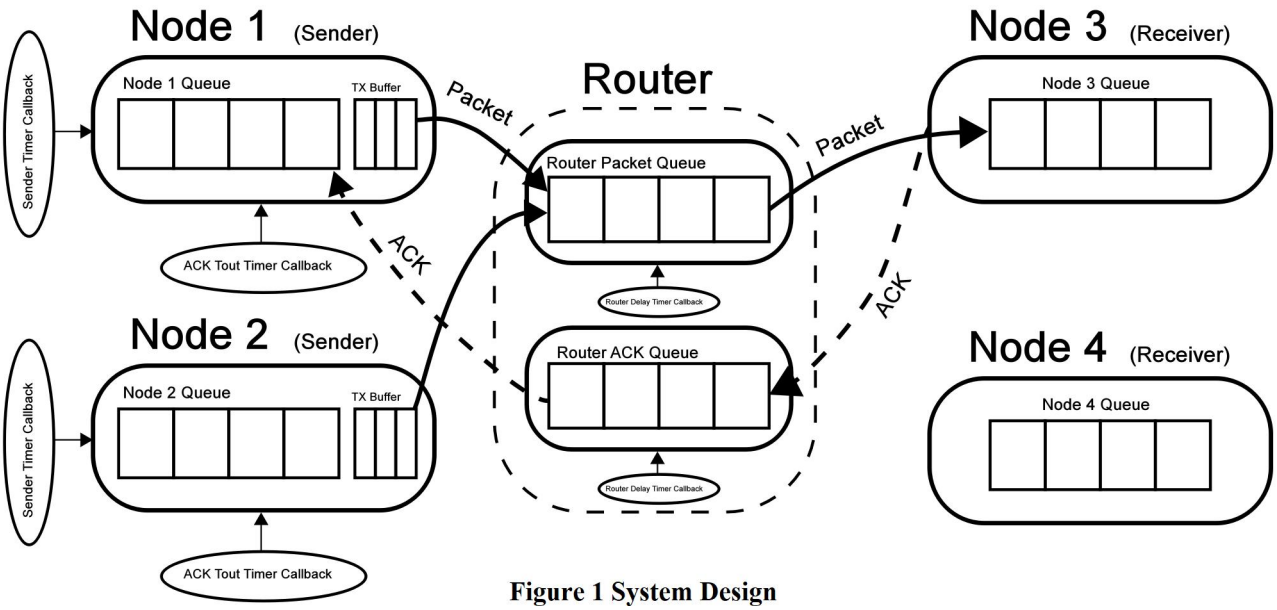


Figure 1 System Design

1.1 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/s 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 re-send the Packet/s starting from the first failure to the end of the buffer. This is repeated 4 times before the sender clears the buffer and moves on to the next packet/s.

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 only if no packets are lost 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 "D + (L * 8 / C)" D is the constant propagation delay, L is the Packet length and C is the constant Link Capacity.

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.

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.

