ABSTRACT

Buffer bloat is a phenomenon in which excess buffering of packets would cause high latency and packet delay variation in packet-switched networks. To prevent buffer bloat, network scheduler comes into play where the sequence of network packets transmit and receive queues from the network interface controller. It uses well-known queueing methods - first come first served (FCFS) and priority scheduling to prevent the packetsize from exceeding the buffer capacity.

If the buffer capacity exceeds the packet size; no buffer bloat would occur which would result in low latency. It therefore creates a huge breakdown on network performance. The one-sentence summary is “Bloated buffers lead to network-crippling latency spikes”. Buffer bloat affects all sorts of devices and networks, even the Internet. Only recently have system engineers come across solutions to what we have described above – the buffer bloat problem. Our aim is to prevent buffer- bloat in the network scheduler, for the above mentioned reasons. We shall then use Round Robin methods as an alternative to prevent the buffer-bloat in the network scheduler (in a packet switched network)

INTRODUCTION AND APPLICATION

A network scheduler, also called packet scheduler, queueing discipline, qdisc or queueing algorithm, is an arbiter on a node in packet switching communication network. It manages the flow of network packets which transmits and then receives queues from the network interface controller.

The network scheduler mechanism helps us to decide which network packet next to forward to. The network scheduler associates with the queuing system thereby storing network packets for sufficient time until they are transmitted. Systems can have a single as well as multiple queues as per the need to hold the packets of one flow, classification, or priority.

Although in some cases, it is impossible to schedule all transmissions within the restrictions of the system. In those cases, the network scheduler is used to decide which traffic next to forward to and what needs to getsdropped.

In a datagram network, each switch has a specific routing table which is updated occasionally. The destination addresses and the corresponding forwarding ports are stored in the routing tables. Each packet in a datagram network carries a header. This header contains data as well as destination address.

When the switch receives a packet, it checks the destination address from which the routing table finds the next available port by which the packet is being forwarded.

Store-and-Forward: The packets are passed from node to node and the nodes may be stored in the data momentarily before it passes to the next node.

Packets are independent entities: In a datagram network, each packet is treated in a different way from other packets. Even though a packet is part of a multi-packet transmission, the network would treat it in such a way that it existed alone. This way of handling the packets are referred to as datagrams.

Different Routes: The packets of a single message may follow different routes before they reach their final destination. This can happen since a link is already involved in carrying packets from different sources and there is not much bandwidth to carry packets which needs to reach the final destination.

Now, coming to buffer bloat and what it exactly is. Buffer-bloat is the unwanted latency which comes from a router or other network devices buffering enormous data. It is a huge breakdown on network performance since it cause packet delay variation and time delay. The one-sentence summary is “Bloated buffer lead to network-crippling latency spikes.”

To fix bufferbloat you must first understand it. A simple way to visualise the problem is via the traffic analogy, an old introduction to the problem that is pretty close to what actually goes on in a network.

Bufferbloat is a phenomenon in packet-switched networks in which excess buffering of packets causes high latency and packet delay variation. Bufferbloat can be addressed by a network scheduler that strategically discards packets to avoid an unnecessarily high buffering backlog.
Examples include CoDel and Random early detection.

LITERATURE STUDY/RELATED WORK

The buffer-bloat is a term presented by Jim Gettys which is exceptionally well known now and means the presence of very large and frequently full buffers inside the network. The buffer-bloat impacts the latency, a vital parameter in the network systems, in particular. The latency comprises of three sorts of delays: processing delay, transmission delay and queuing delay. The queuing delay is the time during which the network packets spend holding up to be transmitted or processed. This time clearly relies upon the size of the queue which can become immense on account of the buffer- bloat.

