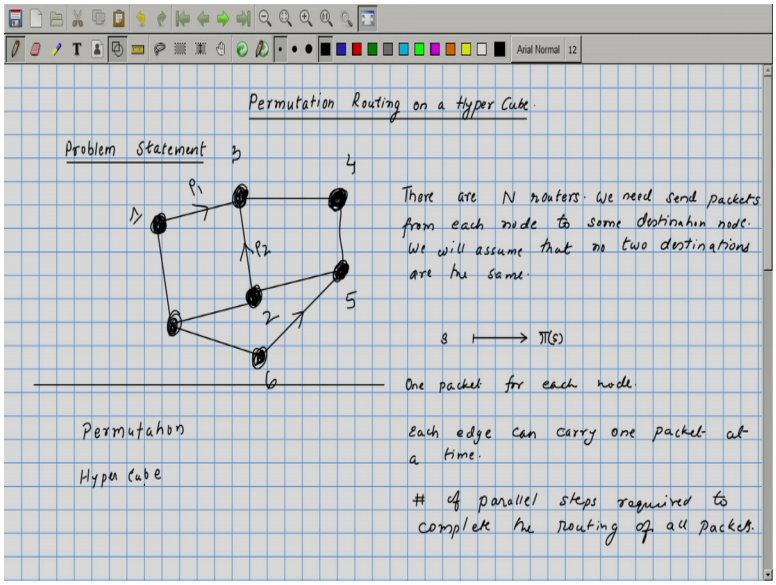
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 11
Permutation Routing on a Hypercube

In this lecture, we will learn about Permutation Routing on a Hypercube.

(Refer Slide Time: 00:35)



So, let us understand what is the problem. So, this kind of questions arise on when you are looking at parallel architectures. So, let us imagine that there are all these nodes which may we may view as let us say the processes in a parallel architecture.

So, imagine a system where there are it is a multi core architecture, where we have different cores, different processing units and they need to communicate amongst each other, ok. So, they might communicate in a particular manner, ok. So, all these edges represent you cannot directly send something from this particular CPU to this particular CPU, you have to choose a path, you have to route the information or the part of computation that you doing from one processor to another processor.

So, as an abstraction of these kind of problems what we will do is a following. We will assume that there are let us say capital N and say routers, ok. So, we will view each of these processing unit as a router. So, it can store information, it can store packets and it

can send packets to one of its neighbors, ok. So, the network topology is known to you. That means, which nodes are connected to or which routers are connected to which other routers that is already known to you and there are N routers we need to send packets from each node to some destination node. We will assume that no we will assume that no two destinations are the same. That is why this problem is called as permutation routing, ok.

So, let us say if s is a source, there is another node which we will call it as pi s, ok. So, it is a permutation of these 1 to N nodes, ok. So, each node you have a fixed destination and that destination is a unique destination for every node or every router in the graph, ok. So, this is what we want to do and when we want to do this, we have to fix the path along which this packet is going to be routed. There is we will assume that there is precisely one packet, one packet for each node and that packet needs to be sent to the corresponding destination.

