

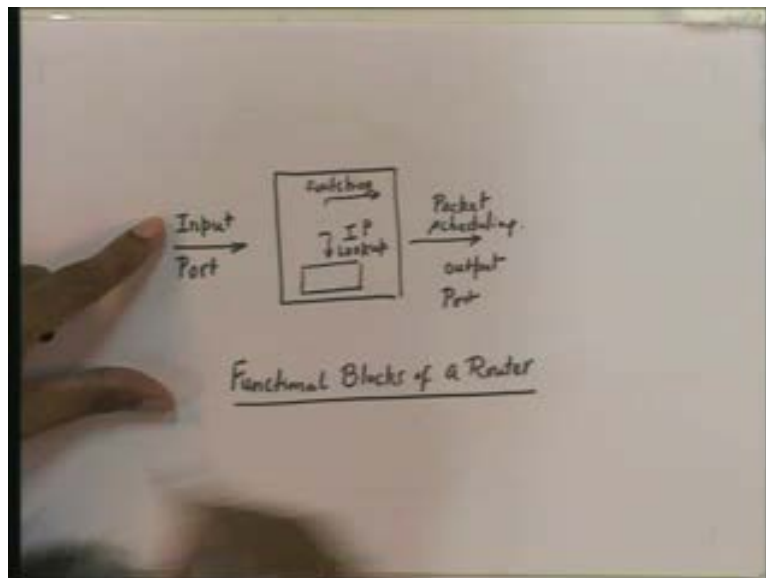
Broadband Networks
Prof. Abhay Karandikar
Electrical Engineering Department
Indian Institute of Technology, Mumbai

Lecture - 31

Metro Ethernet Access Networks

In the previous lectures, we had studied the concept of packet switching. Now, in today's lectures we will see how these packet switches are built and what are the issues involved in these packet switches. Now, we will refer a packet switch by generic name let us say, router and we had seen in the previous lectures that what the major components of a router. So, we will just review the major components of a router.

(Refer Slide Time: 1:50)



So, the major components of a router; this is the input ports, so in the input port typically the packets will come on the input ports and the packets may go for the IP lookup block. So, there may be IP lookup operations and then there may be a switching involved which will switch the packets from a particular input ports to a particular output port and this output port may have the packet scheduling as well.

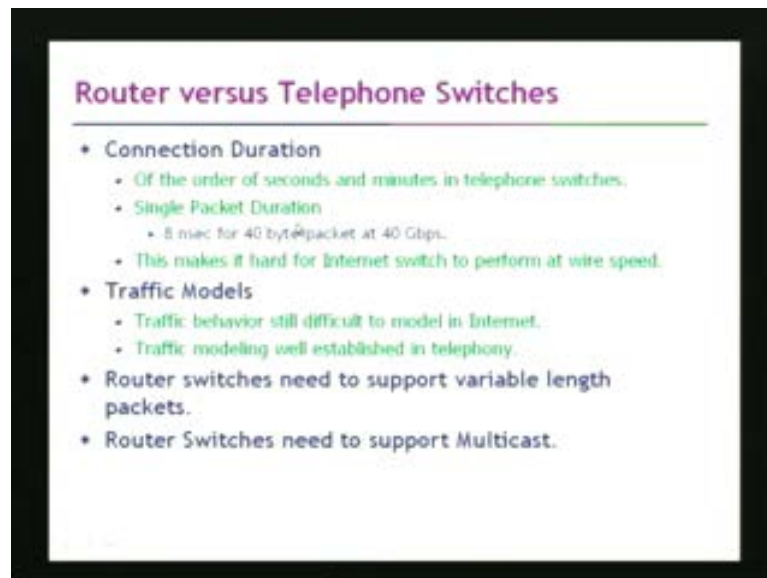
Functional blocks of a router: now, we had already discussed the IP lookup algorithms in our previous lectures and we had already discussed the packet scheduling algorithms. Today we would see the switching. So, the major function of a switching is to switch the packets from an input port to an output port. Typically, these switching operations must be done at the wire speed. By wire speed we mean that before the next packet arrives, the switching logic must take a decision of switching the packets to an output port.

So basically, the switching functionality also becomes complex if the switch has to operate at a high speed. Also, one important difference is that the packets could be of variable length, so

typically as we will see that the approach that is adopted in practice is to divide these variable length packets into fixed length packets in a switch and then reassemble these fixed length packets into the original variable length packet.

The job of the switching is considerably simplified if the packets are of the fixed length. So, let us understand what are the issues that are involved in switching and what are the complexities associated with a switching and then we will suggest solutions to address some of these complexities.

(Refer Slide Time: 4:30)



So, first of all let us look at what is the difference from the switching point of view, between a router and a telephone switch. Now, we are using the word router for a generic packet switch. This is not necessarily to assume that the router is doing the job of routing a packet etcetera. So, one major difference is the connection duration. Now, the connection durations in a telephone switch will be of the order of seconds, if not minutes.

But on the other hand, the connection duration in a router will be a single packet durations and if we consider the packet switch in the core of the networks which will be operating at high speeds of let us say 40 gigabytes per second, then the connection time would be of the order of 8 nanoseconds for a small 40 byte of a packet. So, that is how the orders of magnitude that we are talking of in terms of the connection durations. The connection duration in a telephone switch can be of the order of minutes or while the connection duration in a router will be of the order of milliseconds to nanoseconds kind of time scales.

Now, this makes it hard for the internet switch to perform the switching at the wire speeds. The second difference is in the traffic models and now the traffic behavior is still very difficult to model in internet. There are many traffic models that have been proposed in the literature but still, the traffic behavior is still difficult but on other hand the traffic modelling is very well established in the telephony.

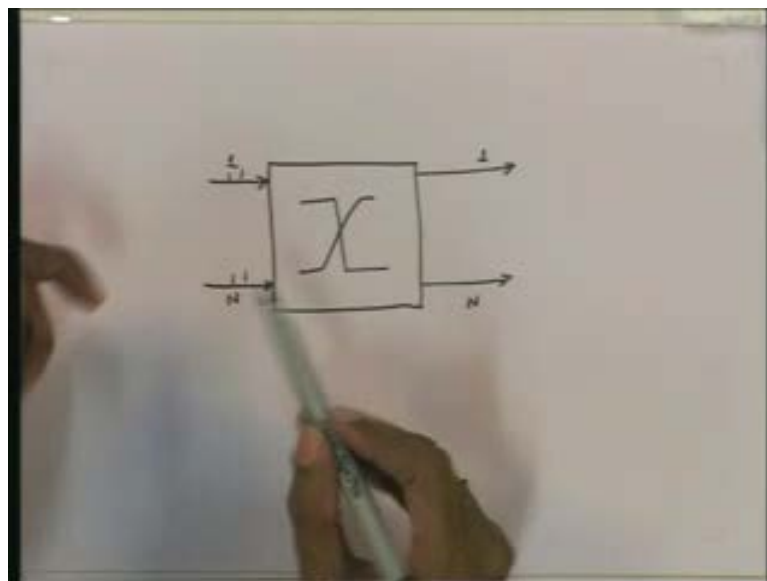
So, the dimensioning a switch and designing a switch becomes extremely simple if the behavior of the traffic model which a switch is going to handle if that is very well known. So,

this is also one of the major differences between switch that is used in a router and a switch that is used to switch connections in a telephone switch.

The third major difference is as I have already pointed out that a router switches, they need to support the variable length packets and this makes the job of the switching considerably complex. On the other hand, in the telephones, the connection will be sliced in the time slots and therefore the variable length packet is not really an issue. Another important difference is that the routers also need to support the multicast connections and that also makes a job of the switching complex.