The reason of this development is a congestion in network path which can experience the ill effects of paths between endpoints that are regularly made up of numerous hops with connections of various bandwidth. In the quick to- moderate progress hops we can have the circumstance when there are packets arriving to be queued or dropped, on the grounds that they cannot be instantly transmitted due to the past entries being not processed yet. The buffers are essential in the "burst send" situations to keep up their flow of packets at the most extreme rate and accomplish the throughput. In whichever case, inaccurate bandwidth estimation would prompt such a large number of packets being sent, that outcomes in buffers overpopulation or packet drops. On the off chance that the packets are too enormous, the packets are not dropped, and the queue grows, and in addition the queuing delay. For this situation we have pointless latency growth which does not prompt any advantages in throughput. In this way, it is anything but difficult to reason that within the sight of congestion in the net with unreasonable buffering the subsequent delay can develop high. Most network hops experience the ill effects of the unreasonable buffering. The other issue is that the network congestion is a typical case.

Buffer-bloat is a serious problem being noticed on the Internet, and it has become frustrating for users and system engineers alike. Jim Gettys' home network was so slow and frustrating he set out to determine what the problem was, and as mentioned above, he went on to coin the term 'buffer-bloat'.

Here are some excerpts from A Discussion with Vint Cerf, Van Jacobson, Nick Weaver, and Jim Gettys:

"...all this excessive buffering ends up breaking many of the timeout mechanisms built into our network protocols. That gets us to the question of just how bad the problem really is and how much worse it's likely to get as the speeds out at the edge of the net continue to increase." [1]– Vint Cerf

The buffer-bloat problem is likely to worsen as the demand grows for Internet-intensive activities such as streaming video and remote data backup and storage, with the online user experience bound to deteriorate as a consequence. The problem is most visible at the edge of the network, but evidence of it can also be found in the Internet core, in corporate networks, and in the boundaries between ISPs. Besides calling for simple tools that people can use to find and measure buffer-bloat, active queue management for all devices is also in order. The best algorithm for the job remains to be determined, however.

Recent developments in the past decade have seen the development of one algorithm that is seen as a solution for buffer bloat – CoDel (pronounced "coddle"). It is a simple and novel scheduling algorithm for the network scheduler developed by Van Jacobson and Kathleen Nichols. It is designed to solve the issues caused by buffer bloat in network link(for e.g. routers), by setting constraints on the delay of network packets that is experienced as they pass

through the buffer. CoDel (the name comes from “controlled delay”) is the first major advancement in the state of the art of network Active Queue Management which took years of research.

The theory behind CoDel is based on observations of packet behaviour in packet-switched networks under the influence of data buffers. Some of these observations are about the fundamental nature of queuing and the causes of buffer-bloat.

What CoDel essentially does is distinguish between two types of queues: good queues (which exhibit no buffer-bloat) and bad queues (which do). Van Jacobson, in his 2006 talk presented at MITLincoln Labs, asserted that existing algorithms at the time such as RED (random early detection), used incorrect means of recognising buffer-bloat.
CoDel was developed based on Jacobson’s notion.

A full implementation of CoDel was realised in 2012 and was made available as Open Source software. Dave Täht and Eric Dumazet also created an implementation of the algorithm for the Linux kernel, known as fq_codel. fq_codel took 4 years of research, development and deployment, and is an attempt at making buffer-bloat relatively easy to fix. What’s more, fixing it may solve a lot of the service problems now addressed by bandwidth caps and metering, making the Internet faster and less expensive for both users and providers.