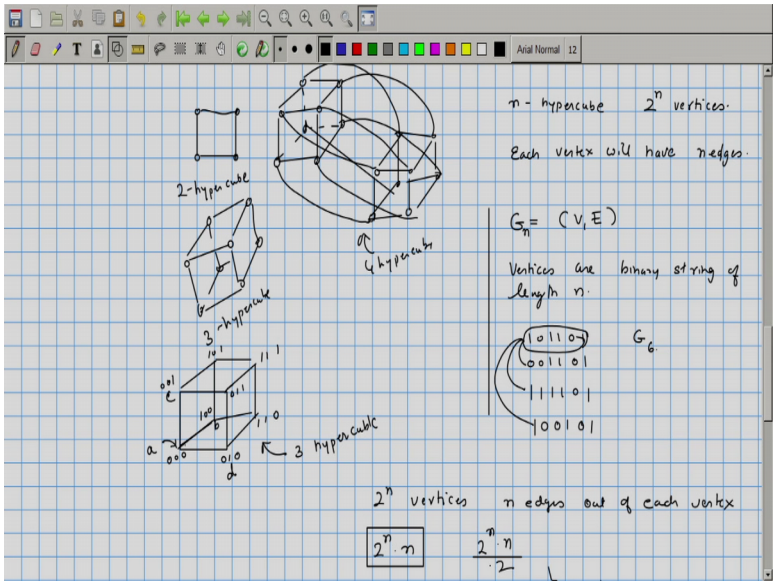
Now, the additional assumptions are as follows. Through each edge we can send at most one packet at a given time, ok. So, let us say there is a packet which comes from node 1 and node 2 to node 3, and both of them send it as at time unit 1, ok. If you want to send it to the node 4, you have to node 3 has to make a choice as to whether it is going to route packet P1 or the packet P2, ok.

So, maybe for the source one the destination was 5 and for the source two the destination was 4. Both those may have to use the path 3 4, in that case the packet we will say it waits or the router can queue the packets at that point and then, process it one after another. So, we will assume that each edge can carry one packet at a time, ok. So, we will say that there is some kind of a synchronization wherein at each clock tick one packet moves from let js say one node to the other, ok.

And when we measure the quality of this of a given routing protocol, we what we want to measure is number of parallel steps required to complete the routing of or the transfer of all packets, ok. So, if we have let us say packet at 1 and the packet at this particular node 6 and if both of them travel to, so one of them travel to node 3 and the other one travels to node 5. They can be carried out in the same time slot and that will be viewed as a parallel step.

So, each node may be carrying some particular packet at a given time. What we are interested in is a total number of parallel steps required to accomplish the complete routing or the complete transfer of all packets, ok. So, that is the problem statement and so that would essentially mean that we have understood what is permutation here. So, permutation because this for each source there is a unique destination and the destination is one among these nodes itself and the hyper cube in the problem statements needs to be now explained, ok.

(Refer Slide Time: 07:37)



So, let us understand what is a hyper cube. It is a generalization of a graph like this or a graph like. So, this is a cube hyper cube is a generalization of this. What does it mean to happen for it to be a generalization, ok? So, if you have let us say 2 cubes 2 copies of this ok, we could further connect them vertex by vertex, ok. So, all the corresponding pairs of vertex you connect that will be a. So, this a 2 uniform hyper cube 2 hyper cube. So, this can be viewed as 2 hyper cube whereas this is a 3 hyper cube and this is a 4 hyper cube.

So, we take two copies of 4 hyper cube connect let us say vertices in some orderly fashion. What you will get is a 5 hyper cube and so on. There is another way of looking at the same thing. So, n hyper cube will have 2 raise to n vertices and each vertex will have n edges, ok. How do you construct such an object ok? So, we can think of binary strings, we can construct a graph G is equal to V E where the vertices are binary strings of length n.

So, they 1 0 1 1 0 1 this will be one of the vertices of G 6, and two vertices are going to be connected by an edge is they differ exactly in 1 bit position. So, this is going to be connected to 0 0 1 1 0 1. It is also going to be connected to 1 1 1 1 0 1, then it is been reconnected 1 0 0 1 0 1 and so on, ok. So, you can flip one bit position. You can flip any other bit positions inside a particular vertex and you will get an additional you will get a new another vertex. Those two vertices are going to be connected. So, let us just look at this carefully with respect to n equals 3.

