PROJECT 4 – SWITCH PERFORMANCE & HOL BLOCKING

Tamoghna Chattopadhyay – 8541324935 –

1. INTRODUCTION

N×N switch's performance under heavy traffic, i.e., always packets on input side is considered. In particular, the HOL (Head of Line) slot is always full. The packet in the HOL position at any input of the N input ports is addressed to output ports $j : j = 1, \dots, N$ with probability α_j and

$$\sum_{j=1}^{N} \alpha_j = 1$$

Packets can be delivered from inputs to outputs in one clock cycle and clock rate is 10^6 cycles per second. If there is more than 1 packet destined to a specific port, only one of them can be delivered in current slot, the others will remain in HOL position on i/p side. This will reduce switch throughput and is called HOL Blocking.

Building the transition probability matrix and solving the Markov chain numerically to find the limit distribution is easy method in case of 2×2 switch to find overall switch performance. But it becomes more difficult as N increases. So simulation is used in that case.

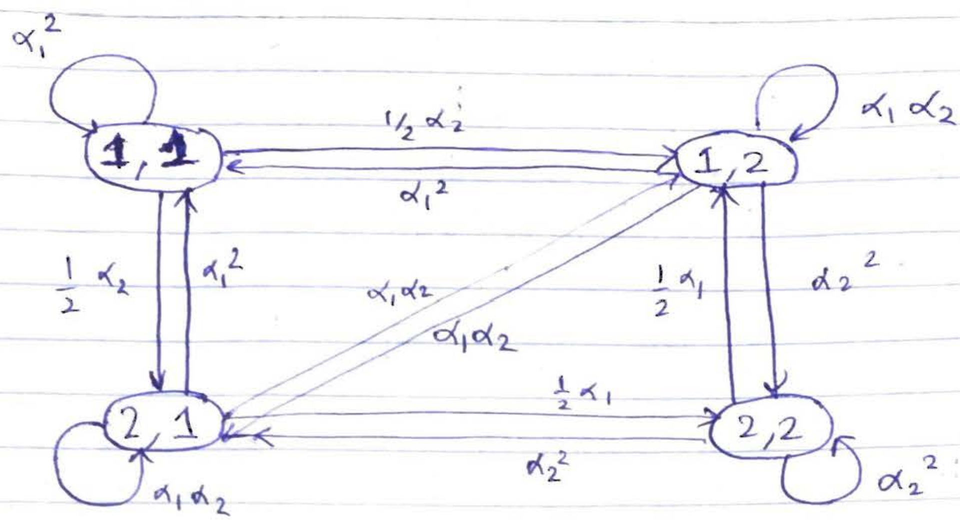
2. THEORY

α_1 : probability of addressed to o/p port 1

α_2 : probability of addressed to o/p. port 2

$\alpha_1 + \alpha_2 = 1$

In case of 2 x 2 switch,



Putting it in a Markov Matrix,

P	1,1	1,2	2,1	2,2
1,1	α_1^2	$\frac{1}{2}\alpha_2$	$\frac{1}{2}\alpha_2$	0
1,2	α_1^2	$\alpha_1\alpha_2$	$\alpha_1\alpha_2$	α_2^2
2,1	α_1^2	$\alpha_1\alpha_2$	$\alpha_1\alpha_2$	α_2^2
2,2	0	$\frac{1}{2}\alpha_1$	$\frac{1}{2}\alpha_1$	α_2^2

$\Pi_{(1,1)} + \Pi_{(1,2)} + \Pi_{(2,1)} + \Pi_{(2,2)} = 1$

$\Pi_{(1,1)} = \alpha_1^2 \left(\Pi_{(1,1)} + \Pi_{(1,2)} + \Pi_{(2,1)} \right)$

$\Pi_{(1,2)} = \frac{1}{2}\alpha_2 \Pi_{(1,1)} + \alpha_1\alpha_2 \left(\Pi_{(1,2)} + \Pi_{(2,1)} \right) + \frac{1}{2}\alpha_1 \Pi_{(2,2)}$

$\Pi_{(2,1)} := \frac{1}{2}\alpha_2 \Pi_{(1,1)} + \alpha_1\alpha_2 \left(\Pi_{(1,2)} + \Pi_{(2,1)} \right) + \frac{1}{2}\alpha_1 \Pi_{(2,2)}$

$$\pi_{(2,2)} = \alpha_2^2 \left(\pi_{(1,2)} + \pi_{(2,1)} + \pi_{(2,2)} \right)$$