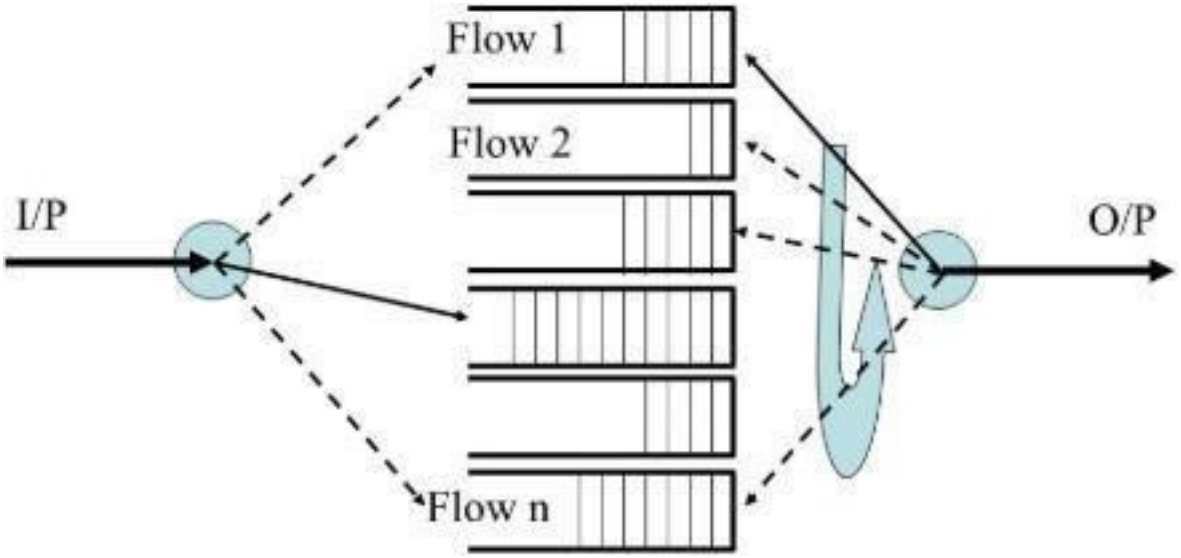
Round robin algorithm for network packet scheduling-

Round-robin scheduling can be used as an alternative to first-come first-served queuing for having quality packet switching and other communication sharing of link.

A multiplexer, switch, or router that uses round-robin scheduling for data flow has a separate queue which may be recognized by its source as well as destination address. The algorithm does its part to let every active data flow that has sufficient data packets wait in the queue to take turns in transferring packets on a shared physical channel in an order repeated periodically. This scheduling saves congestion by allowing the flow of data time to time. Additionally, the scheduling tries to prevent resources that can be linked to go unused.

Round-robin scheduling results in max-min fairness if the data packets are of equal size, since the data flow that has waited for a longer period of time is given a higher priority in scheduling.

