Implementing Four Packet Forwarding Approaches

by

Daniel Makarchuk

7858045

14 Dec 2022

Professor: Sara Rouhani

Comp 4300

University of Manitoba

Project Description:

This project simulates four different packet forwarding techniques.

There is a component (sender.py) that sends a set number of packets, one ever 0.05 seconds. These "packets are generated with a random priority and store the start and later end time of their journey." The number of sent packages at once can be changed by changing the constant PACKETS_NUM. the default is 500 to ensure the runs have a decent amount of packets sent without taking too long.

There is a router component (router.py) that forwards packets on to the receiver using the technique chosen. The router also records the number of packets dropped and at the end of processing displays those figures in the terminal. A packet is forwarded every 0.1 second to have a forwarding speed half of the sending rate ensuring a decent proportion of packets sent are lost. The storage space is set to hold at most 20 packets to ensure a flexible number that can be partitioned if needed.

The techniques are:

- **FIFO** (first in first out): The que stores the packets in the order they arrived and drops any packet received when the que is full. The packet sent is the first packet in the que.

- **Priority Que** (PQ): Where the preference of priority is Ultra, High, Mid and finally Low. When a more preferred packet arrives, it is inserted behind all other packets of its priority (fifo for each priority level). The first packet in the que is forwarded first.

- **Round Robin** (RR): Each packet priority has its own que where the combined size of the ques can not exceed the size limit (20). The packets are forwarded in a rotation dependent on priority class, with first a Low, then a Mid, then a High, and finally an Ultra are sent. The pattern repeats as many times as needed and skips a class if there are none of it to send.

- **Reserved Round Robin** (ReRR): Inspired by weighted fair queuing to ensure there is reserved space for each priority. In this approach the Ultra priority has more reserved space (8) compared to the lower priorities (High = 5, Mid = 4, Low = 3). Then the sending packet is chosen in the same way as in the Round Robin approach. The inserts are done in a priority que manner with a priority being inserted into a full list so long as it has reserved space. This means that in some cases it is possible for a Low packet to displace a Mid, High, or even Ultra packet if the Low reserved space is not full. If there is room in the data structure the packet is just added to the correct que as overflow that is not safe from being dropped if another packet forces it out.

Finally, a receiving component (reciever.py) receives the forwarded packet and stores it. After all are received certain statistics are calculated and displayed in the terminal.

The sender sends the receiver it's current time to be used by the receiver to calculate a time difference. This though not a perfect solution due to the possibility of network delay, should hopefully reduce the impact of time differences in the statistical analysis.

Packets are randomly derived to make each test run unique, with the chance for outliers to appear and see how the approach is affected by the outliers. This helps ensure each run will have different statistics.

Time delays are used to ensure the send rate is faster than the delay rate and to ensure that each packet has a calculatable time gap close to the seconds unit.

Next in the report are the components to the code, the git hub link and running instructions. Further on in this report the data from three runs with each method are displayed and discussed.

Git page:

https://github.com/danielmakarchuk/4300Project.git

Components:

sender.py

reciever.py

router.py

fifo.py

roundRobin.py

reservedRR.py

priorityQue.py

Run instructions

To run on the command line (3 terminals needed):

First: have all components in the same file on the same host (all use localhost for the host).

Second: start the reciever with python 3: python(3) reciever.py

Third: start the router with a forwarding approach on the same host in the second terminal:

python(3) router.py rerr

rerr: for the reserved round robin approach

re: for the round robin approach

pq: for the priority que approach

fifo: for the first in first out approach

Any other (or the lack of) input will break the code and you will need to restart the running process.

Fourth: start the sender on the same host as the other 2 in the third terminal with:

python(3) sender.py

If you want to change the number of packets send change the PACKETS_NUM constant the default is 500.

Results:

Fifo:

Run 1:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	221	113	108	2.2234464	2.4463639
Mid	124	72	52	2.2959916	2.4449295
High	111	62	49	2.2200579	2.4466567
Ultra	44	22	22	2.3372594	2.4430985
total	500	269	231	2.2691888	N.A.

Run 2:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	224	130	94	2.2202391	2.4514749
Mid	108	49	59	2.2464892	2.4506895
High	116	66	50	2.2607824	2.4491785
Ultra	52	24	28	2.4041843	2.4443640
total	500	269	231	2.2829238	N.A.

Run 3:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	215	123	92	2.2593999	2.4478759
Mid	119	59	60	2.2599143	2.4384810
High	116	57	59	2.2112315	2.4443635
Ultra	50	30	20	2.2512405	2.4450483
total	500	269	231	2.2454466	N.A.

Averages:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)	success rate
Low	660	366	294	2.2343618	2.4514749	55.5%
Mid	351	180	171	2.2674650	2.4506895	51.3%
High	343	185	158	2.2306906	2.4491785	53.9%
Ultra	146	76	70	2.3308947	2.4450483	52.1%
total	1500	807	693	2.2658531	N.A.	N.A.

Priority Que:

Run 1:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	234	4	230	2.9343773	4.4410903
Mid	127	126	1	2.3962487	5.8161473
High	99	99	0	0.0763958	0.3093776
Ultra	40	40	0	0.0324417	0.0636718
total	500	269	231	1.3598659	N.A.

Run 2:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	234	3	231	22.7024868	33.0257949
Mid	133	133	0	2.2874111	4.2695469
High	94	94	0	0.0770288	0.4381201
Ultra	39	39	0	0.0440912	0.1230170
total	500	269	231	6.2777545	N.A.