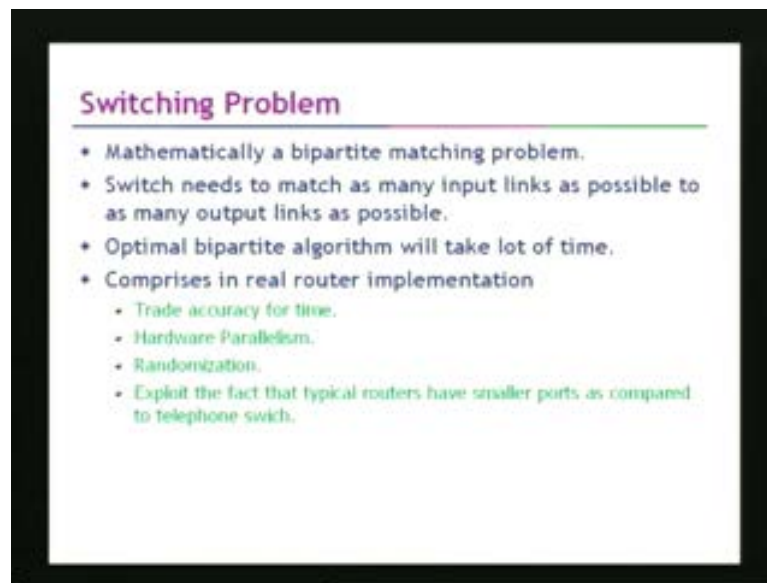
Now, let us look at what is actually a switching problem. So, if you really **see** look at the switching problem; then in switch typically you may have some input ports.

(Refer Slide Time: 7:14)



So, may be 1 to n input ports and 1 to n output ports and this is a switch. So, the job of the switching really is to switch the packets which are coming from the input ports to the output ports.

(Refer Slide Time: 7:44)



Mathematically, **if you look at it** if you look at this time, mathematically this is really a bipartite matching problem if you view the switch as a graph and switch, it needs to match as many input links as possible to as many output links as is possible.

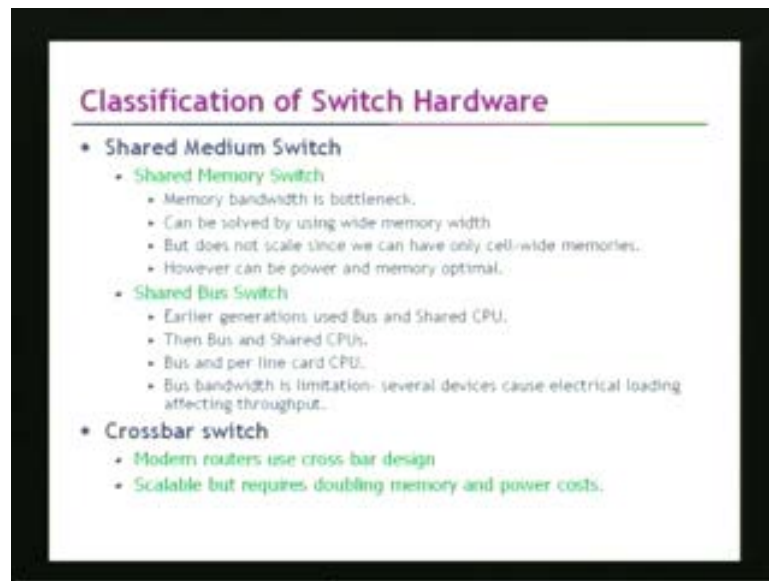
So, what really have to see is that in a particular switch, we need to see that none of the output ports is free **none of the output ports is free** if there is a packet at the input port which is dusting for a particular output port. So, if there is a packet here at the input ports, we should make sure that none of the output ports is free and at the same time, we also need to see that if an input port has some packets to send, then it should be able to send it and it should not be blocked.

Now, in optimal bipartite algorithm, we will take lot of time, the optimal bipartite type matching algorithms are very well known in graft theoretic literatures and they are likely to take lot of time. So, some compromises are done in the real router implementations. So, typically, various compromises can be done. You can either trade accuracy for time or you can introduce some kind of hardware parallelism or you can use randomizations and also you can exploit the fact that typical routers have smaller number of ports as compared to a telephone switch.

So, now again briefly if you see the the fundamental principle of a switching is that there are input ports and then there are output ports, the job of the switching is to switch packets from the input ports to the output ports in an optimal fashion and mathematically, this problem can be posed as a bipartite matching problem. The job really of the switching is to make sure that as many output links are matched to the input ports as is possible in a small amount of time and once that match is found, then you make the make the connections and allow the input packets to get switched to the output packets.

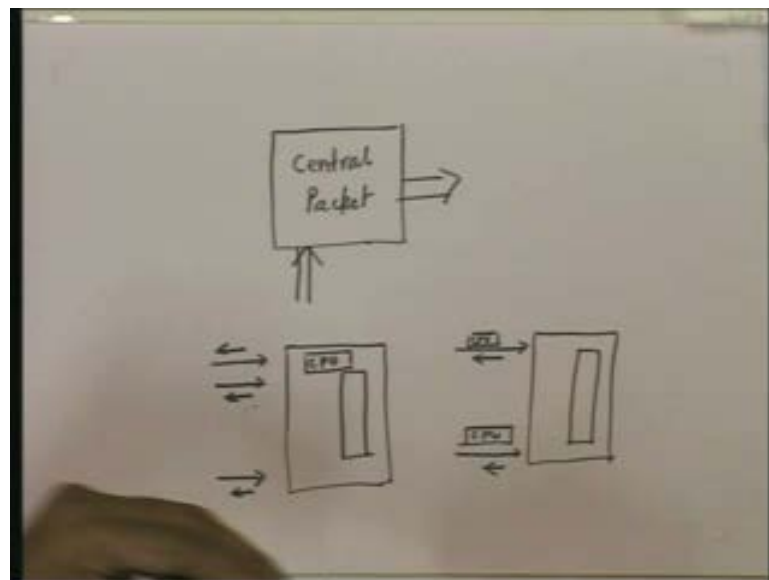
So, but as we have seen that since an optimal algorithm will may take lot of time because of its complexity and therefore some trade-offs and some comprises have to be made in the real router implementation and we will see what are those kinds of trade-off that are possible to be made in a real router implementations.

(Refer Slide Time: 10:06)



Now, we will classify the the switch headers based on several types. One of the most common switch hardware that was used in the past was the shared memory switch and in the shared memory switch, really what is done is that you have a memory, you have you may have a central memory. Basically, so this is central packet memory and we have the input ports and then we can have the output ports.

(Refer Slide Time: 10:28)



So, packets are written into the memory and then they once you determine that on which output port the packets needs to be switched, the packets are read out of the memory. So obviously, in a shared memory switch, the memory bandwidth is really the bottleneck.

The memory if there are n input ports and n output ports, then the memory bandwidth has to be $2n$ times the input link rate. It has to be $2n$ times the link rates so that you can write it

into the memory and at the same time, you can read out of the memory at the faster speeds. Of course, one of the advantages of the shared memory switch is that you do not have to duplicate memories. The central memory serves also as a packet buffer and at the same time, it serves as a shared memory switch also.

There are various designs of a shared memory switch that were proposed in the literature because typically in a router there were several line cards. So, there were several input line cards and then there were at the same time, there could be output ports on this and the shared memory switch was a common memory and there may be a general purpose CPU that was...

So, what we used to happen is that the packets will go to the general purpose CPU, the header of the packets to make a decision where the packets needs to be switched to the output port while the packet is stored in the buffer or the memory and once the CPU determines where the packet needs to be switched, it will read out the packet from the memory and then it will put it onto the output ports.

