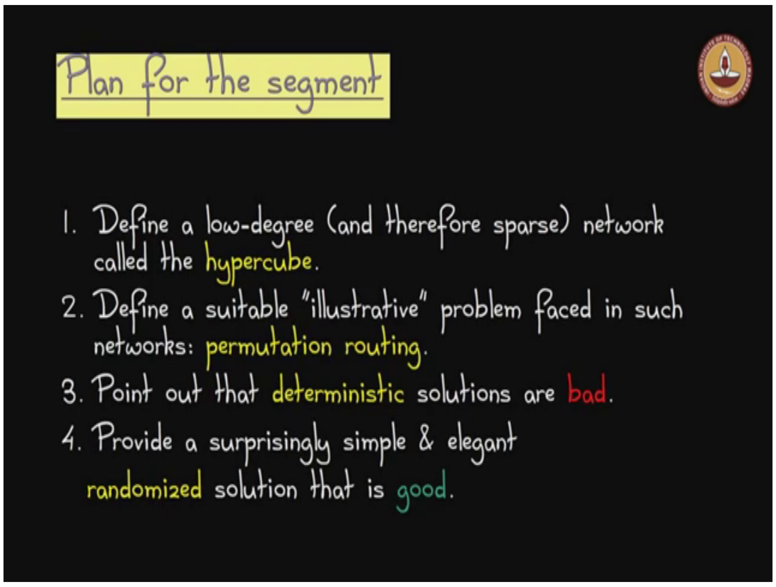
Probability and Computing

Prof. John Augustine
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Module - 04
Applications of Tail Bounds
Lecture – 22
Routing in Sparse Networks

(Refer Slide Time: 00:33)



Plan for the segment

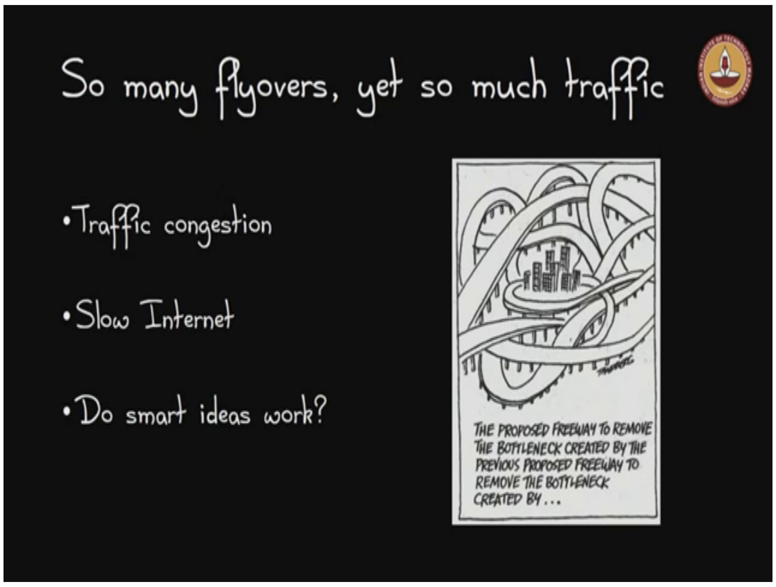
1. Define a low-degree (and therefore sparse) network called the hypercube.
2. Define a suitable "illustrative" problem faced in such networks: permutation routing.
3. Point out that deterministic solutions are bad.
4. Provide a surprisingly simple & elegant randomized solution that is good.

So, we have seen about a couple of applications of tail bounds in this course in this module. And now we are going to see something that is actually quite famous, we are going to talk about routing in sparse networks. So, here is the plan for this segment first of all we are going to define a low degree network. So, if you think about networks in data centers and things like that, you need low degrees sparse networks because in order to maintain the computers in the data centers.

So, that they can I mean you still want it to be low diameter you want the latencies to be small. But at the same time you do not want very high degree or very high number of edges dense graphs because then you will have to invest on a lot of wire. So, a hypercube is one such example that is low degree and low diameter ok.