Figure 1: Packets in queue



Logic diagram

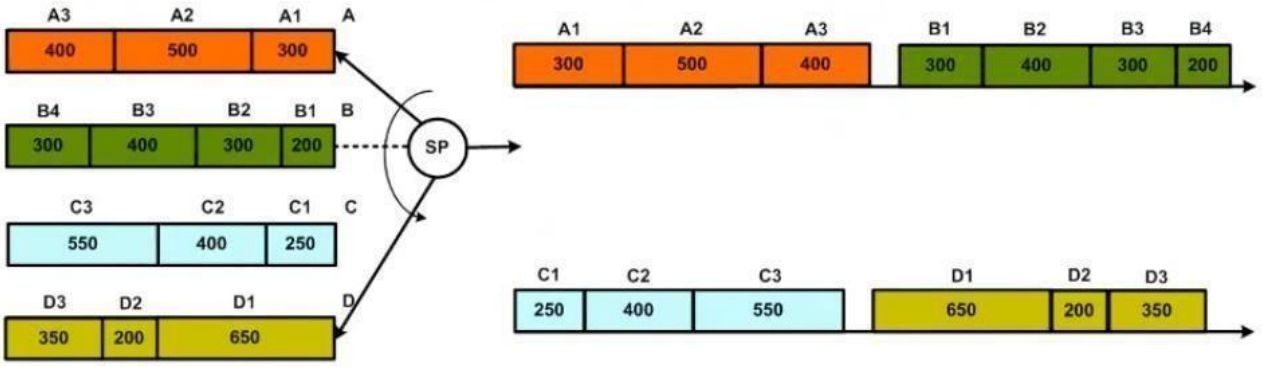


Figure 2: Strict priority scheduler with FCFS-

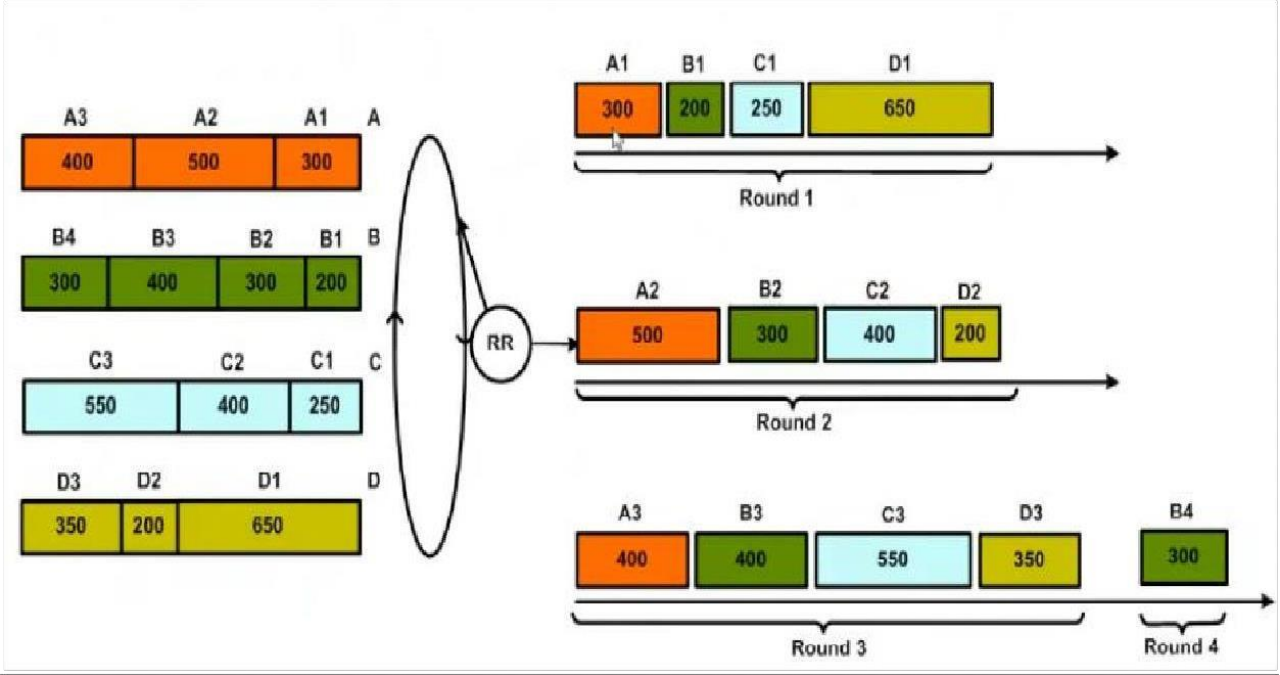


Figure 3: Round Robin Scheduler (Proposed algorithm)

INPUT TABLE

Enter no. of packets queue: 4,5,3,4

Enter no. of bits in each packet:

4	5	3	4
50	30	250	100
100	50	700	40
200	100	200	100
100	200		50

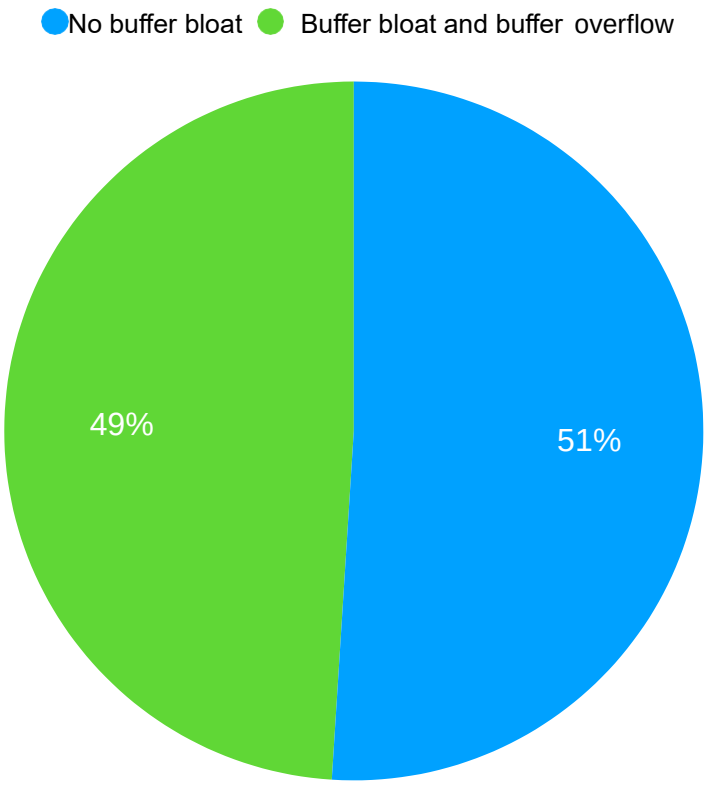
Output table

	FCFS with priority	Round Robin
Buffer bloat on buffer capacity	1	0
No Buffer bloat (success case)	3	4
Efficency	75%	100%

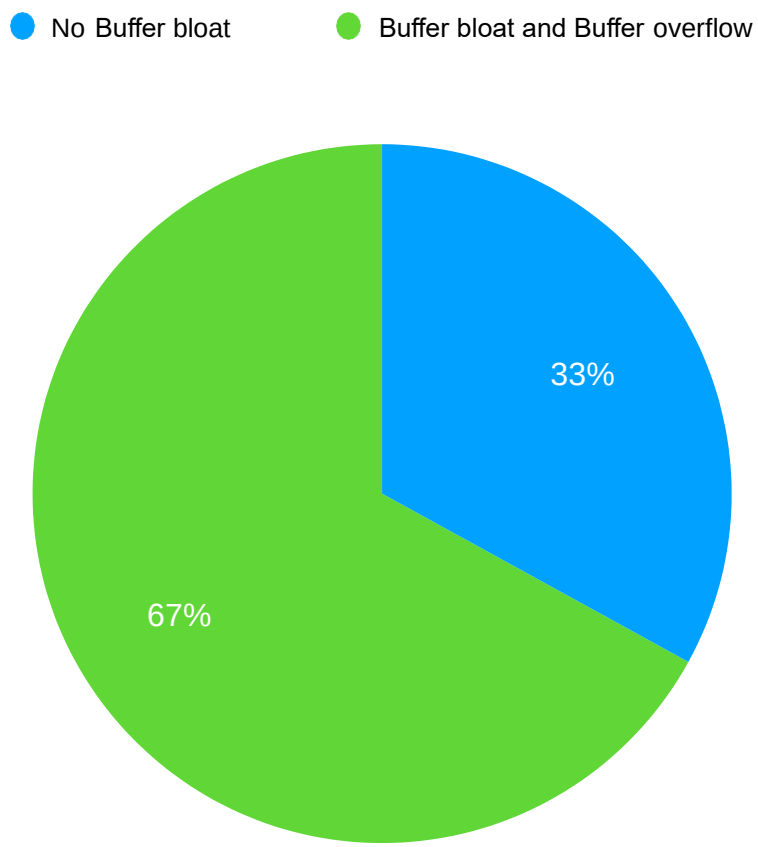
EVALUATION/DISCUSSION

The network scheduler program was implemented in C both with priority scheduling (with FCFS) and Round robin.
The program was run on following parameters Buffer
Capacity – 800
Buffer bloat – When buffer reaches >=700 (High Latency) Buffer capacity overloaded – When buffer reaches >800 And no buffer bloat in remaining cases.
Success case – No Buffer bloat The program was tested for many sample inputs and the following observations were made-