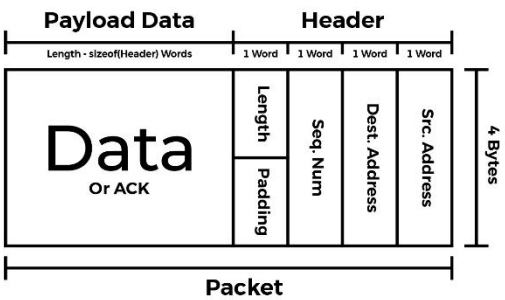


Figure 2 Packet Shape

1.2 Task Flow-Charts:

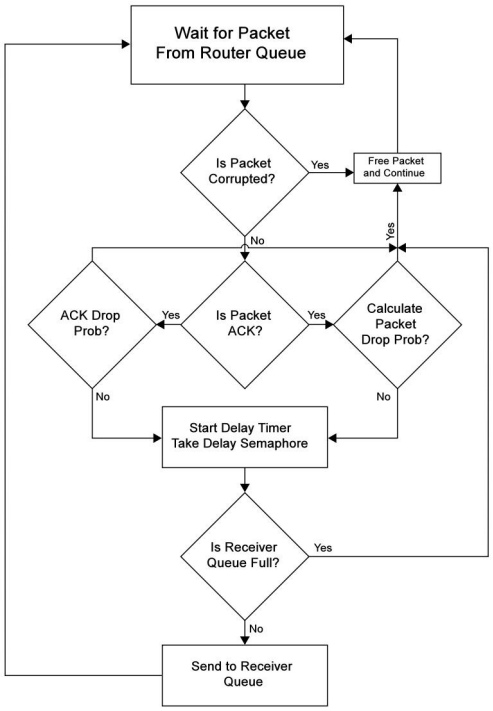


Figure 4 Router Node Flow Chart

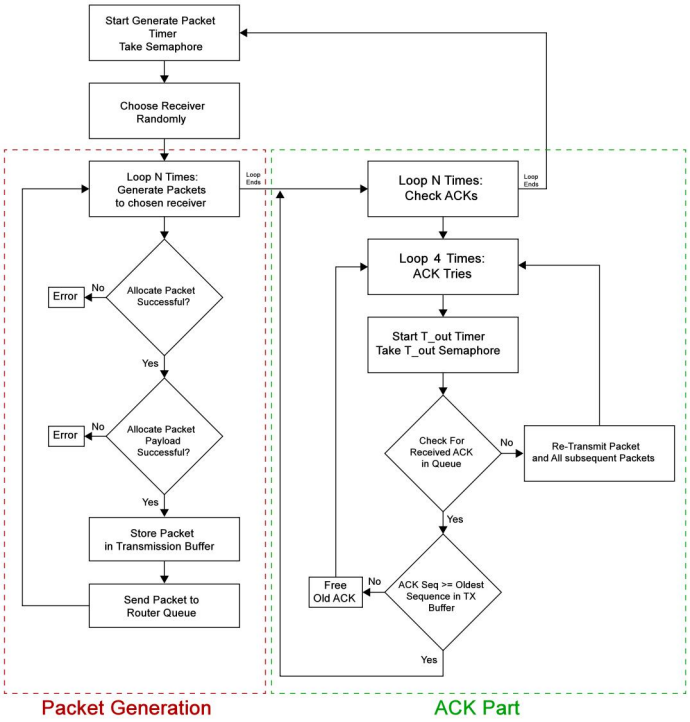


Figure 3 Sender Node Flow Chart

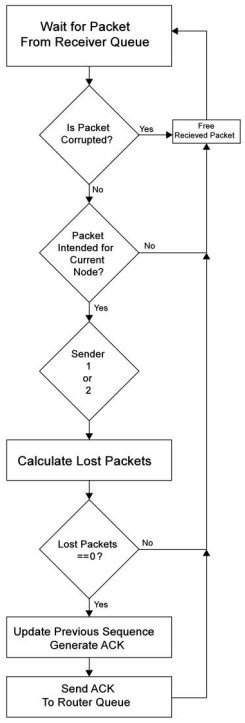


Figure 5 Receiver Node Flow chart

2 Results and Discussion:

2.1 S&W Results (Older Architecture):

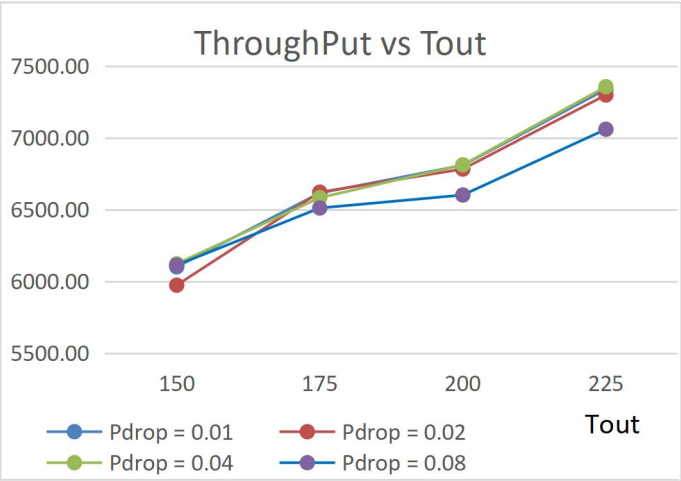
We ran the system for a little more than 2000 received packets as each Receiver Node to see the system behaviour at the following values for Pdrop and Tout. Pdrop = {0.01, 0.02, 0.04, 0.08}, Tout = {150, 175, 200, 225} msec.