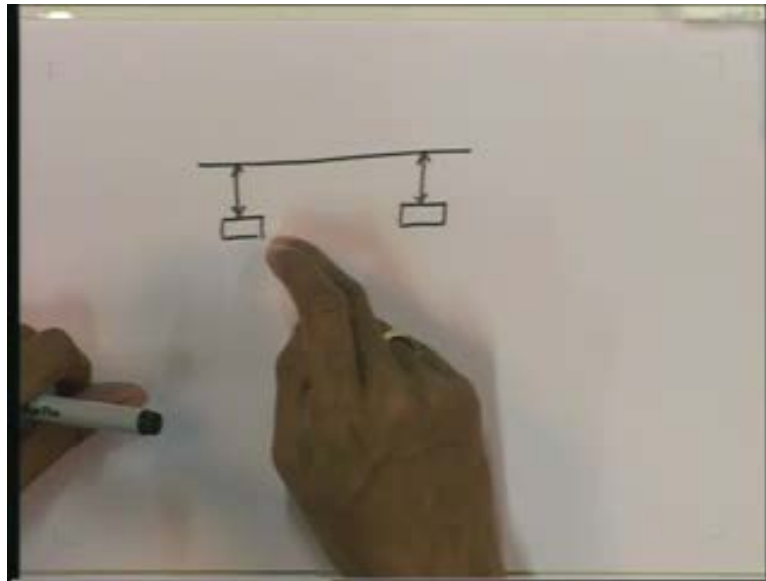
So, this was earlier generation architectures. Then some changes in the architectures took place and what really happened is that this general purpose CPU which could have become a bottleneck in this centralized architectures, each input line cards were also having then a CPU to make a forwarding decision **so** and then there was a memory.

So, you had a CPU in each of these line cards. So, this was distributed kind of architectures, this was more centralized architecture where there is only one general purpose CPU. So, if you look at it is that in the shared memory switch, really memory bandwidth is the bottleneck. However, you can solve the memory bandwidths by using a cell wide memory width. What you can do is that you can use a byte wide or a packet wide memory bits so that the entire packet can be written at once into the memory. So, that is also possible to address the problem of the memory bandwidth bottlenecks.

But however this approach really does not scale and since we can have only the cell wide memories if their cell length is smaller and however they are already pointed out it can be power and memory optimal. So, while shared memory switches are really good if the number of ports, if the number of ports that is n is small; so if the number of ports is small, then there is not so much stress on the memory bandwidth that is manageable. But at the same time the advantage can be that it can be both power and memory optimal. You do not have to duplicate the packet buffers. The central packet buffers serves not only as the place to store the packets but also some kind of a switch.

So, this is one advantage of having a shared memory switch. But obviously, this architecture does not scale if the number of ports increases. So, that is the disadvantage of a shared memory switch. Another switch similar to a shared memory switch is a shared bus switch. So, instead of having a shared memory, you will have some kind of a shared bus.

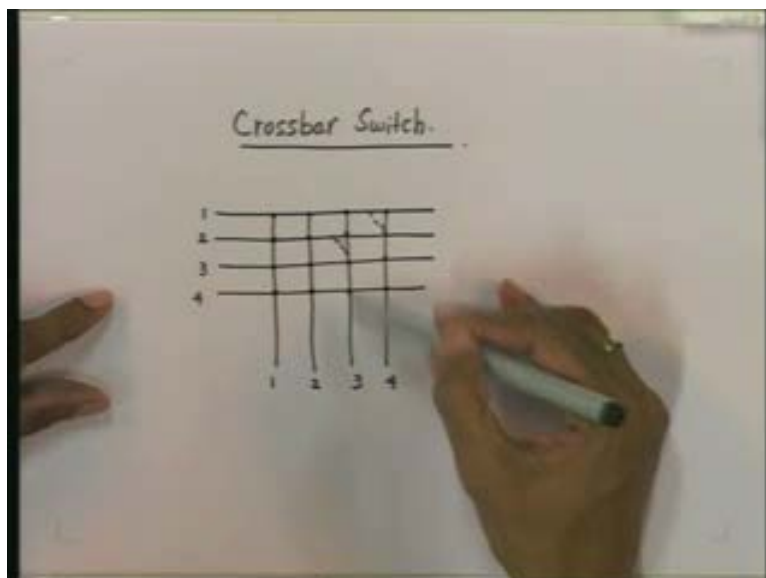
(Refer Slide Time: 14:43)



So, you could have input and output ports connected to this shared bus. Obviously, the bus has to run at the speed of n times the input link rate. So, the bus bandwidth could become a limitation just like as the memory bandwidth is a limitation in a shared memory switch. So, this is the similar architecture, the bus is really time division multiplexed bus and then it has to run at a faster speed.

So, again the disadvantage of shared bus architecture is that it will not scale to large number of ports. If the number of ports is limited, then shared bus architecture can be preferred. The most common on a scalable architecture is based on a crossbar switch. So, a crossbar switch really works like this.

(Refer Slide Time: 15:57)



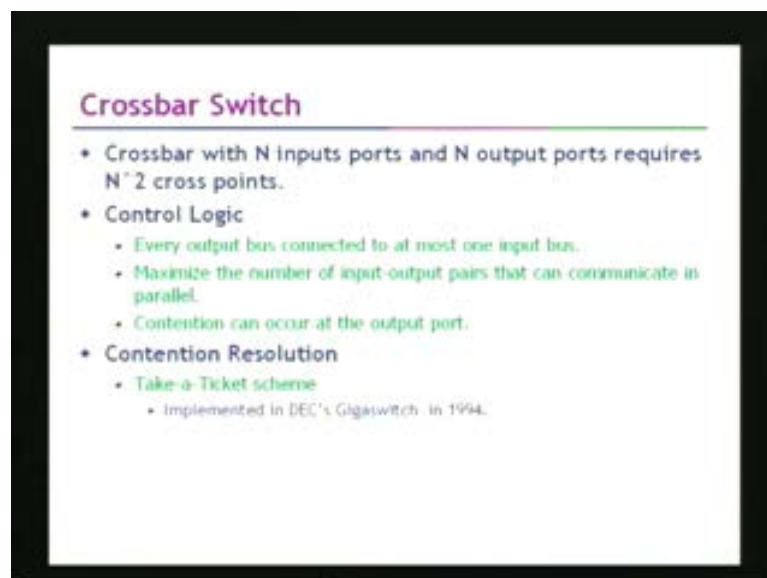
So, this is the most scalable architectures. So, a crossbar will have let us say that you have 4 input ports – 1, 2, 3 and 4 and let us say you have a 4 output ports. So, these are the cross

points. So, in n port crossbar switch, as you can see that the number of cross points will be of the order of the n squares and the advantage of the crossbar switch is that any 2 distinct pairs of input and output ports can communicate simultaneously.

For example, if one wants to send a packet to 4, then you can switch close this and so that that one can send packets to 4 and at a same time, 2 also can send packets to 3 by closing this. So, there may be a switch here, so this switch is closed if one wants to send a packet to 4 and this switch is closed if 2 want to send a packet to 3. Otherwise, these switches will be open and in general there will not be any connections.

So, such a crossbar architecture is scalable but the disadvantage compared to a shared memory architecture or a shared bus architecture is that it will require doubling the packet memory. You will have to have an additional packet memory for storing the packets. But of course, the advantage is that this architecture is scalable.

(Refer Slide Time: 17:35)



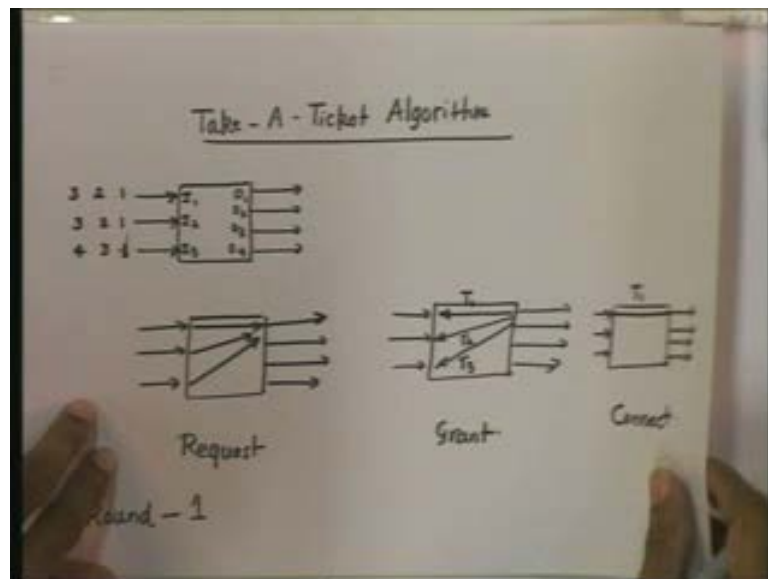
So, as we said that here that a crossbar switch with an n input ports and n output ports, it requires n square cross points. Of course that we require control logic in the crossbar switch where every output bus is connected to utmost one input bus and in this, we need to maximize the number of input/output pairs that can communicate in parallel.