Run 3:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	218	5	213	1.7764419	2.7793402
Mid	139	121	18	3.5621854	6.7757117
High	93	93	0	0.0856757	0.4409952
Ultra	50	50	0	0.0375758	0.1271789
total	500	269	231	1.3654697	N.A.

Averages:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)	success rate
Low	686	12	674	9.1377686	33.0257949	1.75%
Mid	399	380	19	2.7486150	6.7757117	95.2%
High	286	268	0	0.0797001	0.4409952	100%
Ultra	129	129	0	0.0380362	0. 1271789	100%
total	1500	807	693	3.0010300	N.A.	N.A.

Round Robin:

Run 1:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	234	123	111	4.3513535	6.5768773
Mid	104	62	42	0.6129864	2.6848192
High	96	52	44	0.4969291	2.0711169
Ultra	66	32	34	0.2655730	1.1870608
total	500	269	231	1.4317105	N.A.

Run 2:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	203	112	91	4.5085846	6.3159544

Mid	130	67	63	0.3966677	1.4330279
High	117	63	54	1.1302291	3.1948664
Ultra	50	27	23	0.1784524	0.5625004
total	500	269	231	1.5534834	N.A.

Run 3:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	232	124	108	4.4080159	6.9332575
Mid	127	66	61	0.6289043	2.3246684
High	87	47	40	0.2331737	0.9435780
Ultra	54	32	22	0.2794082	1.0631659
total	500	269	231	1.3873755	N.A.

Averages:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)	success rate
Low	669	359	310	4.4226513	6.9332575	53.7%
Mid	361	195	166	0.5461861	2.6848192	54%
High	300	162	138	0.6201106	3.1948664	54%
Ultra	170	91	79	0.2411445	1.1870608	53.5%
total	1500	807	693	1.4575231	N.A.	N.A.

Reserved Round Robin:

Run 1:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	225	74	151	3.8881639	6.9529180
Mid	118	71	47	1.6369571	2.9534008
High	100	67	33	1.8313880	3.4488515

Ultra	57	57	0	1.0466066	2.7020459
total	500	269	231	2.1007789	N.A.

Run 2:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	234	77	157	3.9652197	6.3356685
Mid	108	69	39	1.6642459	3.5172257
High	105	70	35	1.9294873	3.0113542
Ultra	53	53	0	0.6026473	1.4331223
total	500	269	231	2.0404000	N.A.

Run 3:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)
Low	199	73	126	2.7399083	4.7549014
Mid	113	66	47	1.8977610	2.9405045
High	122	68	54	2.3551778	4.3297917
Ultra	66	62	4	1.7012748	4.0765483
total	500	269	231	2.1735304	N.A.

Averages:

Priority	Sent	Received	Dropped	Average time (Seconds)	longest (seconds)	success rate
Low	658	224	434	3.5310973	6.9529180	34%
Mid	339	206	133	1.7329880	3.5172257	60.8%
High	327	205	122	2.0386844	4.3297917	62.7%
Ultra	176	172	4	1.1168429	4.0765483	97.7%
total	1500	807	693	2.1049031	N.A.	N.A

Observations and Ideas

There are many things the generated data indicates about each approach. Most of which supports the material discussed in class.

First there is fairness to consider. Both the FIFO and RR approach had pretty even success rates across the priorities. The PQ was the least fair with almost all packets of Low priority being dropped (only 12 of 686 reaching the reciever). The ReRR approach was also unfair, sending almost all Ultra packets, a majority of Mid and High packets and a minority of Low packets. Over all the FIFO and RR packets with this testing have been the fairest in terms of receival rate and should be used if you want an even receival rate across priorities.

If Priority matters then the PQ approach is the best. This is because all High and Mid priority packets reached their destinations. Though this came at the cost of Low packets and Mid packets, though most Mid did arrive. Low packets however suffered a 1.75% success rate. ReRR is second best with almost all Ultra packets being received, and more High packers received then FIFO and RR approaches. This approach could likely have the high success rate which is almost even with the mid at approximately 60% success by changing the reserved values, say taking one from mid to be added to the High reserved space. RR and FIFO were terrible for the Ultra, dropping over 40%. Thus, if you need all Ultra to arrive and don't care about Low packets use the PQ approach, and if you are ok with slight losses of Ultra and want a reasonable (over 10%) Low success rate then use the ReRR and tune it to the implementation.

Each method dropped and forwarded the same number of packets. Thus, there is no apparent throughput advantage to any of the methods here in this simulation, likely due to the imposed dime delays between forwarded packets. If this time delay was reduced or delays added to the more complex actions of various methods, such as PQ or ReRR insertion, it is possible that the throughput of the more complex approaches will dip slightly.

The RR method greatly improved the average send time of Mid, High, and Ultra packets while increasing the Low send time in comparison to the more even average times of FIFO. This is likely due to the high quantity of Low priorities in the low que needing to wait longer to be sent then the other priorities in their respective send ques. These higher priorities were likely sent faster in the RR then FIFO because in fifo each priority, once the que was full, had to wait 19 rounds before being sent, with Fifo they only had to weight their respective priority class ques size-1 rounds.

A note about the PQ Low packets. It is likely that the majority of low packets sent were either sent at the beginning when no higher priority was yet blocking the Low's sending, or at the end, added to the Que and not bumped off since no higher priority were arriving. There is also the 33 second outlier that greatly changes the average send time of Low packets. Likely caused by a low that was kept from being sent for a long time by a higher priority constantly pushing it down in the list.

Video:

I uploaded a short video showing myself running the code with the PDF to UM learn to display how I run the project.