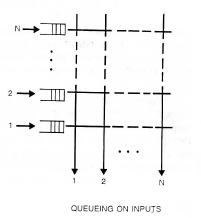
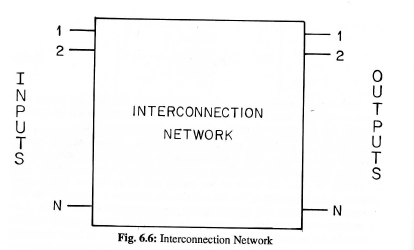
# N\*N CROSS BAR SWITCH

**ESE546: Computer Communication Networks Jaya Godugu 108209853**

**Cross Bar Switch (Input Queuing):**

A Simple way to design interconnection network is with a cross bar architecture. In an N\*N crossbar switch the inputs are connected to horizontal wires (buses) and the outputs to vertical buses. There is a switch which can be independently closed or opened; everywhere a horizontal bus crosses a vertical bus. These are Interconnection cross points. An “X” in the figure indicates a closed switch creating a path from input to output. Such a crossbar switch is said to be a space division in that the distinct paths through it.



Some of these switches are N/w blocked; these reduce number of cross points at the expense of blocking. Blocking N/w can be advantageous as they reduce the number of cross points. In what follows through, a non-blocking packet switch is considered with queues at the input. The reason for queuing is that as packets arrive they may be destined for the same output. Only one of these packets can use the output line at a time, and the rest are queued.

For Input queuing we have the following relation’s:

Where: P🡪 saturation throughput,

Ri-> No of packets at the head of the input queues that are blocked at the end of the mth time slot and are destined for output I i.e. number of packets destined for output I that are not selected by switch controller during the mth time slot.



Also we have the following below equation where

Fm-> is the number of free input queues at the end of (m-1) time slots

N-> Switch size



Hence at equilibrium we have:



**Pseudo code:**

Simulation of an N\*N switch (Input queuing):

1. Vary for different switch size starting from a minimum value to a value say 100
2. Initialize the No. of input, output packets
3. Allocate memory according each of the queues \* No. of queues through the calloc function, similarly do in order to find out the length of the queue and line\_stats describing the status of the O/p lines
4. Input Processing: do this for MAX\_TIME\_SLOTS in this case set to 10,000
5. To add a packet to input queue the conditions to be fulfilled are:
6. The queue should not be full
7. The packet itself identifies the output line
8. If the above functions are satisfied call the pkt\_arrival function to add the packets
9. Output Processing:
   * 1. Select an output line between 1 to N for a given queue randomly
     2. Check if the line is free. The line is free if dispatch probability is greater than out probability since we are not interested in packet, we simply decrement the number of element count in the queue
10. If the above conditions are satisfied the call the function pkt\_dispatch() to process the output
11. Calculate the through put at each run called cur\_thruput given by (no packets/ switch size)
12. Calculate the throughput for the given switch size through the cumulative sum of curr\_thruput and take the average
13. Repeat steps 1-10 for varying switch size.

**Code:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <strings.h>

#include <stdlib.h>

#define MAX\_SWITCH\_SIZE 100

#define MAX\_TIME\_SLOTS 20000

#define QSIZE 9

/\*

#ifndef DEBUG

#define DEBUG

#endif

\*/

struct que{

int buf[QSIZE];

int ele\_cnt;

};

int pkt\_arrival(struct que \*ptr, int sw\_size ){

int count, rnum;

count= ptr-> ele\_cnt;

// printf("count %d\n ", ptr -> ele\_cnt);

if(count < QSIZE){

rnum = (rand()%sw\_size);

ptr-> buf[count] = rnum;

ptr->ele\_cnt= (count+1);

// printf("count %d\n ", ptr -> ele\_cnt);

return 1;

}

else

return 0;

}

/\*--------------------------------------------------------------------------------------------------------- \*/

int pkt\_dispatch(struct que \*ptr, int line\_stat[]){

int i, count, packet;

packet = ptr -> buf[0];

count= ptr-> ele\_cnt;

if(count >0){

if(line\_stat[packet] == 0){

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

ptr->buf[i]= ptr->buf[i+1];

ptr->ele\_cnt= (count-1);

line\_stat[packet] = 1;

return 1;

}

else

return 0;

}

return 0;