FCFS with Priority



Round Robin

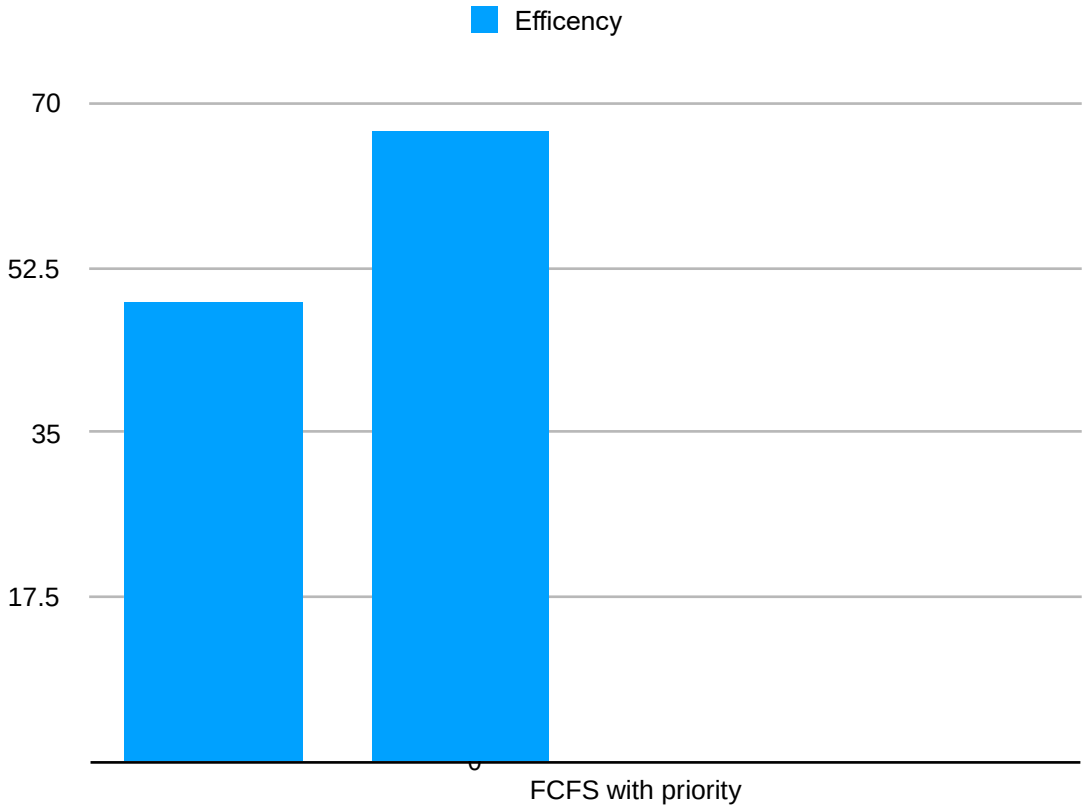


Efficiency -

Efficiency was calculated as-

1. For FFCS with priority scheduling = (No of success cases/ No of data packet queues)*100%

2. For Round Robin =(No of success cases/ No. Of Rounds)*100%



CONCLUSION

The no of times buffer bloat occurred or buffer capacity overloaded was less in case of round robin than priority scheduling (with FCFS) and hence the efficiency in case of round robin was 16% more than in priority scheduling. This can also be implemented in multiple-access networks where round-robin scheduling may be provided by token passing channel access schemes (for e.g. token ring)..Also round robin can be implemented In centralized wireless packet radio network, where many stations share a single frequency channel. It may reserve time slots for the mobile stations in a round-robin fashion and provide fairness. Round Robin may not be desirable if the size of the data packets varies widely from one job to another. A user that can produce large packets would be considered over other users. Nevertheless, the algorithm can be extended to fair queuing using weighted round robin and deficit round robin which may be even more efficient. Hence Round Robin is a better alternative than priority scheduling (with FCFS) in network schedulers to prevent buffer bloat.

REFERENCES

1. Getty, Jim. Packets on the Highway. n.d. Web page. 01 11 2018.
2. Jiang, H., Wang, Y., Lee, K., & Rhee, I. (2012). Tackling bufferbloat: https://sci-hub.tw/ 10.1145/2398776.2398810
3. Bufferbloat: Dark Buffers in the Internet. IEEE Internet Computing, IEEE, 1990. Jacobson, Van & Floyd, Sally. Random Early Detection Gateways for Congestion Avoidance.IEEE/ACM Transactions on Networking, 1993
4. A Discussion with Vint Cerf, Van Jacobson, Nick Weaver, and Jim Gettys – https:// www2.cs.duke.edu/courses/common/compsci092/papers/bufferbloat.pdf
5. Controlling Queue Delay ACM Queue, Kathleen Nichols, VanJacobson, May, 2012: http:// queue.acm.org/detail.cfm?id=2209336
6. Iljitsch van Beijnum (2012-05-10). "CoDel buffer management could solve the Internet's bufferbloat jams". Ars Technica. Retrieved 2012-08-16.
7. Wikipedia contributors. "Network scheduler." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 12 Oct. 2018. Web. 1 Nov. 2018.