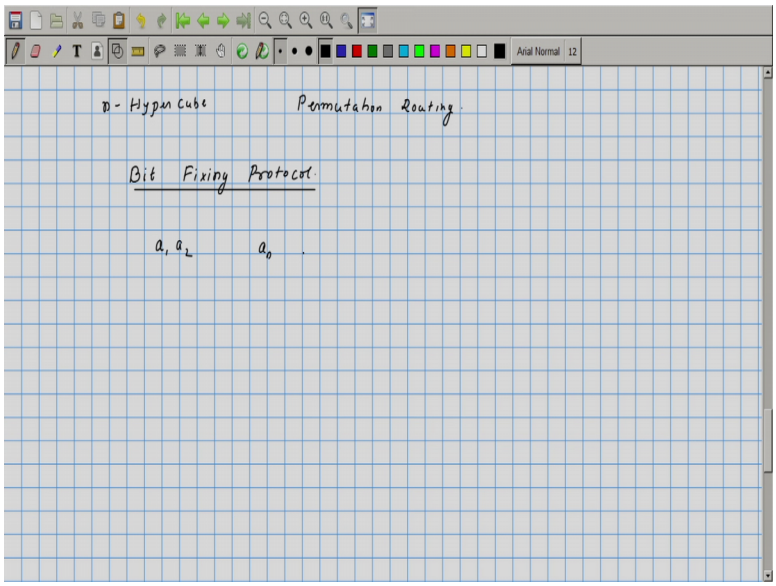
So, this is going to be if you think of this as the unit cube, this is the vertex 0 0 0 binary string 0 0 0. This we can think of as 0 1 0, this is 1 1 0, this we can see as 1 0 0, ok. So, the z coordinate of all these are 0 and this will be 0 0 1. Directly above this will be 1 0 1, this will be 1 1 1 and this will be 0 1 1.

And we can see that these two differ in exactly one position that is the first coordinate, these two differ in the last coordinate and these two differ in the second coordinate, ok. It is vortex a if you take b c d are the vertices which differ from the binary representation of the vertex a in precisely one position and this is true for every other vertex. So, this is called as the 3 hypercube and we can use this as the definition of the n dimensional hypercube or n hyper cube, ok. So, n dimensional hypercube will have 2 raise to n vertices and it will have n edges out of each vertex. For our purpose, we will imagine that the edges although we have drawing it as a single edge, there is an edge from say a to d and there is edge from d to a

So, when we look at the routing problem, we can send a packet from a to d and d to a at the same time in a in one parallel step. So, total number of edges in this graph would be 2 raise n times n, ok. If we did not have this condition that an edge can be used both in forward direction and backward direction, the total number of edges would have been 2 raise n into n by 2 ok, but since we here we are assuming that the forward direction edge is a different edge from the backward direction edge, we will say that there are means. So, the total number of edges will be 2 raise to n times n.

So, now we have the complete description of our problem. The network topology is that of a hypercube an n dimensional hypercube and we need to do permutation routing.

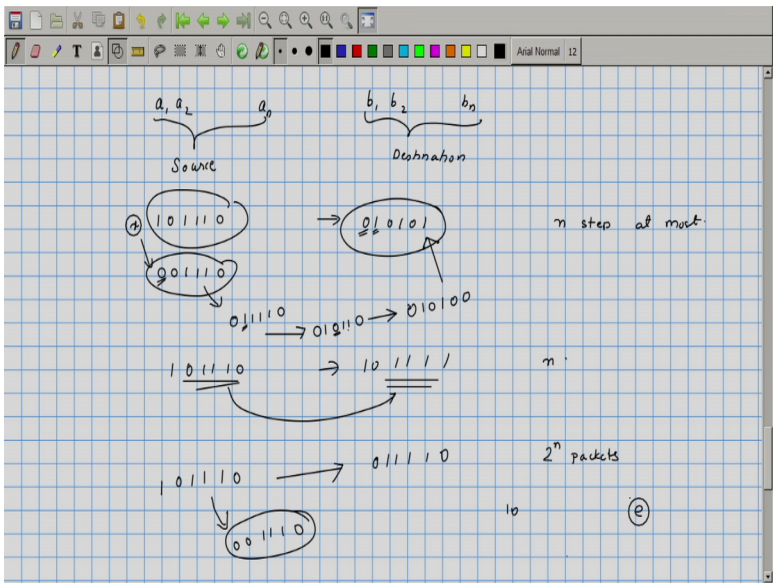
(Refer Slide Time: 14:15)



So, each packet needs to be sent to some destination packet and the destination is going to distinct for each of the sources. We need to do this in such that the total number of parallel steps required is minimised.