Now, as a result, contention can occur at the output port. So basically, it may be just possible that two input ports, let us say 1 or 2 may want to send a packet to 4. If that is the situations, then only one of these input ports can be switched to an output packet 4. So, a contention at output port may occur. A contention at output port means that more than 1 input port is trying to send packets to the output port.

So basically, you need a control logic that will determine that which input ports can be connected to which output port and we of course have to have a control logic in such a manner that we need to maximize the input/output pairs that can communicate simultaneously. So, one of the simple algorithms that work in practice is like take-a-ticket

algorithms and that was implemented in DEC's giga switch in 1994. So, we will see how that algorithm works. So, let me just explain a take-a-ticket algorithm.

(Refer Slide Time: 18:59)



So, let us say that we have a so we have a 3 input ports let us say and 4 output ports let us say 1, 2, 3, 4 may be you can name it as $I_1, I_2, I_3, O_1, O_2, O_3$ and O_4 and let us say that you have packets which are 1, 2, 3 going to output port 1, output port 2, output port 3. This also output port 1, output port 3 and this is also output port 1, output port 3 and output port 4.

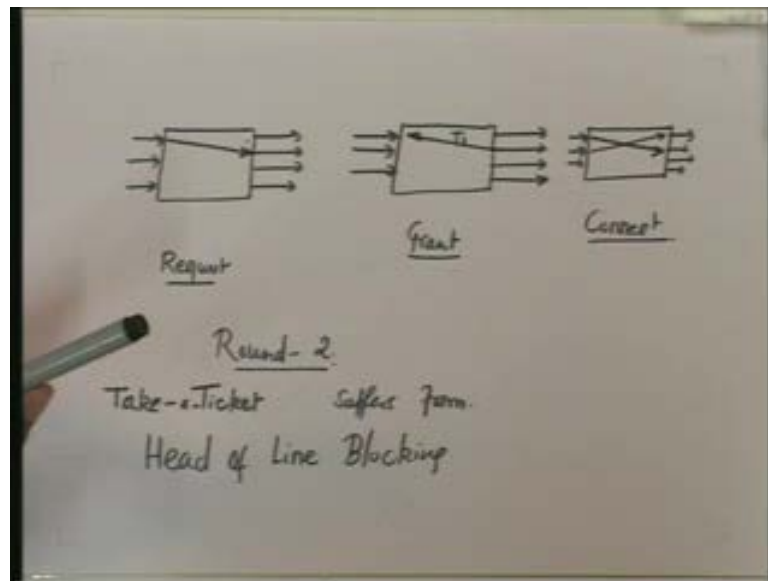
Now, what happens is that this, each of these input ports in the first round, they will communicate. So, this will be a request face. So, the request face would work something like this that these input ports, they will communicate. So, this is a request phase. So, in the request phase, now note that all the 3 input ports, they want to send packets to the output ports. So, each of them will communicate their request. So, here is this request which is sent to an output port 1.

Now, the grant phase, in the grant phase, now this output port 1, obviously can have only one input port. So, it will grant only 1 input port. So, in the grant phase, what will happen is let us say that it gives a grant to send it to, right now it will say first is grant is T_1 that means it should send first as a T_1 , then it will give him the grant as T_2 and then this will give a grant which is T_3 .

So, the input port 1 will first send the packet, then input port 2 and then input port 3. So, this is like a grant phase. So now, first time the connect phase would be that this input port which is the input port 1, this would be sending a packet. So, this is a T_1 time. So, this is the the connect phase.

Now, this is so this is this is the round one, so this is what will happen in the round 1; now, as you can see in round 2, then the input port 2 will send the packet to output port 2 and in round 3. So, let us see what happens in the round 2.

(Refer Slide Time: 22:31)



In round 2, now as you can see that input port 1 has already sent a packet 1; now it has the packet which is to be sent, so input port has a packet which needs to be sent to now 2. So, this packet has already been sent in the first round. So, this is the request phase and input port 1 sends a request to the output port 2. Note that input port 2 and 3, they cannot send request because they still have a packet to be transmitted to output port 1.

So, this is the request phase. Now, this is the grant phase and which of course, the output port 2 will give a grant to the input port. So, the output port 2 can give a grant, let us say call it T_1 the grant. So, this is the grant phase and in the connect phase, as you can see, this input port 1 will send packets to output port 2 and the input port 2 will send the packet to output port 2. So, this is the connect phase. This is happening in the round 2. So on the round 3 will happen and that way the packets can be transmitted through this crossbar switch to the other end.

Now, one disadvantage of this scheme as you can see here is that note here that all the input ports are having the packets to be sent to the output port 1 and obviously when one has been given a grant to send, the port number 2 and 3 they are blocked they cannot send packets.

Now, if you look at behind a packet one, you can see here that when input port 1 was sending a packet to output port 1; input port 2 could have sent the packet to output port 2 and input port 3 could have sent the packet to output port 3. So, it was possible for them to send it if these packets were not there. But now what is happening really is that these head of the line packets are blocking the transmissions of these packets also to otherwise free ports.

So now, this problem therefore is actually **so this problem** is called head of the line blocking. So, this algorithm take-a-ticket suffers from head of the line blocking algorithms. So, we can see here that if the head of the line packets cannot be transmitted due to output port contention, then it blocks all other packets behind even if they are destined for free different output ports.

(Refer Slide Time: 25:49)

Head of Line Blocking

- If Head of Line packet can not be transmitted due to output port contention, it blocks all other packets behind it even if they are destined for different "free" output ports.
- Assuming uniform traffic distribution, HOL blocking can reduce throughput to 58%.

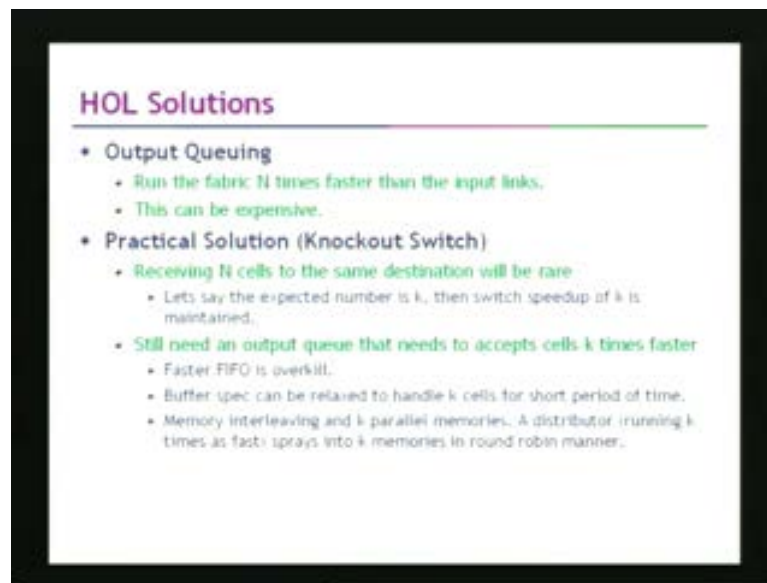
```
graph LR; I1[2] --> S[Switch]; I2[1 2] --> S; S --> O1[Port 1]; S --> O2[Port 2];
```

So, this basically, the head of the line packets basically limits the throughput of a switch and it can be shown that if the traffic distribution is fairly uniform, then the switch throughput will be limited to just 58%.

Now, in practice actually, the traffic distributions may not be uniform. There can be several pathological cases in practice and if that happens, the throughput will be even worse, throughput will turn out to be even less than 58%.