So, we are going to define a hypercube, and then we are going to take an illustrative problem the permutation routing problem will define that. And we will point out that deterministic algorithms are bad for solving this permutation routing. We would not prove this, but we will just point that out. And then in this module, in this segment, we will discuss an a very, very simple and very neat algorithm; it is a little bit counterintuitive possibly, but very nice and elegant algorithm. And in the next segment, we will analyze its properties ok.

(Refer Slide Time: 01:53)



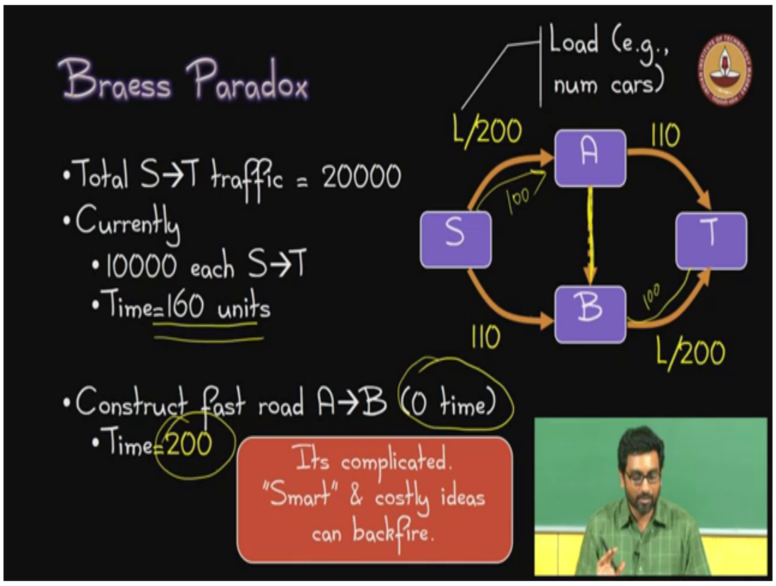
So many flyovers, yet so much traffic

- Traffic congestion
- Slow Internet
- Do smart ideas work?

THE PROPOSED FREEWAY TO REMOVE THE BOTTLENECK CREATED BY THE PREVIOUS PROPOSED FREEWAY TO REMOVE THE BOTTLENECK CREATED BY...

Just to motivate this is this problem is set in the context of congested networks, they are all used as you know traffic congestion, slow internet and stuff like that. And sometimes when you see how the flyovers are built in china you wonder you know what thought went into that.

(Refer Slide Time: 02:15)



Braess Paradox

- Total S→T traffic = 20000
- Currently
  - 10000 each S→T
  - Time=160 units
- Construct fast road A→B (0 time)
  - Time=200

Its complicated. "Smart" & costly ideas can backfire.

So, yeah so this is let us actually look at some science behind this. And this is something called a Braess paradox. So, let me just briefly talk about this. So, this paradox kind of tells you about you know smart ideas that you might have, but may not and smart in quotes that may not really lead to improvements. Let us let us see what happens over here.

So, you have this two cities S and T. And you have two ways to go from S to T. And let us say there are some 20,000 cars that want to go from S to T ok. And these routes have two segments each for now ignore the dotted route. This and the if you look at the top route, there is a segment that has this L over 200 this means the time it takes for a car to go from S to A is the is given by the load the number of cars that are going to use that route divided by 200 ok. So, this is you can think of it as a small road, where if more number of cars go that, then there is more congestion and therefore, this more delay ok.