So, $\pi_{(1,2)} = \pi_{(2,1)}$

Putting values of α_1 and $\alpha_2 = 1 - \alpha_1$, we can get the individual probabilities

The limiting distribution (as $k \longrightarrow \infty$) is found by solving

$$\pi = \pi P$$

and

$$\sum \pi_i = 1$$

The solution is :

$$\pi_{(1,1)} = \pi_{(1,2)} = \pi_{(2,1)} = \pi_{(2,2)} = \frac{1}{4}$$

In this case $\alpha_1 = \alpha_2 = 0.5$

3. SIMULATION METHODOLOGY

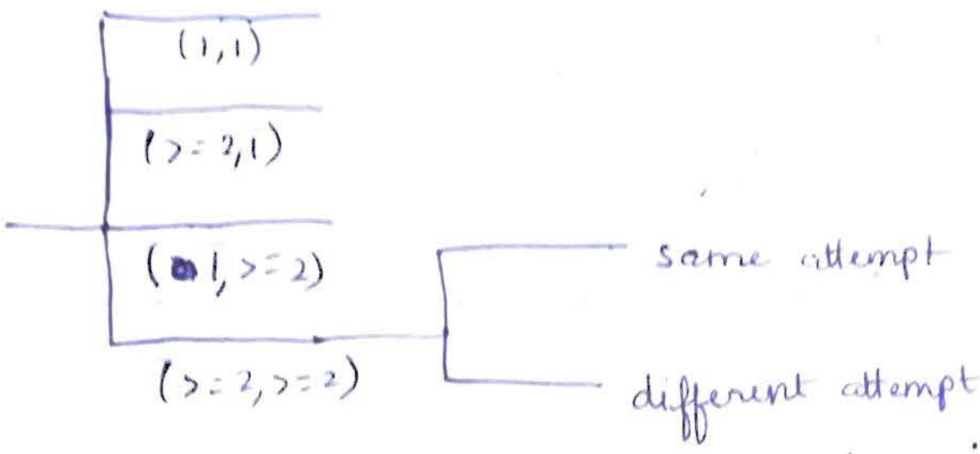
For a 2×2 switch, it is easy to directly simulate the states. We use a variable to record how many packets in each input after some time, at each time slot, there are only four cases of the state :

case 1 : input 1 has packet(s) and input 2 has no packet

case 2 : input 1 has no packet, input 2 has packet(s)

case 3 : both input (1) and input (2) has packet(s).
case 4 : neither has packet

For case 1, the switch just lets packet in input 1 to pass. For case 2, the switch just lets packet in input 2 to pass. For case 3, there are two situations, first, both inputs have the same attempt to go to same output, leading to blocking situation. second: two inputs have different attempt and both of them passed the switch.



We can also plot the Histogram of the number of packets in buffer at input side and the distribution of number of packets processed per slot by the switch.

For generalizing to N×N switch with balanced traffic, i.e., $\alpha_j = \frac{1}{N}$ $\forall j$, it's difficult to model the switch according to the status. It's easier to simulate it based on random numbers. Generate N random numbers and check the uniqueness.

Depending upon the uniqueness, decrease 1 wherever the no '> 0', so that means a packet is passed. Count this number of subtractions as num. The next time generate num random packets and keep on simulating. Adding num to total in each step gives the final total pps by,

$$pps = total / steps.$$

This is easy to simulate for a $N \times N$ switch when there's balanced traffic.

In case of hot - spot traffic,

$$\alpha_1 = \frac{1}{k}$$

and $$\alpha_j = \left(\frac{1}{N-1}\right)\left(\frac{k-1}{k}\right) \quad \text{for } j \neq 1$$

This is much more difficult as each number has a different probability for being generated.

So, for eg. $k = 2$
and its' a 4×4 switch.

Then,
$$\alpha_1 = \frac{1}{2}$$

$$\alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{3} \times \frac{1}{2}$$
$$= \frac{1}{6}$$

So all the other positions on the switch has a probability of being generated one third of the first case.

To simullate this, we can use the randsrc function given in the Communications Toolbox of MATLAB.