Now, therefore, the whole problem in the switch scheduling is really to address this problem of head of the line packets and schedule or switch the input ports in such a manner that maximum number of input and output pairs matches are achieved. So, that is really the crux of the switch scheduling algorithms. Various algorithms and various approaches have been proposed in the literature to address the problem of head of line blockings and we will see some of these approaches in today's lecture. So, what is a solution for addressing this head of the line blocking?

(Refer Slide Time: 27:05)



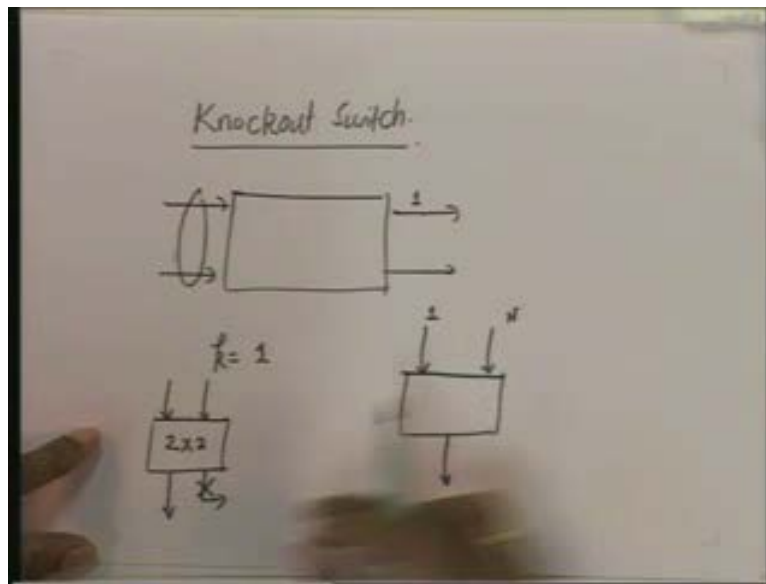
So, the one solution is output queuing. Now, in output queuing, what you do is that so basically, in output queuing, as you have seen, **in the previous** in the previous slides that what we were doing is really is that these packets were being stored at the input port. So, there was a queue at the input ports.

Now, what is done in the output queuing is that you have the queue at the output instead of having the queue at the inputs. What does it mean? It means that let us say the switch is run n times faster **the switch** that means the time slot for an output port is having certain durations and if in that durations, if you can switch 3 packets let us say or n packets for an n cross n switch, if you can switch n packets that means if the switch is running n times faster; then it would have been possible to avoid these head of the line blockage.

So, if you look at our previous examples, so in this case, if the switch is run as 3 times faster then what would have happened that in the first time slot itself all the 3 packets could have been switched to the output port 1. And of course, the output port one cannot transmit these packets simultaneously so then these packets will have to get queued at the output queue. So, this is what is called as the output queuing. So, output queuing as you can see here is one of the solutions to the actual blocking. But however, this can be extremely expensive.

Now, there is one practical solution that has been proposed in the literature to avoid this head of the line blocking and that earliest approach what is called as a knockout switch.

(Refer Slide Time: 28:54)



Now, the basic idea of the knockout switch is that typically, the worst case of the actual blocking would be so if you really see what would be the worst case for the actual blocking if it is an n cross n switch, then the worst case would be that all the n input ports are having a packet which is meant for a particular output port.

So, let us say output port 1; all the n input ports, they want to send a packet to the output port 1. Now, this is really a pathological case. In practice, this is not likely to happen. What in practice if you assume the expected number of ports which are going to send the packets to the same output ports.

So, if you assume here that let us say that this receiving ends cells to the same destination will be really very rare and if you assume that our expected number is k instead of capital N if the expected number is k , then we will require a switch speed up of only k times. As opposed to a perfect output queuing where you need to run the fabric as n times faster than the input links, if you assume that the n cells are not sent to the same destinations; in fact this expected number will be k , then a switch speed up of k is required.

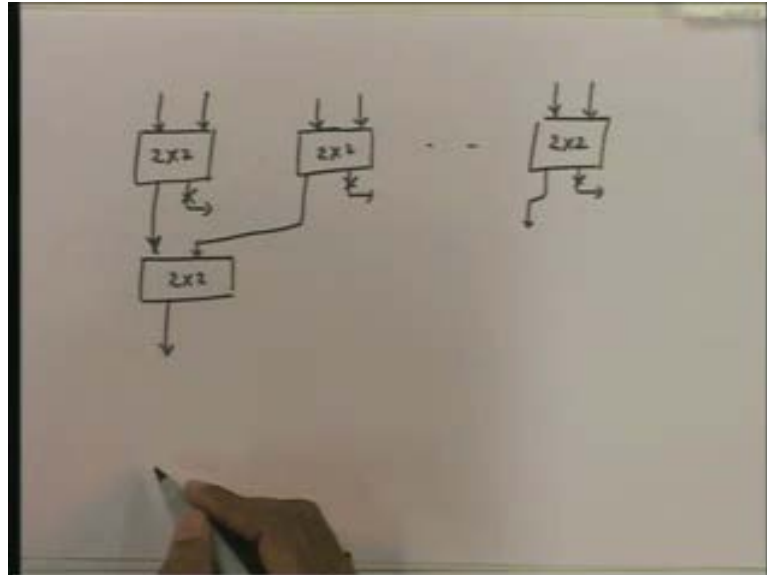
Now however, the problem really in this case is that it may be just possible at sometimes that more than k packets may want to send it to the output ports so in that case you need to select the k packets which need to be switched to the output ports.

so this is really the challenge and this challenge is really achieved by playing some kind of a knockout games which is done in a tennis kind of tournament so that the same thing is done here in the implementations of the knockout switch so let us take for example practical case that say let us say that you have a k is equal to 1 let us say k is equal to one then what does it mean is that you have about n players.

So, one to n players and you need to sort of determine that which one of them is the winner then what you can do is that you can have a 2 by 2 concentrators. So, you can have a match of 2 by 2 . So, you can give 2 input ports to a 2 by 2 concentrator. So, one of them it will randomly select one of them and one will be of course a loser so here is the winner and so

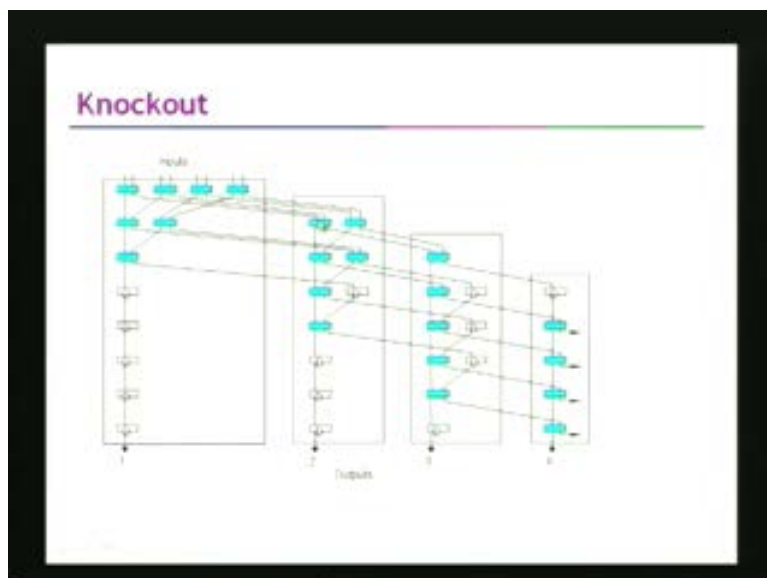
you have such n by 2 such 2 by 2 concentrators and then all the winners which come they again then they go into another 2 by 2 tree structures. So, this becomes like a level like this.

(Refer Slide Time: 32:02)



So, let me just explain here. You have 2 by 2 and so here is the winner and this is definitely the loser and here is the winner, so the losers are all discarded and the winners are then paired up again in a 2 by 2 and again you have a winner and so on. So finally, only one winner will be left and that will be what it will be chosen. Now, if however it is a general k which is not k equal to 1 , then for a general k , you have the architectures something like this is shown in the figure.