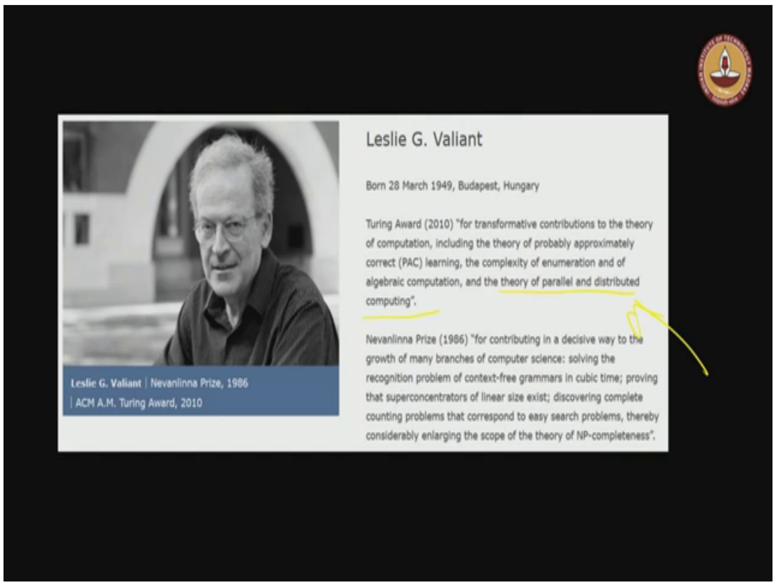
But then there the other segment from A to T if you notice has a fixed time. So, this is like a highway plenty of lanes. So, the your latency does not depend on the number of cars mean you have a similar situation in the bottom route except the constant time segment is first followed by the segment that in which the delay depends on the load. And, so now our example we have 20,000 cars and it is not hard to see that with without the middle road that connects A and B ok, ignore that for the moment.

You expect people to you know say little game theory here, people will realize that they you know if one side gets too crowded, then they will start using the other side and so the cars will split roughly into 10,000 each ok. And it is not hard to see that the latency here is going to be 160 minutes or whatever units of time you want to take ok. Why, because there is 20,000, 10,000 cars each, so if you divide that by 200, you are going to get something like 50 plus 110 units for this segment. So, you have a total of 160 units.

But let us say what happen let us see what happens if you build this very high speed corridor that connects A and B ok. This is very fast and you know really high investment road that is built between A and B takes zero time to go from A to B well (Refer Time: 05:19) ok. But then let us see what happens you will start to realize that people will start going from S, so they what is what will be the natural way people will try to gain the system, they will go from S to A, then try to use this very fast connection. And then from A to B, and then go from B to T and what will happen is this will have a snowballing effect and ultimately everybody will want to do that.

And if everybody does that then we are in a bad situation because this segment will take 20,000 divided by 200. So, that will take about 100 units. This is 0 and this will take another 100 units. So, it becomes the Nash equilibrium if you will becomes this bad situation ok. So, this is just to illustrate that congestion is important and funny things can happen. So, we have to be very careful and just being so called smart it is not necessarily the right thing to do ok.

(Refer Slide Time: 06:24)



Leslie G. Valiant

Born 28 March 1949, Budapest, Hungary

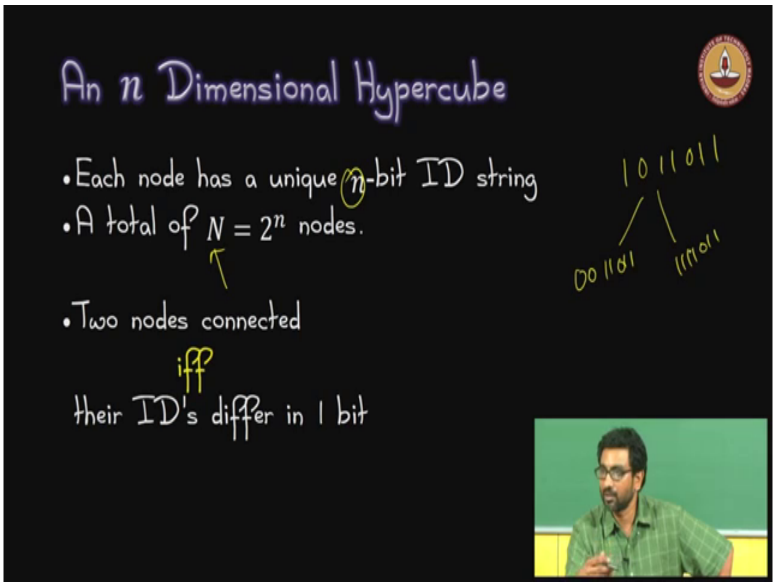
Turing Award (2010) "for transformative contributions to the theory of computation, including the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and the theory of parallel and distributed computing".