CODE:

```
#include<stdio.h>
int checkbuffer(int n,int max,int z)
{
int count1=0; int f = max-100; if(n<f)
{
printf("No bufferbloat,low latency\n");
}
else if((n>=max-100)&&(n<=max))
{
printf("Bufferbloat,high latency\n"); count1++;
}
else if(n>max)
{
printf("Buffer capacity overloaded\n"); count1++;
}
return count1;
}
int main()
{
printf("FCFS with priority scheduling-\n\n");
inta[20]={0},b[20]={0},c[20]={0},d[20]={0},la,lb,lc,ld,max,select,h,ha,hb,hc,hd,nr;
int s[20],temp,buffer[20],g[20]; int pa=0,pb=0,pc=0,pd=0,n=0;
int p=0,q=0,r=0,t=0,asum=0,bsum=0,csum=0,dsum=0,i=0,j=0,k=0; printf("Enter no of
packets for queue 1\n"); scanf("%d",&la);
printf("Enter number of bits for each packet\n"); for(i=0;i<la;i++)
{
scanf("%d",&a[i]); asum = asum + a[i];
}
s[0]=asum;
printf("Enter no of packets for queue 2\n");
scanf("%d",&lb);
printf("Enter number of bits for each packet\n");
for(i=0;i<lb;i++)
{
scanf("%d",&b[i]); bsum = bsum + b[i];
}
s[1]=bsum;
printf("Enter no of packets for queue 3\n"); scanf("%d",&lc);
printf("Enter number of bits for each packet\n"); for(i=0;i<lc;i++)
{
```

```
scanf("%d",&c[i]); csum = csum + c[i];
}
s[2]=csum;
printf("Enter no of packets for queue 4\n");

scanf("%d",&ld);
printf("Enter number of bits for each packet\n”);

for(i=0;i<ld;i++)
{

scanf("%d",&d[i]); dsum = dsum + d[i];
}
s[3]=dsum;
printf("Enter the maximum capacity of buffer\n");

scanf("%d",&max); for(i=0;i<3;i++)
{

for(j=i+1;j<4;j++)

{

if(s[i]>=s[j])

{

temp=s[i];

s[i]=s[j]; s[j]=temp;
}

}

}
i=0; for(i=0;i<4;i++)

{

if(s[i]==asum)

{

pa = i+1; g[0]=pa;
}

} i=0;
```

```
for(i=0;i<4;i++)

{

if(s[i]==bsum)

{

pb = i+1; g[1]=pb;
}

}

printf("priority of queue a = %d\n priority of queue b =%d\n priority of queue c=%d\n
priority of queue d=%d\n",g[0],g[1],g[2],g[3]);

for(k=0;k<4;k++)

{
printf("Enter priority number\n"); scanf("%d",&select); if(select==g[0])
{

for(p=0;p<la;p++)

{

buffer[p] = a[p];

}

for(i=0;i<la;i++)

{

printf("|%d|\t",buffer[i]);

ha = checkbuffer(asum,max,la);

}



}


else if(select==g[1]) { for(q=0;q<lb;q++) {
buffer[q] = b[q]; } for(i=0;i<lb;i++) { printf("|%d|\t",buffer[i]); hb
=checkbuffer(bsum,max,lb);
}
```

```
else if(select==g[2]) { for(r=0;r<lc;r++) {
buffer[r] = c[r];

}

for(i=0;i<lc;i++) { printf("|%d|\t",buffer[i]); hc =checkbuffer(csum,max,lc);
}


else if(select==g[3]) { for(t=0;t<ld;t++) {
buffer[t] = d[t];

}

for(i=0;i<ld;i++)

{

printf("|%d|\t",buffer[i]);

hd = checkbuffer(dsum,max,ld);

} else {

}

}

}

}

printf("Invalid input");

}
h = ha+hb+hc+hd;
printf("\n number of times buffer bloat occurred or buffer capacity overloaded%d\n",h);

printf("\n number of times no buffer bloat occured(success cases%d\n",4-h);

int go;
printf("\n Press 1 to run scheduler for Round Robin\n");

SCANF("%D",&GO); if(go==1)

{

int ex = 4-h; i=0;j=0,k=1,t=0,nr=0;
```

```
int rsum=0,h2=0,h3=0;

printf("\nRound Robin PACKET scheduling\n"); while(nr==0)
{

j=0;
printf("\n Round %d\t",k); BUFFER[J]=A[I]; buffer[j+1]=b[i]; buffer[j+2]=c[i];
buffer[j+3]=d[i];
for(t=0;t<=3;t++)
{

rsum = rsum + buffer[t];

}

for(t=0;t<4;t++)

{

printf("|%d|\t",buffer[t]);

}
h2 = CHECKBUFFER(RSUM,MAX,LA);

if(h2==1)
{

h3++;

}

i++; k++;

IF(A[I]==0&&B[I]==0&&C[I]==0&&D[I]==0)

{

nr = 1;

}

rsum=0; h2=0;
}
FLOAT ef,er;
printf("\nNumber of times BUFFERBLOAT occured or buffer CAPACITY
OVERLOADED
%d\n",h3);
```

```
printf("\nNumber of times no BUFFERBLOAT occured (success CASES) %d\n",k-1-h3);
int ey = k-1-h3;

ef = (FLOAT(EX)/4)*100;
er = (FLOAT(EY)/(K-1))*100;

printf("\nEfficiency of FCFS with priority scheduling is %f percent \n",ef);
printf("\nEfficiency of Round Robin is %f percent \n",er);
if(ef>er)
{

printf("\nFCFS with priority scheduling is %f percent more efficient\n",ef-
er);

}
else if(er>ef)

{

printf("\nRound Robin is %f percent more efficient\n",er-ef);

}
else if(er==ef)

{

printf("\Efficiency is SAME in this CASE for both ALGORITHMS");

}

}
```