(Refer Slide Time: 32:58)

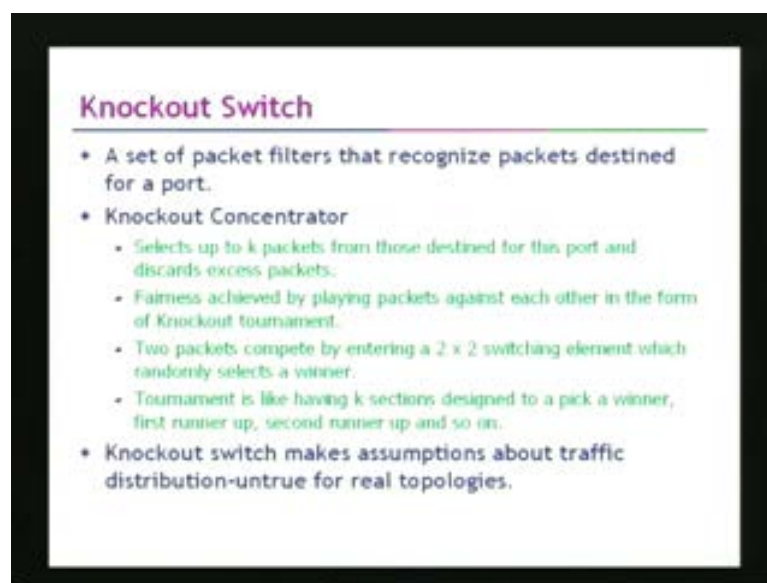


So, in general k what you will do is that loser is also played again. So, here as you can see here that there are 8 input ports, there are 8 input ports and maybe there are 4 outputs that we want. So, this is k is equal to 4 here.

So, as you can see, you give it to 2 by 2 concentrators here. So, one of them is a winner and the winner goes to the second level but the loser comes here. Then again here is the winner which goes into this and the loser. So, there are 4 losers from this and these 4 losers, they come into the next level and then here is third level and so on.

So, this way the losers are also played up again. So, here are the 4 losers. So, this is the winner here and from the next level again, there will be 2 losers which goes to the third level and so on. So, as you can see here that finally we will get one winner from here and one winner from this level and one winner from this level and one winner from this level. So, it is like playing like tennis tournament and finally we will get 4 winners.

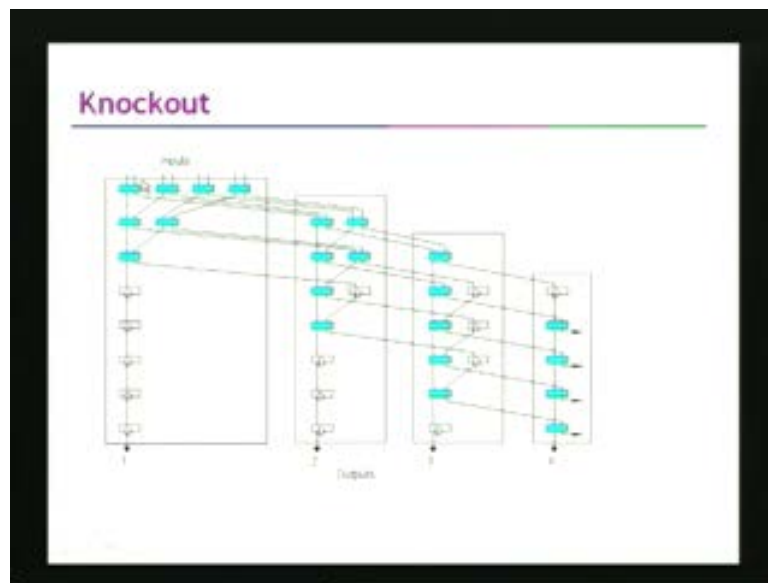
(Refer Slide Time: 34:08)



So, what really you do is that you in knockout concentrators, you select up to k packets from those which are destined for this port and we will discard excess packets. So basically, in this particular example that we had shown, there were 8 packets which were destined for 4 ports and **these** basically the switch is running at a k times speed. So, there were there were 8 packets which were destined to this port and you need to select only 4 packets and you need to discard 4 packets.

So, the fairness is achieved by playing the packets against each other in the form of a knockout tournament and 2 packets compete by entering a 2 by 2 switching element which actually randomly selects a winner and tournament is like having k sections designed to pick a winner, first runner up second runner up and so on. So, as you can see here that there are k sections.

(Refer Slide Time: 35:00)



So, there are actually 4 sections here. This is the first section, second section, third section and fourth section. So, let me just again explain this algorithm again here. So, there are 8 packets or 8 input ports which want to send the packet to a particular output port. Now, our switch speed up is 4. So therefore, out of these 8 packets, you need to select 4 packets which will be sent to this port.

So, how do you select that in the first section, what we have is that we have this 2 by 2 concentrators. So, 4 of them are there. Each 2 by 2 concentrator selects one as the winner and the other as a loser. So, this is the loser and here this is the winner, this is the winner and this is the winner. So, there are 4 winners out of these.

Now, these 4 winners are again paired up in 2 concentrators and then we again get 2 winners. So, again they are paired up in one concentrator and we get finally one winner. So, here is now we get one winner here. Now, these previously 4 losers which were there, they go into the next sections. They go into 2 by 2 concentrators. So, there were 4 losers.

So, output of this will be one winner and one winner here. So, you select 2 winners again. So, that forms sort of one level and the 2 losers which were coming out from here, they again form another 2 by 2 concentrators. So, you get one winner here and another winner here and finally the 2 winners will give one winner and one loser and so on. So, this way you can have four sections. So, this is how a typical knockout switch will operate.

Now, the knockout switch of course that one disadvantage of the knockout switch is that it makes assumptions about the traffic distributions which could be untrue for real topologies. So, this assumption is that on an average, only the k of these packets will be switched in output ports and this k , on an average the number of times that this n is higher than k will be very small. So, that is really the assumptions and therefore you can make do with speed up of k .

So, that is really the fundamental assumption that is used in the design of a knockout switch. However that assumption may not be true in practice. There are again some disadvantages of

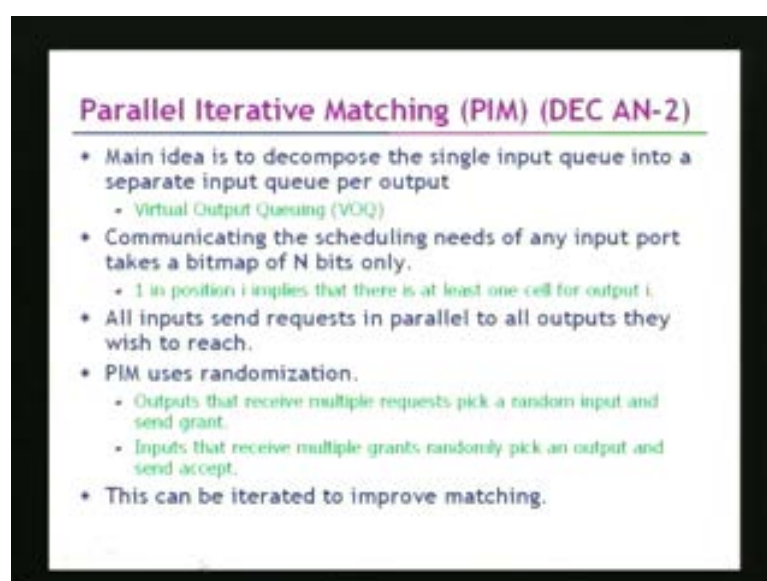
a knockout switch. Knockout switch still leads an output queue that needs to accept the cells k times faster because the switch is still running as a k times faster. So, a faster FIFO, one can have a FIFO but a faster FIFO is really an overkill. One can have a better design where the buffer specifications can be really relaxed to handle the k cells for a very short period of time.

Basically, the fact that these k cells will be switched to an output port, this event is likely to occur only for a short duration of time and therefore having a faster FIFO at the output queue; it may really become an overkill. So, some novel technique is used in the knockout concentrators what you can do really is that you can use memory interleaving and k parallel memories and finally a distributor which runs k times as fast. It actually sprays into k memory in the round robin manners. So, it is like the k parallel memories and there is the distributor, this distributor of course whenever the k cells comes, this distributor will of course have to run the k times faster and it sprays these cells into the memories in a round robin manner.