Nevanlinna Prize (1986) "for contributing in a decisive way to the growth of many branches of computer science: solving the recognition problem of context-free grammars in cubic time; proving that superconcentrators of linear size exist; discovering complete counting problems that correspond to easy search problems, thereby considerably enlarging the scope of the theory of NP-completeness".

Leslie G. Valiant | Nevanlinna Prize, 1986 | ACM A.M. Turing Award, 2010

So, this brings us to a contribution by a recent Turing award winner Lesley Valiant known for many things, but we are going to be particularly focusing on the theory of parallel and distributed computing. In particular one of his famous contributions here is what we are going to talk about today ok.

(Refer Slide Time: 06:51)



An n Dimensional Hypercube

- Each node has a unique n-bit ID string
- A total of $N = 2^n$ nodes.
- Two nodes connected

iff

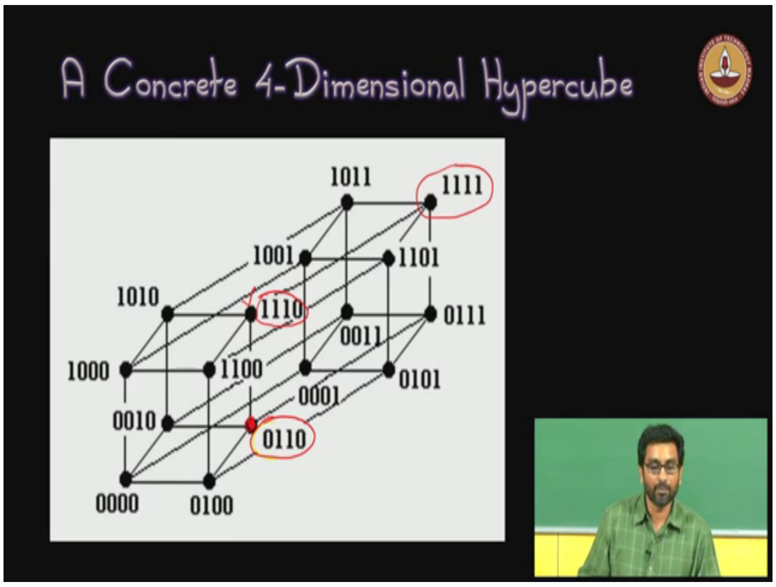
their ID's differ in 1 bit

1011011
0011011
1111011

So, first to set it up we have an n dimensional hypercube. What is that? Well, basically each node in this hypercube is denoted by an n-bit id ok. And here the number of bits is small n. And so the total number of nodes in this n dimensional hypercube is uppercase

N. And if you win are two nodes connected or not connected, they are connected if and only if their id is differ by exactly 1 bit ok. So, if you have 1011011, it will be connected to one, two, three, four, five, six, seven other nodes. So, for example, you be connected to 001101, why because there if they differ the first bit. And it will also be connected to 1111011, because they differ in the second bit and so on.