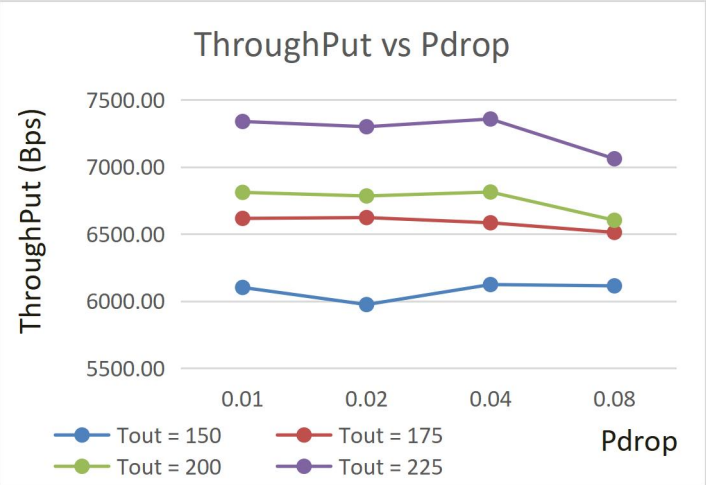
The following results were obtained on an older version of the system where the router had a single queue for processing both ACKs and normal Packets.

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	Thro (Byt
0.01	150	793	4981	4862	119	4965330	4838395	126935	61
	175	757	4994	4990	4	5012577	5008257	4320	66
	200	729	4956	4956	0	4964384	4964384	0	68
	225	678	4981	4980	1	4976764	4975357	1407	73
0.02	150	806	4980	4804	176	5001978	4814863	187115	59
	175	750	4965	4959	6	4973372	4966633	6739	66
	200	724	4895	4891	4	4916255	4910976	5279	67
	225	681	4972	4972	0	4970731	4970731	0	72
0.04	150	793	4994	4866	128	4989264	4855459	133805	61
	175	756	4950	4934	16	4994878	4976951	17927	65
	200	724	4887	4886	1	4932804	4932241	563	68

	225	677	4982	4982	0	4980553	4980553	0	73
0.08	150	789	4950	4835	115	2700844	4823329	121255	61
	175	761	4999	4985	14	4971654	4955330	16324	65
	200	749	4953	4943	10	4957398	4944979	12419	66
	225	697	4944	4994	0	4921169	4921169	0	70

Table 1 S&W Results





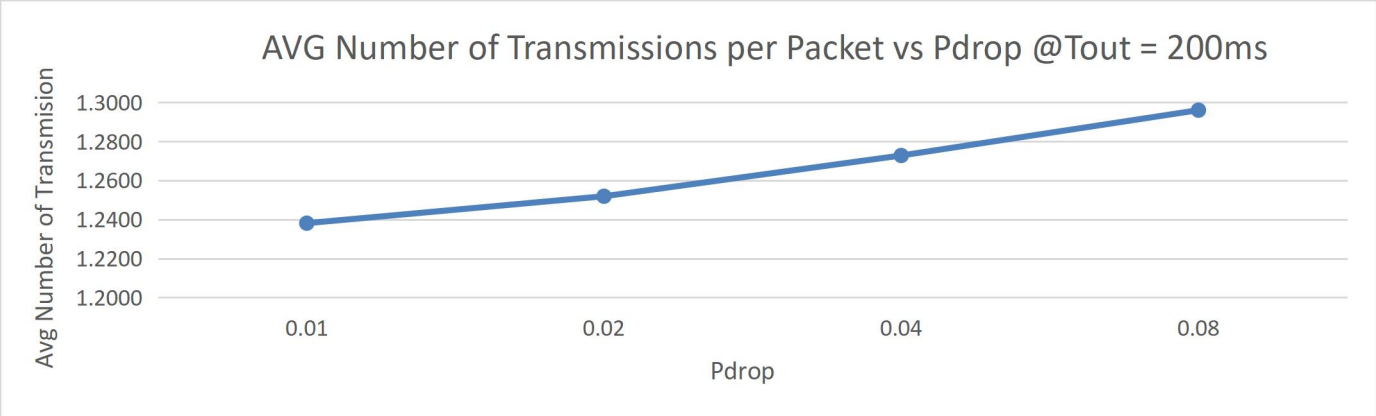
2.1.1 Result Analysis:

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. It should be noted that throughput won't increase indefinitely and **would start dropping after the optimum Tout is reached** as the system would be waiting for already received ACKs longer.

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.