So, today we will first discuss a protocol called as bit fixing protocol. It is a simple protocol. Let us say we have one vertex a 1 a 2 a n and the packet at this particular source.

(Refer Slide Time: 15:17)



So, this is our source and our destination is b 1 b 2 b n. We may assume that these are a 1 a 2 a n. This is the that vertex the binary representation of the vertex b 1 b 2 b n is the binary representation of the destination.

Now, how do we send it? The bit fixing protocol essentially takes the packet and fixes 1 bit at a time. So, it sends it to that neighbour with 1 bit taken care off, ok. So, let us do this with an example. If you had this vertex 1 0 1 1 1 0 and this had to be sent to 0 0 1 1 1 0 ok, so the path will essentially consist of various vertices. So, this can of course we sent it. So, first we will fix the first bit. So, that will be 0 0 1 1 1 0, Ok sorry this is the bad example.

So, let us say so let us say that the input the source was 1 0 1 1 1 0 and the destination was 0 1 0 1 0 1. The way we will send it to this particular destination is by fixing one vertex at a time. So, let us say there is a packet, the packet is initially at this point, the packet is immediately sent to 0 0 1 1 1 0, ok. So, now the first bit position matches the next time unit it will send it to 0 1 1 1 1 0. That means, it is taken care of the second position; second position matches.

In the next instance it will go to 0 1 0 1 1 0, ok. So, the 3rd position is now taken care of. At the next instance the 4'th position already matches. So, we do not have to do anything so, but now you look at the 5'th position that is not matched. So, at the next instance you will send it to 0 1 0 1 0 0 and in the last step you will just send it to this.

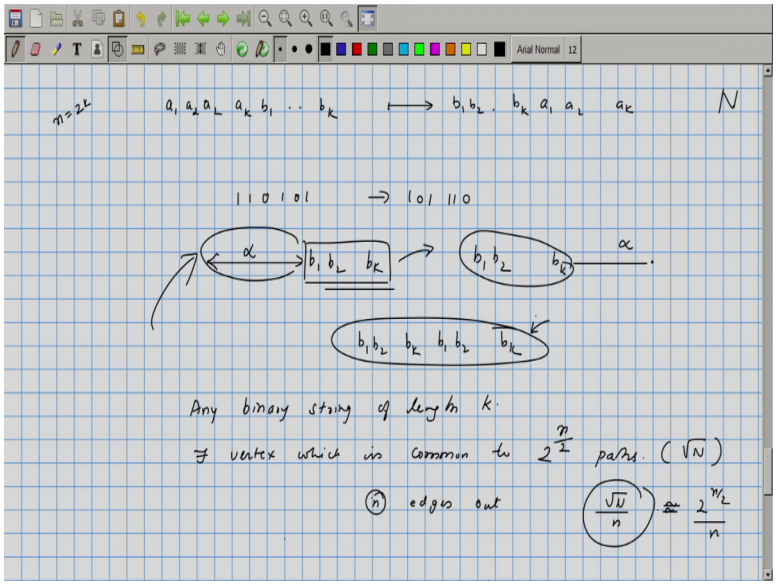
So, if x was the single packet which had to be routed, we could do it via these steps and that would take only n steps at most, ok. For example, if it is 1 0 1 1 1 0 to be sent to 1 0 1 1 1 1, the first step itself we can just send it to because these two the sources and destination, these nodes differ in just one bit position. So, you can essentially send the packet from that source to that destination in a single step.