5. RESULTS

For 2x2 switch case,

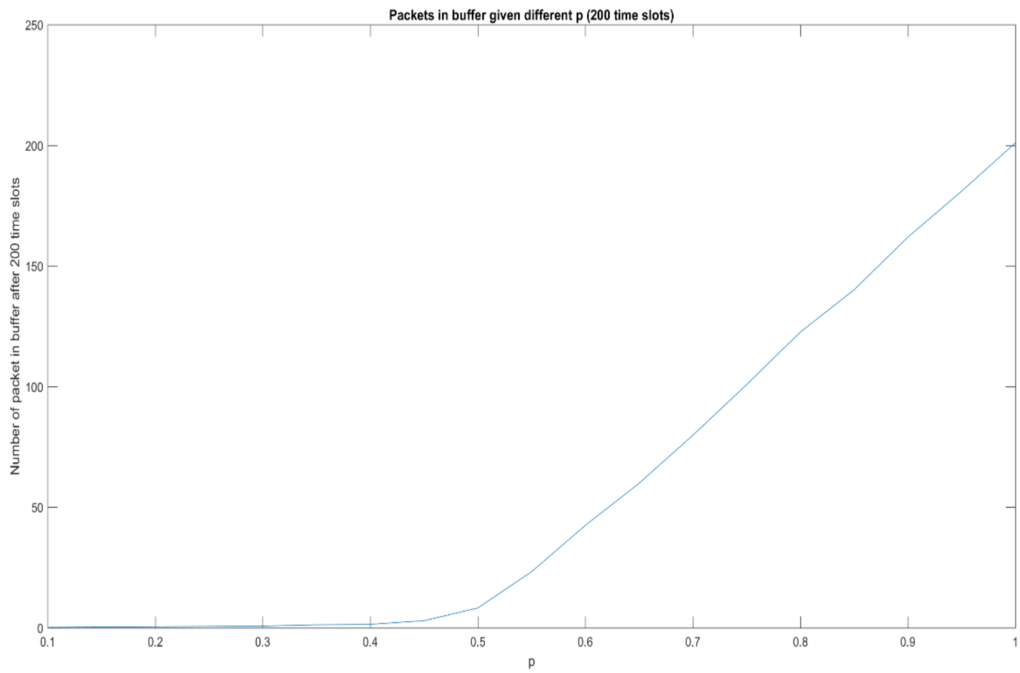
we choose parameters:
tend = 200 ; run 200 time slots
In one given p, we repeat the experiment 100 times to calculate the mean.

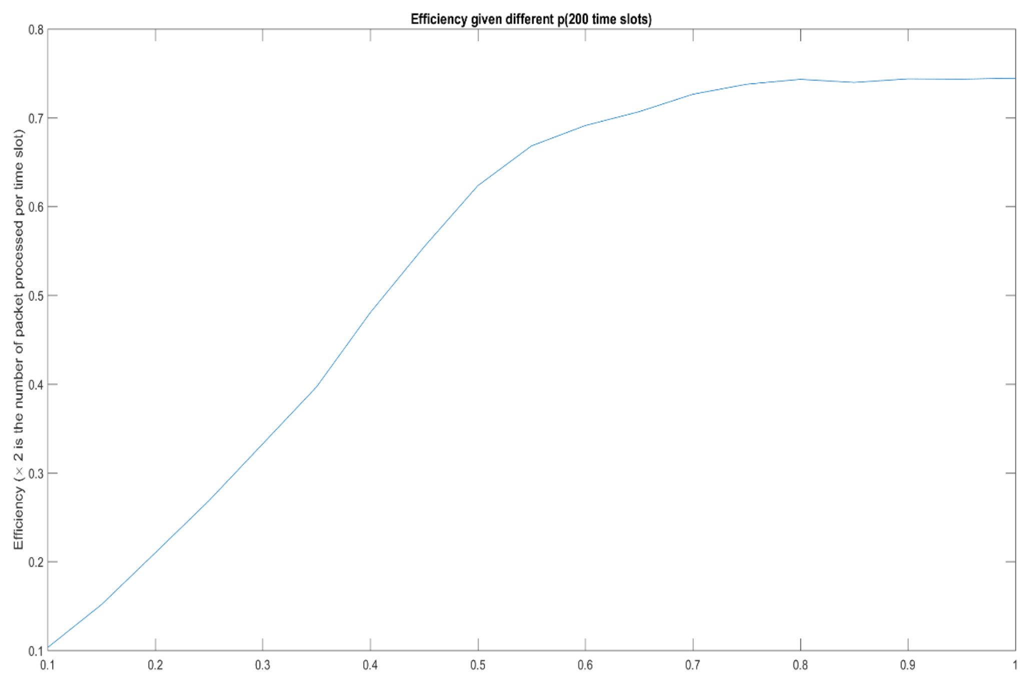
For $r_1 = r_2 = 0.5$, when we choose p less than 0.7, we may not have HOL blocking problem.