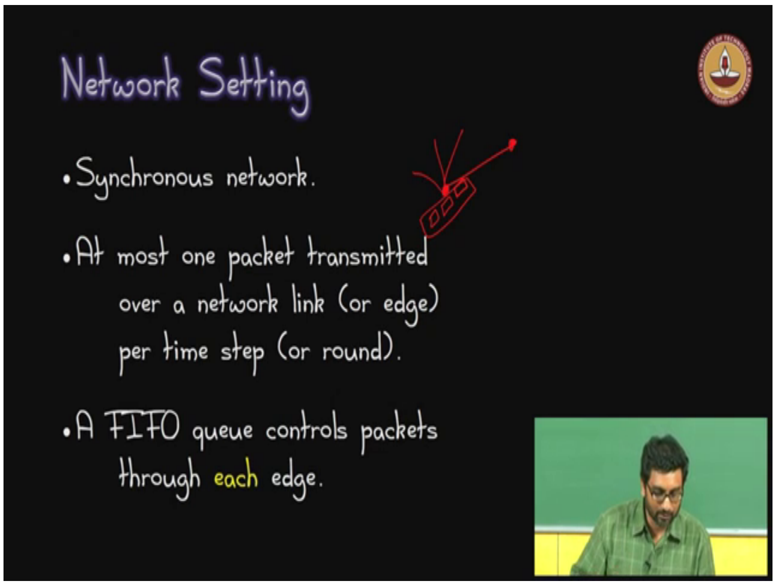
(Refer Slide Time: 07:59)



So, now let us actually look at how that hypercube looks and it is this is an example of a four-dimensional hypercube. So, notice that each node in this in this hypercube is denoted by a four bit string that is its id. And let us see this, this, this is this node right. So, who are its neighbors, it is connected to this node. And if you notice they differ in the first bit and so that is why they are connected. And let us take for example, this node, they differ in two bits, so they are not connected right.

So, this is how of an n-dimensional hypercube looks like ok. So, it is it gets a little hard to visualize as the dimensionality increases, but for now the only thing that you intuition that you need is two nodes are connected as long as their id is different 1 bit.

(Refer Slide Time: 09:03)



Network Setting

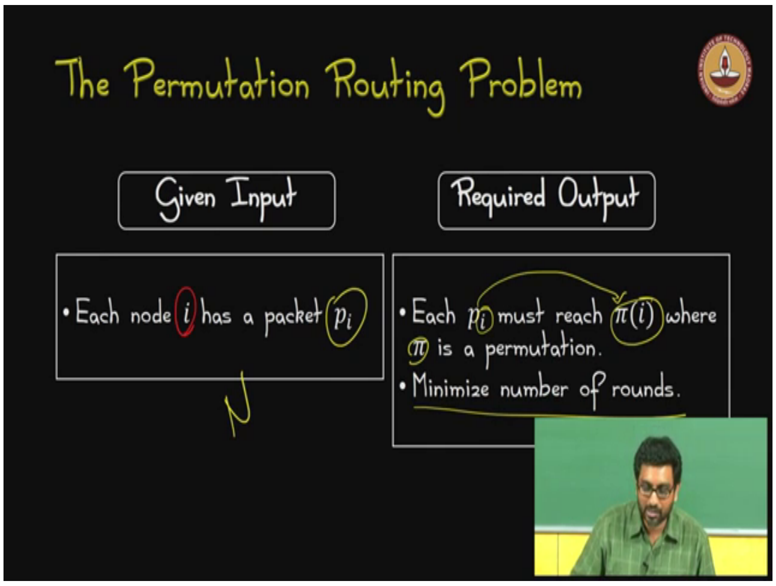
- Synchronous network.
- At most one packet transmitted over a network link (or edge) per time step (or round).
- A FIFO queue controls packets through each edge.

So, this is the network topology or structure. And we want to also understand some of the rules by which this network place. So, it is a synchronous network which means everybody has a global clock, it is like a conductor saying this is time step one and everybody does something, then the conductor says time step two and somebody said as in the next step occurs and so on.

So, everybody follows the same clock ok. And if you look at any edge in the network at most one packet of information can go through that network per time step ok. This is this is where congestion issues start to show up this is the restriction, but it is a realistic restriction and so, if this restriction was not there this problem would not be any interesting what we are whatever we are going to study ok.