}

/\*--------------------------------------------------------------------------------------------- \*/

void printq(struct que \*ptr){

int i;

printf("Data-0 ");

for(i =0; i < (ptr -> ele\_cnt); i++)

printf("%d ", ptr -> buf[i]);

printf("\n");

}

/\*----------------------------------------------------------------------------------------------\*/

main()

{

int que\_num;

struct que \*ptr;

int sw\_size, num\_slots;

int i, n, len;

int no\_in\_pkts, no\_out\_pkts;

float thruput, cur\_thruput;

int line\_stat[200];

FILE \*fp;

fp = fopen("result","w");

if(fp == NULL)

printf("Error in openting file\n");

fprintf(fp, " Switch\_Size Thruput\n");

/\* N X N switch ...... N iputs and N outputs \*/

for(sw\_size= 1; sw\_size <= MAX\_SWITCH\_SIZE; sw\_size++){ /\* for each switch size \*/

no\_in\_pkts = 0;

no\_out\_pkts = 0;

thruput = 1.0;

ptr = (struct que \*)calloc(sw\_size, sizeof(struct que)); /\* memory for N - input queues intialized to 0\*/

len = (sw\_size \* sizeof(struct que));

// line\_stat = (int \*)calloc(1, sw\_size \* sizeof(int) );

for(num\_slots = 0; num\_slots < MAX\_TIME\_SLOTS; num\_slots++){ /\* for each time slot \*/

/\* INPUT processing

To add a pkt to input que the conditions to be fullfilled are

1. The que should not be full

2.The packet itself identifies the output line

\*/

no\_out\_pkts=0;

// printf("in TS%d \n", num\_slots);

for(que\_num = 0; que\_num < sw\_size; que\_num++){ /\* for each que \*/

struct que \*qptr;

qptr = ptr + que\_num;

// printf("ptr %x qptr %x \n", ptr, qptr);

if(pkt\_arrival(qptr,sw\_size ))

no\_in\_pkts++; /\* one pkt added to this que \*/

}/\* end input processing \*/

/\*--------------------------------------------------------------------------- \*/

#ifdef DEBUG

for (i=0; i<sw\_size;i++){

printf("------------------------Que %d--------------\n", i);

struct que \*qptr;

qptr = ptr + i;

printq(qptr);

}

#endif

/\* OUTPUT processing

1. Select an output line between 1 to N for a given que randomly

2. Check if the line is free. The line is free

if dispatch probabilty is grater than out probability

since we are are not interested in pkt, we simply decriment the num elemnt count in the que \*/

for( i=0; i<200; i++)

line\_stat[i] = 0;

for(que\_num = 0; que\_num < sw\_size; que\_num++){ /\* for each que \*/

struct que \*qptr;

qptr = ptr + que\_num;

// printf("ptr %x qptr %x \n", ptr, qptr);

if(pkt\_dispatch(qptr, line\_stat))

no\_out\_pkts++; /\* one pkt dispatched from this que \*/

}/\* end out processing \*/

cur\_thruput=(float )no\_out\_pkts/(float )sw\_size;

thruput= (thruput + cur\_thruput)/2.0;

// printf("%f is the cur\_thruput out pkts = %d thruput %f \n",cur\_thruput, no\_out\_pkts, thruput);

}

/\*---------------------------------------------------------------------------------\*/

fprintf(fp, " %d %f\n", sw\_size, thruput);

free(ptr);

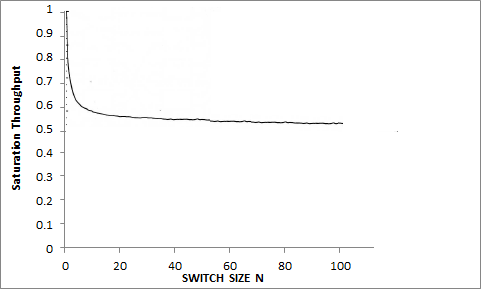
}/\* end of sw\_size \*/

fclose(fp);

}/\* end main \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**GRAPH:**



**Conclusion:**

The saturation throughput is the maximum when the switch size =1 i.e. only 1 input line/bus, which means as and as and when the input is it service at the output. Now on increasing the switch size further i.e. as and when more and more input lines are added the saturation throughput keeps gradually decreasing. Now as the switch size tends to grow very large i.e.  the saturation output gets saturated at 0.586 as shown in the graph and given by the below formula:



This upper bound on the throughput of any non-blocking interconnection network based on switch using input queuing as  which is seen in the graph.