As we increase p, it leads to increase in both Mean of N_pkts in buffer and Mean of N_pkts processed per slot. This is because the probability that a packet will arrive to input is increasing then the traffic becomes heavier and this leads to more conflict but higher throughput.

- For 4x4 switch, in case of balanced traffic, I got pps = 2.6060
- For 8x8 switch, in case of balanced traffic, I got pps = 4.9820

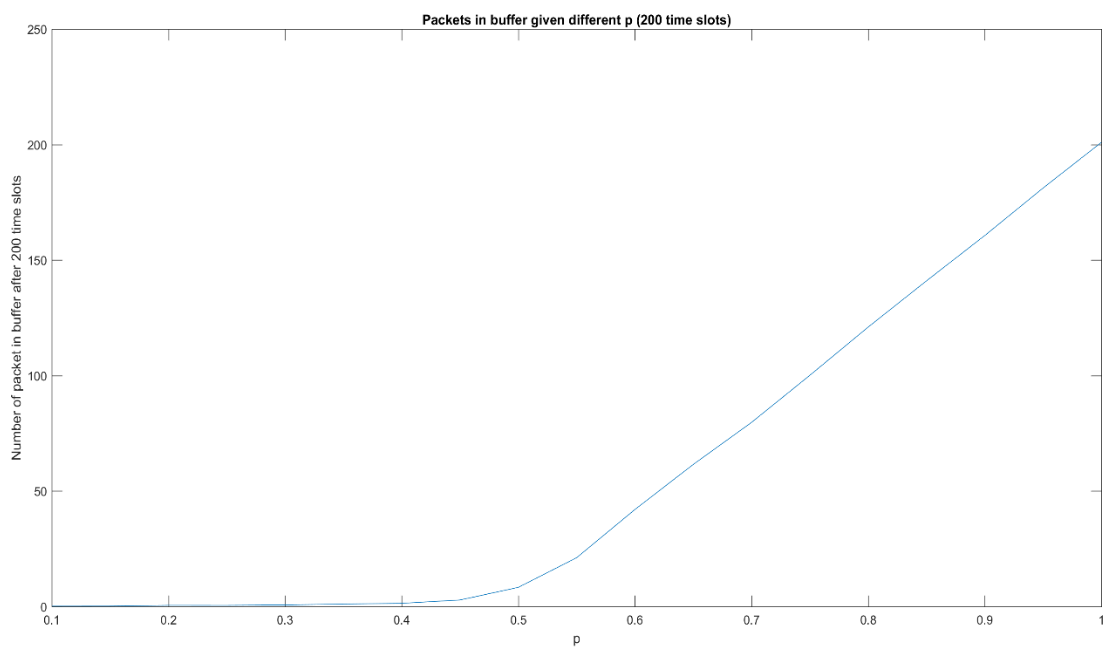
For a 2x2 switch,





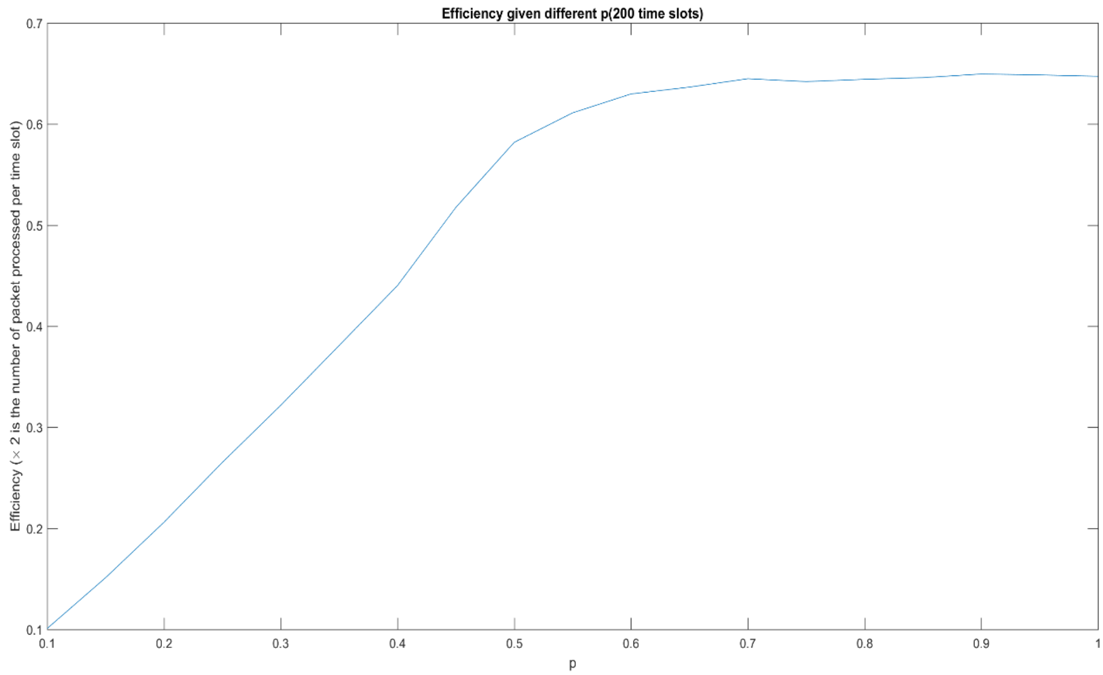
Therefore, when p > 0.7, the efficiency seems to be steady to be around 0.75. We can conclude that