OUTPUT

```
Enter no of packets for queue 1
4
Enter number of bits for each packet
50
100
200
100
Enter no of packets for queue 2
5
Enter number of bits for each packet
30
50
100
100
200
Enter no of packets for queue 3
3
Enter number of bits for each packet
250
400
200
Enter no of packets for queue 4
4
Enter number of bits for each packet
100
40
100
50
Enter the maximum capacity of buffer
800
priority of queue a = 2
 priority of queue b =3
 priority of queue c=4
 priority of queue d=1
Enter priority number
1
|100|   |40|    |100|   |50|    No bufferbloat,low latency
Enter priority number
2
|50|    |100|   |200|   |100|   No bufferbloat,low latency
Enter priority number
3
|30|    |50|    |100|   |100|   |200|   No bufferbloat,low latency
Enter priority number
4
|250|   |400|   |200|   Buffer capacity overloaded

Number of times bufferbloat occured or buffer capacity overloaded 1

Number of times no bufferbloat occured (success cases) 3



 Press 1 to run scheduler for  Round Robin
1

Round Robin packet scheduling

 Round 1        |50|    |30|    |250|   |100|   No bufferbloat,low latency

 Round 2        |100|   |50|    |400|   |40|    No bufferbloat,low latency

 Round 3        |200|   |100|   |200|   |100|   No bufferbloat,low latency

 Round 4        |100|   |100|   |0|     |50|    No bufferbloat,low latency

 Round 5        |0|     |200|   |0|     |0|     No bufferbloat,low latency

Number of times bufferbloat occured or buffer capacity overloaded 0

Number of times no bufferbloat occured (success cases) 5

Efficiency of FCFS with priority scheduling is 75.000000 percent

Efficiency of Round Robin is 100.000000 percent

Round Robin is 25.000000 percent more efficient
```