So, **this is** you can actually use this principle a distributor and an interleaved memory instead of going for a faster FIFO which runs at a speed of k times more. Now basically, as you see that the problem in a switch scheduling is really to address the problem of the head of the line blockings and in the head of the line blockings, our objective is really to maximize the input/output pairs such that the maximum number of packets can be switched from an input port to an output port. So, this is really the objective of a switch scheduling algorithm in practice.

Now however, as we have seen that while a knockout switch was proposed in literatures, very elegant solutions have been recently proposed in the literature that have made their way into the practical routers implementations because they can address the problem of switch scheduling at a almost at a wire speeds and one of the popular algorithms that is actually used in practices what is called as parallel iterative matching or PIM.

(Refer Slide Time: 39:42)

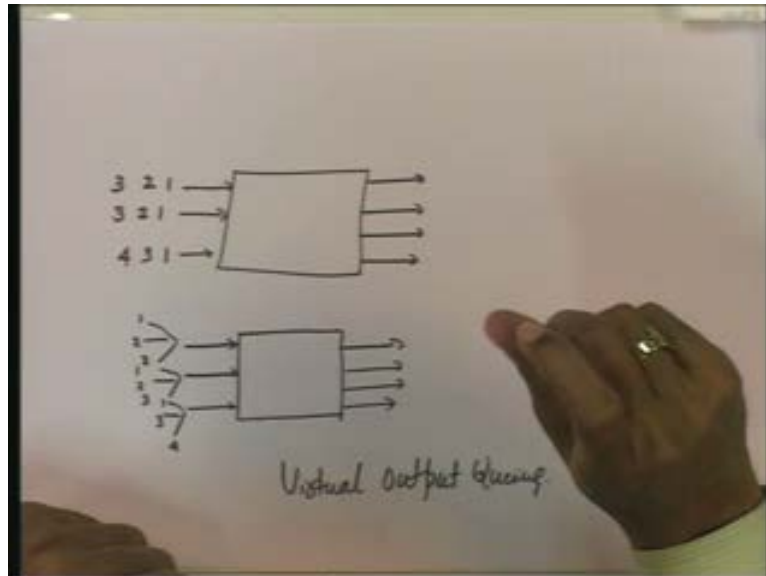


Parallel Iterative Matching (PIM) (DEC AN-2)

- Main idea is to decompose the single input queue into a separate input queue per output
 - Virtual Output Queuing (VOQ)
- Communicating the scheduling needs of any input port takes a bitmap of N bits only.
 - 1 in position i implies that there is at least one cell for output i .
- All inputs send requests in parallel to all outputs they wish to reach.
- PIM uses randomization.
 - Outputs that receive multiple requests pick a random input and send grant.
 - Inputs that receive multiple grants randomly pick an output and send accept.
- This can be iterated to improve matching.

So, let us look at the parallel iterative matching, the concept of PIM. Now, the main idea of the parallel iterative matching is really to decompose a single input queue into a separate input queue per output.

(Refer Slide Time: 40:03)



So, as you can see here that in the previous example that we had shown that we had the packets at the input which were meant actually so this packet 1, 2, 3 and this was again 1, 2, 3 and this was 1, 3, 4. So, what you do is that instead of having an input queue like this, you actually have a queue per output port.

So, this is a queue per output port; 1, 3 of course, on the 2 there is no packet, so 4. So, this is like having a queue per, input separate input queue per output. So, this is what is called as the virtual output queuing.

So, the principle of virtual output queuing was one of the fundamental principle that was used in the parallel iterative matching. So, the main idea was to decompose the input queue into a separate input queue per output port. That resulted in the virtual output queue. And then, what you do is that when you communicate the scheduling needs of any input ports, it will require a bitmap of n bits only, corresponding to n input ports. So, one position in the i 'th implies that there is at least one cell for output I .

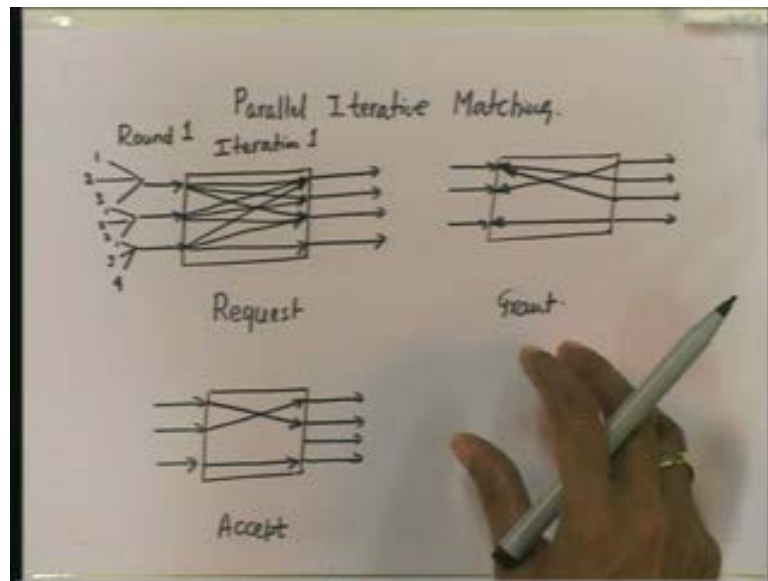
So for example, in this case, there will be a bitmap of 4 since there are 4 output ports: so, 1, 2, 3, 4. So, if you consider the bitmap for input port 1; **so the one** this position will be 1, this position will be 1, this position will be 1 and this position will be 0. Similarly, if you consider the bitmap for port 1, then again this position will be 1, this position will be 1, this position will be 1 and this position will be 0 and similarly the bitmap for port 3 will be this position will be 1, this position will be 0 because there is no packet meant for the output port 2. This position will be 1 and this position will be 1.

So, this is how to communicate the scheduling needs of any input ports. You basically require only about n bits where n corresponds to the output port. Then, what happen really in this

algorithm is all inputs, they will send requests in parallel to all output ports which they wish to reach by using this bitmap.

So, there will be a request phase just like in the take-a-ticket algorithm. So, there will be a request phase and the parallel iterative matching then uses the principle of randomizations. The outputs that receive the multiple requests, they will pick a random input and send the grant and inputs that received multiple grants, they will randomly pick an output and send accepts and this can be sort of iterated to improve the matching because we need to achieve sort of a maximal match.

(Refer Slide Time: 43:15)



So, let me just give you an example of a parallel iterative match. So, for example, we will consider our same example; so here we had 1, 2, 3; similarly 1, 2, 3 and 1, 3, 4. So, this is let us say, this is a round 1. So, you have the request phase. So, this has the input port packet to send in output port 1, output port to 2 so it will send a request to output port 1, output port 2 and output port 3.

Similarly, this will also send a request to output port 1, output port 2 and output port 3. This will send a request to output port 1, output port 3 and output port 4. So, all these requests have been sent. So, this is a request phase and then you have the grant phase. Now, as you can see here output port 1, it receives request from input port 1, input port 2 and input port 3. So, it can randomly select a grant, it can randomly select at which one of these input ports it would want to give a grant.

So, let us say that this output port 1 selects this second input port. So, it gives in a grant. It randomly selects one of the 3, then the output port 2 gives a grant to input port 1 and an output port 3 also let us say gives a grant to input port 1 and of course, this gives a grant to output port. So, this is like a grant phase.