if we choose the p to be 0.7, the packet may not pile up in the buffer and still the switch has a good efficiency. To compute the 95% confidence interval we choose p from 0.1 to 0.9,

p	CI efficiency (%)
0.1	[6.25,12.5]
0.2	[15.50,23.25]
0.3	[25.50,33.75]
0.4	[35.50,45.50]
0.5	[44.50,54.25]
0.6	[54.25,64.25]
0.7	[65.75,72.50]
0.8	[70.0,77.0]
0.9	[71.0,78.0]



We just change the parameter r1,r2 in the function and we can get the switch's behaviour when r1 = 0.75 and r2 = 0.25.

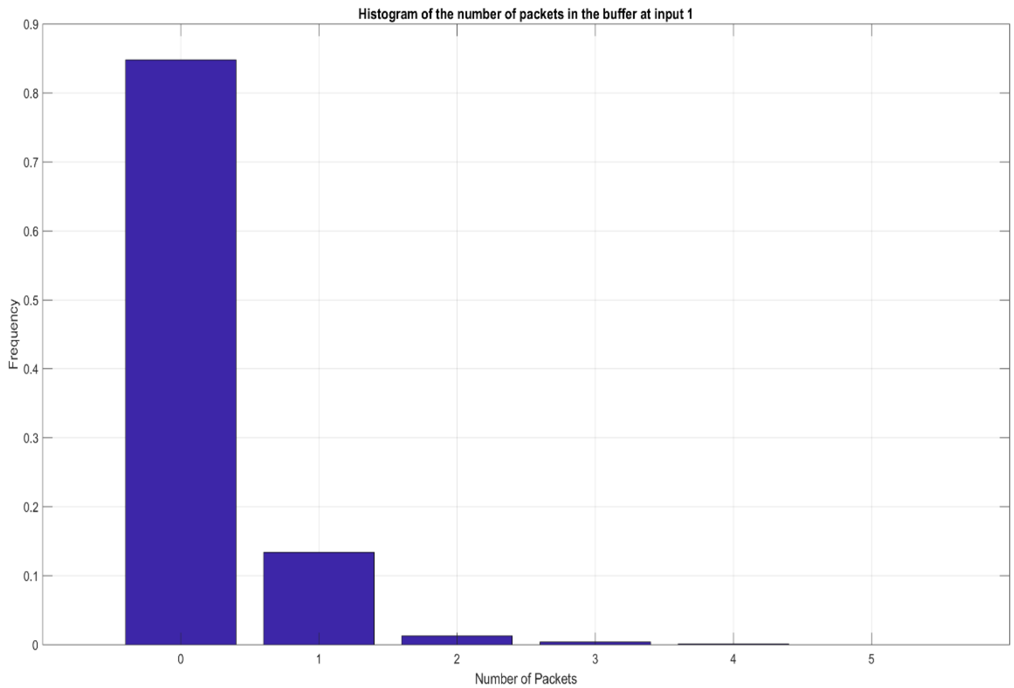
Clearly, the efficiency decreases and it is more easily for the buffer to pile up packets.