So, in any case you do not require more than n steps. So, n steps is an upper bound on the number of steps required if you have to route one packet, but our question or the problem with we have to address is a little more complicated than that we have 2 raise to n packets starting at each source. So, in this case if we do this what could go wrong? So, this protocol is called the bit fixing protocol that is you fix 1 bit at a time. When we follow the bit fixing protocol for every source the path taken by every vertex is clear that is fixed, but there could be lot of paths intersecting with each other.

For example if x was to be sent from this I mean if this was the packet in this, there is one source and destination and let us say another packet say 1 0 1 1 1 0 is to be sent to 0 1 1 1 1 0, ok. Then at the first instance, this packet will go to the node 0 0 1 1 1 0 that is agreeing with this particular that these two nodes are the same and therefore, only one of those packets can use that particular edge. So, the question becomes when you have these multiple packets to be routed, so 2 raise to n packets to be sent to 2 raise to n different nodes maybe there are there is a particular node or a bottleneck edge through which lot of packets have to pass through.

In that case, let say if there is 10 packets which had to pass through one particular edge e, then since only one packet can be routed at one unit of time at the edge e itself, there will be a delay of 10 units. What we want to know is how bad can this delay be ok? If we can say that if we can come up with the protocol such that these delays are minimised if we can guarantee that the maximum delay that should be there for any particular packet is going to be a small, then we can say that our protocol is a good protocol, but unfortunately there are inputs on which bit fixing protocol could behave in a particularly poor fashion. So, let us see a particular case.

(Refer Slide Time: 21:39)



So, let us look at an input a 1 to b k. So, we will think of it as a, we will say n is equal to 2 k ok, a 1 to a k and b 1 to b k. So, any source can be viewed as a 1 to a k followed by b 1 to b k. Suppose a destination the corresponding destination is fixed as b 1 b 2 b k

followed by a 1 a 2 a k, so suppose this is the permutation that is given if the node is of the form 1 1 0 1 0 1 that has to be sent to 1 0 1 1 1 0, ok. So, this basically fixes the permutation which source is to be the packet from which source is to be sent to which destination that is clear. Now, is there any particular edge which contains lot of paths? So, suppose you look at any particular source of the form let us say alpha followed by b 1 b 2 b k, they will have to go to b 1 b 2 b k. The final destination of any such node is followed by alpha.

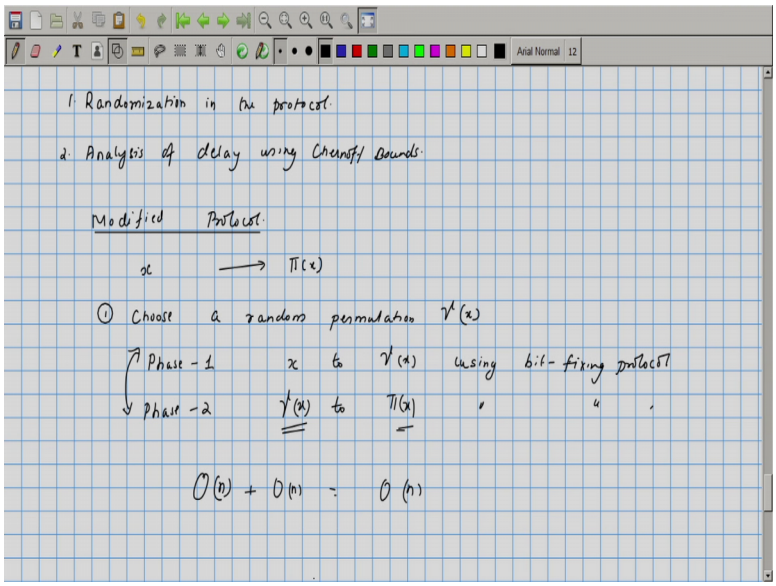
But since we are fixing the bit fixing proto, we are following the bit fixing protocol the following node that is b 1 b 2 b k followed by b 1 b 2 b k. This will be an intermediary node for any such path. So, any path from alpha b 1 b 2 b k to b 1 b 2 b k followed by alpha in that this particular node or this particular vertex will surely have to be present because this had to be we are following bit fixing protocols. So, the first part should become b 1 b 2 b k and the last part automatically is and that by definition of the problem it is b 1 b 2 b k.