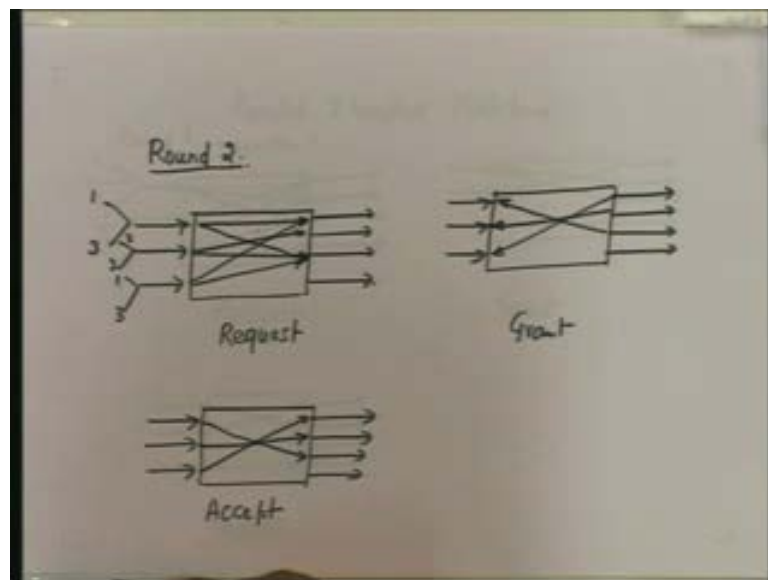
Now, in the accept phase, as you can see here, **so this is** so this is round 1 and this is iteration 1 as you can see, iteration 1. So, in the accept phase, this will of course accept this because this has only 1 request and this will also accept it, this has 1 request. Now, the input port 1

has received grants from 2 output ports - output port 2 and 3. So, it has to only accept 1. So, let us say, it accepts output ports 2. So, it gives an accept.

Now, as you can see that in the next the connect phase basically, input port 1 can send data to output port 2. So, these packets can be sent. Input port 2 can send data to output port 1, so this packet can be sent. Input port 3 can send data to output port 4 that means this packet can be sent and we have already, now this cannot be improved further because even though the output port 3 is busy, none of the input ports are free to send the packets to that output port and therefore we have already achieved a maximal match.

So, we have achieved a maximal match just in one iteration, in this case. Sometimes if maximal match is not achieved in one iteration, then what you can do is that you can mask the present matching and then try to achieve another match in the second iteration. So, you can actually improve upon the match that you have obtained. So, this is what is done in the round 1.

(Refer Slide Time: 47:40)

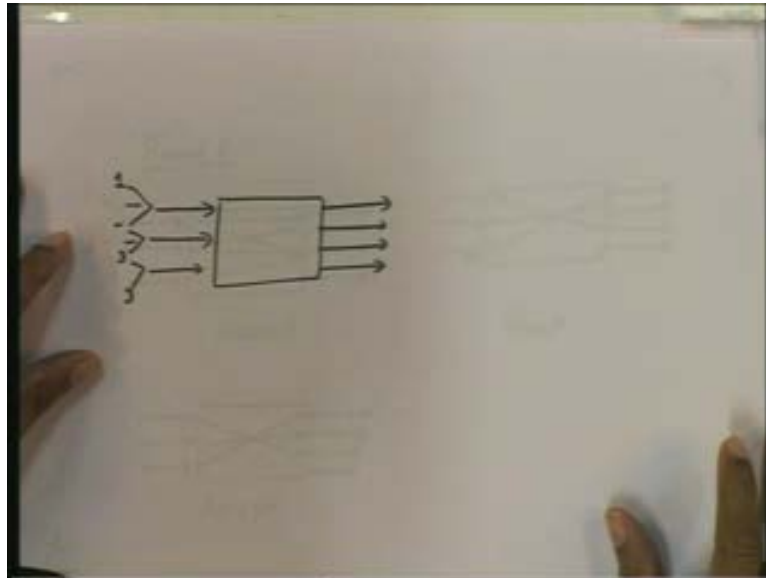


Now, in the round 2, as you can see here, the round 2 is again the request phase. Now, recall that in the round 1, if you see here, this packet 2 has already been sent. So, 1 and 3 are left really. So in the round, this 1 and 3 are left, 2 has already been sent. In the first round, this packet has already been sent. That is the packet 1 has been sent here. So, only 2 and 3 are left and here of course, 1 and 3 are left, the 4 has been sent. So, you will send a request to output port 1 and output port 3. This port will send a request to output port 2 and output port 3. This will send a request to output port 1 and output port 3 again. So, this will be the request phase.

Now, in the grant phase, now this output port has received the request from input port 1 and 2. So, it can randomly select, let us say it gives a grant to output port 3. Then this output port has received requests only from port 2. So, this can grant it to port 2 and port 3 has received request from port 1 and 3. So therefore, it can grant it to port 1 let us say, if it grants. So, this is the grant phase.

Accept phase of course, all the 3 input ports will accept it. So, 1 will accept it to output port 3, 2 will accept it to output port 2 and 3 will accept it to output port 1. So, this is the accept phase and so this is again the maximal match that we have already achieved here because in the one iteration itself, we have the achieved the maximal match, nothing further can be done to improve the things.

(Refer Slide Time: 50:53)

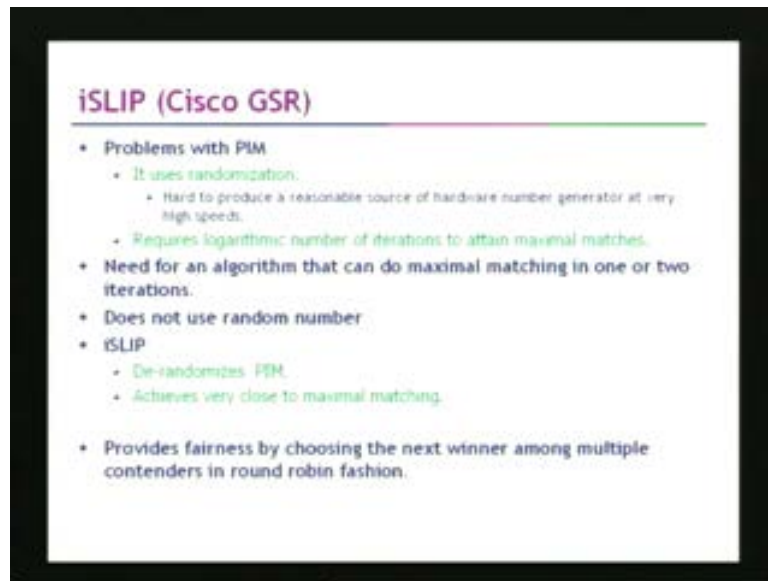


And now, if you really see the situation, after this grant, if you really see, then what is the situation how many packets have been transmitted after the round 2; if you see, then in this case, the input port as you can see here, has transmitted 3. So now, only packet 1 is left here, these have already been transmitted. Input port 2 if you see, then only again packet 3 is left here and in this case, 1 **has transmitted here** will transmit here. So, packet 3 is left here.

Now, in the round 2, then again it will go to the request grant and accept phase and the packets can be transmitted. So, as you can see, the parallel iterative matching actually has 3 phases: the request phase, in the request phase each input port makes a request to transmit packets to the output ports by having a separate input queue per output port at its input and once it convey the scheduling decision, the output ports actually randomly selects one of the input ports and gives a grant.

If an input port receives more than one grant then it randomly accepts one of these grants and then will start transmitting the packets.

(Refer Slide Time: 52:55)



So, the problem however with this parallel iterative matching algorithm is that it uses randomizations. Both in the grant phase as well in the accept phase, we use randomizations and it is hard to produce a reasonable source of hardware number generator at such high speeds, at such greater speeds which it requires.

Now, again it requires a logarithmic number of iterations to attend the maximal matches. Now therefore, we need an algorithm that can do the maximal matching in either 1 or 2 iterations and it does not use the random number.

So, the basic concept that we want is the same as the parallel iterative matching. But we would be looking for some scheme that does not use this kind of randomizations. If it can use some kind determinism and can also achieve a maximal match in 1 or 2 iterations, then that would be the best.

Now for this, a new algorithm that has been proposed is what is called as iSLIP. This iSLIP algorithms has been used in the Cisco series of routers also and can actually achieve the maximal match in not more than 2 or 3 iterations and therefore can achieve a very high performance. So, we will study the iSLIP algorithms in our next lecture.