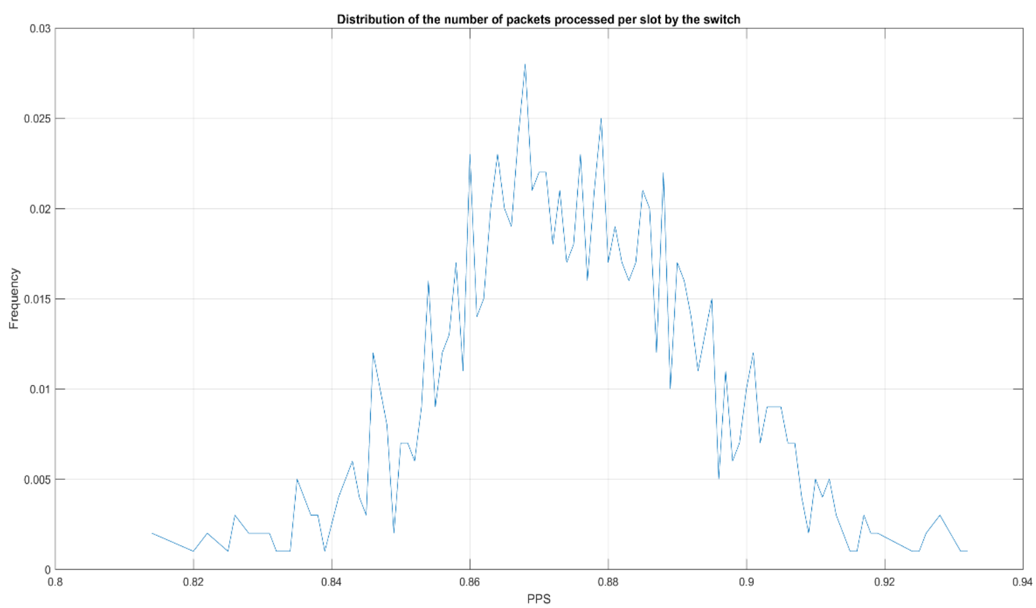


p	CI efficiency (%)
0.1	[7.75,12.75]
0.2	[16.75,23.75]
0.3	[25.50,34.50]
0.4	[35.25,45.25]
0.5	[43.75,53.75]
0.6	[55.25,62.75]
0.7	[60.75,67.00]
0.8	[61.25,65.50]
0.9	[60.75,70.55]

Mean_N1_pkt = 0.1573

Mean_Throughput = 0.8754





The distribution of the number of packets in buffer is a geometric distribution; and to my expectation, the distribution of the number of packets processed per slot is a normal distribution. This can be explained by the Central limit theorem.