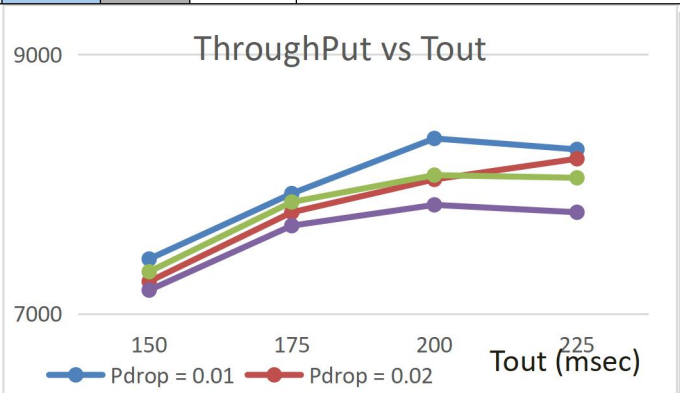
As we can see the Average number transmissions increase with increasing Pdrop albeit they are all very small values and landing closer to the minimum of the expected range, this is due to the efficiency of the S&W system as it is not in need of many re-transmissions, it is expected to be higher though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

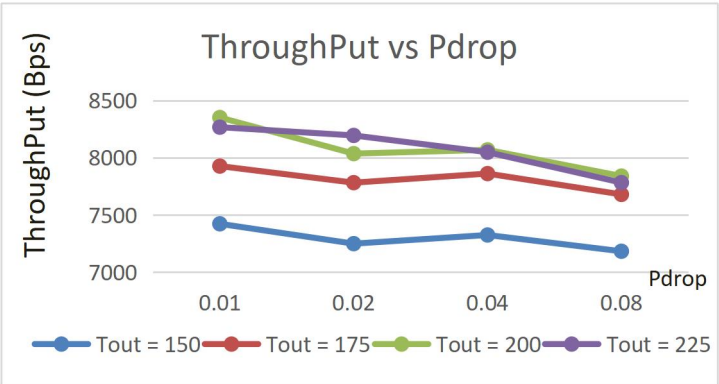
Question 2: Packets dropped due to being transmitted more than 4 times can be seen in Table 1.

2.2 S&W (Improved Architecture) :

We did another run with the improved Architecture for S&W with better bi-directional handling of Packets and ACKs, the results stayed mostly similar following the same trends but with improved overall throughput and zero lost packets, here's a summary.

N	P_Drop	T_OUT	Time Elapsed (Sec)	Total Packets Received	Total Packets Successful	Total Re-transimitted	Total Packets Failed	Total Bytes Sent	Total Bytes Successful	Total Bytes Failed	Through Put (Bytes/Sec)	Average Re-transmitted Packets
2	0.02	200	117	982	982	247	0	97876	978760	0	8365.47	1.2515
4	0.02	200	17	64	58	110	6	64274	57908	6366	3406.35	2.7188
8	0.02	200	4	16	16	38	0	14307	14307	0	3576.75	2.38





3 Go-Back-N Protocol:

For the Go-Back-N we built it upon the improved S&W architecture with the bi-directional handling of packets and ACKs. But our implementation of Go-Back-N experienced significant delays and crashes at lower Tout values of N bigger than 2. So we decided to test on a smaller range of values for Pdrop and Tout as we are constrained by time limits and ***running the simulation till failure*** instead of a pre-determined number of packets sent with a maximum of about 1000 packets per receiver sent. Here's a table summarizing our findings.

The third run with N=8 Completely Failed as the Memory Filled up quickly on the system and Malloc Failed halting the system entirely. We can also see the Throughput decreased at N = 4. The unexpected decrease in performance is probably due to the relatively *small Tout period and low performance of the Router Node*. As the number of packets sent in the patch increase more ACKs are required at a time and Tout is not long enough to account for router delays causing the Sender to re-transmit packets causing more memory allocation which quickly slows down the system or overwhelming it to a state of crash like we see at N = 8.

4 Future Work:

Attempting this project again in the future we would like to experiment with using multiple Queues at the router node. 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 also improve the Go-Back-N performance.

4 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;
```

2) Packet Generation and sending:

```
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
```

5 Appendix:

- Project Code on the GitHub Repository (Check Go-Back-N Branch for its code)

ameerlouly/NetworkSwitchSimulator_RTOS_Project