Therefore, all these nodes have to pass through b 1 b 2 b k followed by b 1 b 2 b k, ok. So, this node is going to route many packets, therefore there has to be some congestion at that particular node and this there are how many such nodes. Well alpha could be any binary string of length k, ok. So, we know that there exists an edge or there exist a vertex through which there exist a vertex which is common to 2 raise to n by two paths, ok. So, this is roughly when if they were total an n number of nodes, those are vertex which is common to route n paths.

So, there could be a, so because there are route n packets coming at a particular point and there at most let us say log n or log capital N or n edges out ok, so at any instance you can send at most route n by mean let say you can send at most n packets out. So, route n by n at least this much delay could be there if the input permutation was this, ok. So, this is approximately 2 raise to n by 2 divided by n. It is equal to 2 raise to n by 2 divided by n, ok. So, this is a particularly bad input for bit fixing protocol.

And if you are if you wanted to do permutation routing on a hypercube bit, fixing could perform poorly on some particular inputs. What is the way out? So, we will essentially use randomization to take care of this, ok.

(Refer Slide Time: 26:27)



So, there are two key ideas. First is Randomization in the protocol, second is the Analysis of delay using Chernoff Bounds. We are going to assume couple of things during this analysis. So, we argued that once you fix the path, the packet takes that particular path. So, that is one part of the protocol.

Now, if there are multiple packets coming to same to a given vertex and it want to use this and all those packets want to use a certain edge during, its during next time slot, we will assume that in all those cases the I mean we will just toss a coin and decide which packet goes through, ok. So, in particular what we will insist is if there are packets at a particular vertex which won to use a certain edge, that edge is certainly used during the next particular time slot, ok. So, there are packets in the queue, the packet will be occupied during the next time slot, ok.

So, let us just look at the modified protocol. So, we had we had all these sources and the corresponding destinations. Now, what we will do is instead of the destinations given by the input specification, we will choose a random permutation. So, choose a random permutation let say let us call it as gamma x, ok. So, gamma x gives the image of the vertex x under this random permutation. So, then what we will do is, so during phase 1 we will send x to gamma x using bit fixing protocol and during phase 2 we will send.

So, we will again use bit fixing to send packet from. So, when I say x to gamma x there is a packet at the sources x which is being sent to gamma x and that packet from gamma

x we will send to pi x again using bit fixing protocol. So, this we will call as phase 1 and this is the second part we will call as phase 2, ok.

Now, how does this help? The first thing is there were when we analyse the worst case running time of bit fixing protocol, we said that there are particular input output combinations which could be fatal for the bit fixing protocol, but if we choose randomly a destination that destination I mean if you if you use if you choose a random destination for each source, that particular choice or that particular permutation can probably be routed using bit fixing protocol without incurring much penalty.

So, we have to do that twice and the second part is exactly similar to the first because both phases are in some sense the analysis will be identical. So, we just have to do it for one phase because whether you send from x to gamma x, the time taken for that it is essentially similar when you follow the protocol from gamma x to pi x, but we will have to make an assumption that only after the first phase is completely over, we will begin the second phase. This assumption we can later on relax.

So, what we will show is this phase, phase 1 will require O n time to complete. That means, all packets would reach its destination in O of n time may be we will wait for some additional time. The chances that there is some packet which is not completed its travel from its source to the destination gamma x that we will show is very small and 1. So, after that we will begin our phase 2 which is again another instance of random permutation routing. So, the total time will be O n plus O n which is again O n, ok. So, that is the bit fixing, the modified bit fixing protocol. We will continue with the analysis of this in the next lecture.