For a 4x4 switch with balanced traffic, throughput = 2.6060 pps

For a 8x8 switch with balanced traffic, throughput = 4.9820 pps

5. CODES

For 2x2 Switch

(i)

```
function [nstate, efficiency1, efficiency2] = Input(p,r1,r2,tend)
%
% nstate: record how many packets are present in input1 and input2
% p: the arriving probability
% r1: the probability to target at output 1
% r2: the probability to target at output 2
% tend: continue until tend

% nstate how many packet in each of the input
nstate = [p > rand(1) , p > rand(1) ];
passed = 0;
maxpassed = 0;

% r1 = 0.5; r2 = 0.5;
% nInput1 = nstate(1);
attempt = [r1 > rand(1) , r1> rand(1)];
nextAttempt = [r1 > rand(1) , r1> rand(1)];

% nInput2 = nstate(2);
priority = 1;
t = 1;

while t<tend
    if sum(nstate) > 0
        if nstate(1) > 0 && nstate(2) == 0
            nstate(1) = nstate(1) - 1;
            passed = passed + 1;
            maxpassed = maxpassed + 1;
        else
            if nstate(1) == 0 && nstate(2) > 0
                nstate(2) = nstate(2) - 1;
                passed = passed + 1;
                maxpassed = maxpassed + 1;
            else % when both have packets at the input
                if attempt (1) ~= attempt (2)
                    nstate(1) = nstate(1) - 1;
                    nextAttempt = [r1 > rand(1) , r1 > rand(1)];
                    passed = passed + 2;
                    maxpassed = maxpassed + 2;
                else
                    if priority == 1
                        nstate(1) = nstate(1) - 1;
                        priority = - priority;
                        nextAttempt = [r1 > rand(1) , attempt(2)];
                    else
                        nstate(2) = nstate(2) - 1;
                        priority = - priority;
                        nextAttempt = [attempt(1) , r1 > rand(1)];
                    end
                    passed = passed + 1;
                    maxpassed = maxpassed + 2;
```

```
                end
            end
        end
        % else
        %     nextAttempt = [r1 > rand(1) , r2 > rand(1)];
    end

    % New packet arrives
    nstate = nstate + [p>rand(1) ,p>rand(1)];
    attempt = nextAttempt;
    t = t+1;

    % efficiency = passed/t;
    % nstate

end

efficiency1 = passed/maxpassed;
efficiency2 = passed/(t*2);
```

(ii)

```
buffer = zeros(1,100);
k = 1;

for p = 0.1:0.05:1
    for i = 1:100
        [nstate,efficiency1,efficiency2] = Input(p,0.5,0.5,200);
        buffer(i) = sum(nstate);
        efficiency(i) = efficiency2;
    end

    recordNumber(k) = mean(buffer);
    recordEfficiency(k) = mean(efficiency);
    k = k+1;
end

figure,
plot (0.1:0.05:1 ,recordNumber);
title ( 'Packets in buffer given different p (200 time slots)');
xlabel ( 'p' );
ylabel( 'Number of packet in buffer after 200 time slots');

figure,
plot (0.1:0.05:1 ,recordEfficiency);
title ( 'Efficiency given different p(200 time slots)');
xlabel ( 'p' );
ylabel( 'Efficiency (\times 2 is the number of packet processed per time
slot)');
```

(iii)

```
close all;
clear;
```

```
clc;

times = 1000;
casei = 1;

for p = 0:0.1:1

    N1_pkt = zeros(1,times);
    N2_pkt = zeros(1,times);
    throughput = zeros(1,times);

    for i = 1:times
        [N1_pkt(i), N2_pkt(i), throughput(i)] = Load_Data(p,casei);
    end

    p
    Mean_N1_pkt = mean(N1_pkt)
    Mean_throughput = mean(throughput)
end
```

(iv)

```
function [N1,N2,throughput] = Load_Data(p, casei)

n_slot = 1000;
P1 = rand(1,n_slot);
P2 = rand(1,n_slot);
R11 = rand(1,n_slot);
R21 = rand(1,n_slot);

% Packets processed per slot
pps = zeros(1,n_slot);
throughput = 0;
N1 = 0;             % Number of packets in buffer 1
N2 = 0;             % Number of packets in buffer 2

if casei == 1
    r = 0.5;
else
    r = 0.75;
end

for i = 1:n_slot
    % Situation that input 1 and input 2 both have packets

    if(P1(i)<p && P2(i)<p)
        % Input 1 switches to output 1 and input 2 switches to output 2

        if(R11(i)<r && R21(i)>r)
            pps(i) = 2;
            [N1] = popbuffer(N1);
            [N2] = popbuffer(N2);
        end

        % Input 1 switches to output 1 and input 2 switches to output 2
```

```
    if(R11(i)>r && R21(i)>r)
        P_sel = rand(1);

        % Select packet with the same probability
        if P_sel <= 0.5
            pps(i) = 1;
            [N1] = popbuffer(N1);
            [N2] = pushbuffer(N2);
        else
            pps(i) = 1;
            [N1] = pushbuffer(N1);
            [N2] = popbuffer(N2);
        end
    end

    % Input 1 switches to output 2 and input 2 switches to output 1

    if(R11(i)>r && R21(i)<r)
        pps(i) = 2;
        [N1] = popbuffer(N1);
        [N2] = popbuffer(N2);
    end

    % Input 1 switches to output 1 and input 2 switches to output 1

    if(R11(i)<r && R21(i)<r)
        P_sel = rand(1);

        % Select packet with the same probability
        if P_sel <= 0.5
            pps(i) = 1;
            [N1] = popbuffer(N1);
            [N2] = pushbuffer(N2);
        else
            pps(i) = 1;
            [N1] = pushbuffer(N1);
            [N2] = popbuffer(N2);
        end
    end
end

% Situation that input 1 has packet but input 2 doesn't

if(P1(i)<p && P2(i)>p)
    pps(i) = 1;
    [N1] = popbuffer(N1);
end

% Situation that input 2 has packet but input 1 doesn't

if(P1(i)>p && P2(i)<p)
    pps(i) = 1;
    [N2] = popbuffer(N2);
end

% Situation that input 1 and input 2 both don't have packet
```

```
    if(P1(i)>p && P2(i)>p)
        pps(i) = 0;
    end
end
throughput = sum(pps)/n_slot;
end
```

(v)

```
function [N1] = popbuffer(N1)
% if the number of packets in buffer > 1 then pop one packet out
% if number <= 0 then number becomes 0 after pop

if N1>1
    N1 = N1 - 1;
else
    N1 = 0;
end
```

(vi)

```
function [N1] = pushbuffer(N1)
% Push one packet into buffer, number increases by 1

N1 = N1 + 1;
End
```

(vii)

```
close all;
clear;
clc;

times = 1000;
p = 0.5;
casei = 1;

N1_pkt = zeros(1,times);
N2_pkt = zeros(1,times);
throughput = zeros(1,times);

for i = 1:times
    [N1_pkt(i), N2_pkt(i), throughput(i)] = Load_Data(p,casei);
end

Mean_N1_pkt = mean(N1_pkt)

figure,
A = 0:1:5;
[a,b] = hist(N1_pkt,A);
bar(b,a/sum(a));
grid on;
title('Histogram of the number of packets in the buffer at input 1');
xlabel('Number of Packets');
ylabel('Frequency');

Mean_throughput = mean(throughput);
```

```
figure,
A = unique(throughput);
dist_throughput = histc(throughput,A);
a = dist_throughput/sum(dist_throughput);
plot(A,a);
grid on;
title('Distribution of the number of packets processed per slot by the
switch');
xlabel('PPS');
ylabel('Frequency');
```

(viii)

```
close all;
clear;
clc;

times = 1000;
casei = 1;

for p = 0:0.1:1

    N1_pkt = zeros(1,times);
    N2_pkt = zeros(1,times);
    throughput = zeros(1,times);

    for i = 1:times
        [N1_pkt(i), N2_pkt(i), throughput(i)] = Load_Data(p,casei);
    end

    p
    Mean_N1_pkt = mean(N1_pkt)
    Mean_throughput = mean(throughput)
End
```

Generalizing to NxN switch

(i)

Balanced Traffic :

```
close all;
clear;
clc;

prompt = 'What is the size of the switch?';
N = input(prompt)

Hash = zeros(1,N);
total = 0;
steps = 1000;
num = N;

for j = 1:steps
```

```
    empty = randi([1,N],1,num)
    for i = 1:num
        Hash(empty(i)) = Hash(empty(i)) + 1;
    end
    num = 0;

    for i = 1:N
        if Hash(i) > 0
            Hash(i) = Hash(i) - 1;
            num = num + 1;
        end
    end

    total = total + num;
end

pps = total/steps;
```

(ii)

Hot-Spot Traffic :

```
close all;
clear;
clc;

prompt = 'What is the size of the switch?';
N = input(prompt)

Hash = zeros(1,N);
total = 0;
steps = 1000;
num = N;
k = 2;

for j = 1:steps

    empty = randsrc(1,num,[1,2,3,4;1/k,(k-1)/(k*(N-1)),(k-1)/(k*(N-1)),(k-
1)/(k*(N-1))])
    for i = 1:num
        Hash(empty(i)) = Hash(empty(i)) + 1;
    end
    num = 0;

    for i = 1:N
        if Hash(i) > 0
            Hash(i) = Hash(i) - 1;
            num = num + 1;
        end
    end

    total = total + num;
end

pps = total/steps;
```