And what so now you may ask if only one can go and an a bunch of packets are waiting over here how do you handle that well just put them in a first in first out queue ok. This first in first out queue is there is one queue for every edge incident at this vertex. So, for every edge, there are a bunch of packets waiting to go along that edge. So, those packets will be put in a queue, and they will be sent out one by one along that edge. So, for each vertex, they will actually be n queues in that vertex.

(Refer Slide Time: 10:34)



The Permutation Routing Problem

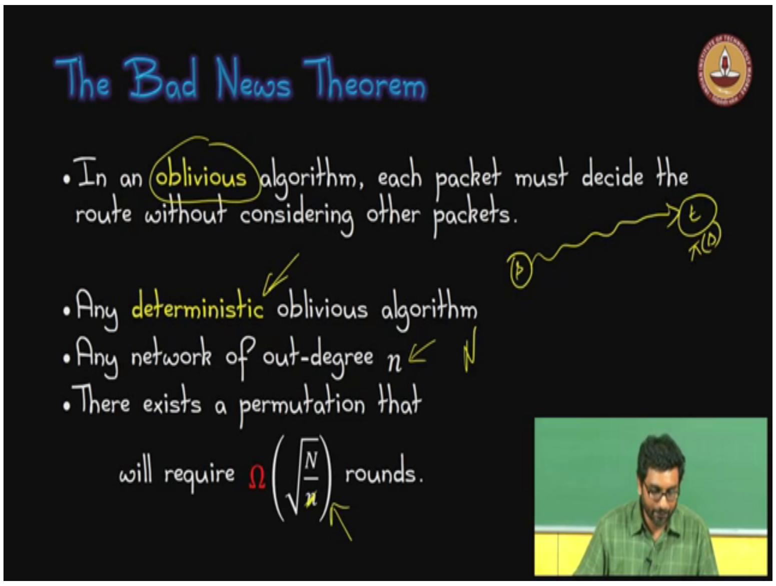
Given Input	Required Output
• Each node i has a packet p_i	• Each p_i must reach $\pi(i)$ where π is a permutation. • Minimize number of rounds.

N

So, this is the permutation routing problem this is the problem that we want to solve in this hypercube network setting. Each node has a packet each node i has a packet p sub i ok. And what do you want out of this at the end p sub i each p sub i should which is starting at node I much must reach another node pi i and this pi is a permutation. So, basically everyone will start with the packet, and everyone will end with a corresponding pi packet all right. Pi is a permutation. You want to the goal is to minimize the number of rounds required to get this permutation routing to be completed ok.

And you can sort of see the issue here that there are if each node has a packet, there are capital N number of packets, they are all kind of jostling with each other in order to reach their final destination. Remember along each edge only one packet getting go per time steps. So, if there is too much congestion that takes place you have to wait along the way its several steps and so the number of rounds totally needed in order for all the packets to reach their corresponding destinations can become uncomfortably large, and that is what you want to try and minimize.

(Refer Slide Time: 12:01)


The Bad News Theorem

- In an oblivious algorithm, each packet must decide the route without considering other packets.
- Any deterministic oblivious algorithm
- Any network of out-degree n
- There exists a permutation that will require $\Omega\left(\sqrt{\frac{N}{n}}\right)$ rounds.

So, let us start with the bad news first. The bad news is that we are interested in oblivious algorithms. So, oblivious algorithms means these are this is a very natural class of algorithm so in the distributed setting. If you have a packet that is has to go from S to some T, so T is basically pi of as the permutation point at which it has to go. This route that it takes should only depend on S and T ok.

This is this makes sense because in a distributed setting you do not know what other packets are doing you do not know what others packets source and destination pairs are and things like that. Those should not be part of your concern. If you are if you are a packet going from S to T your path should only depend on S to T, and this is a very natural restriction. So, we are only interested in oblivious algorithms.

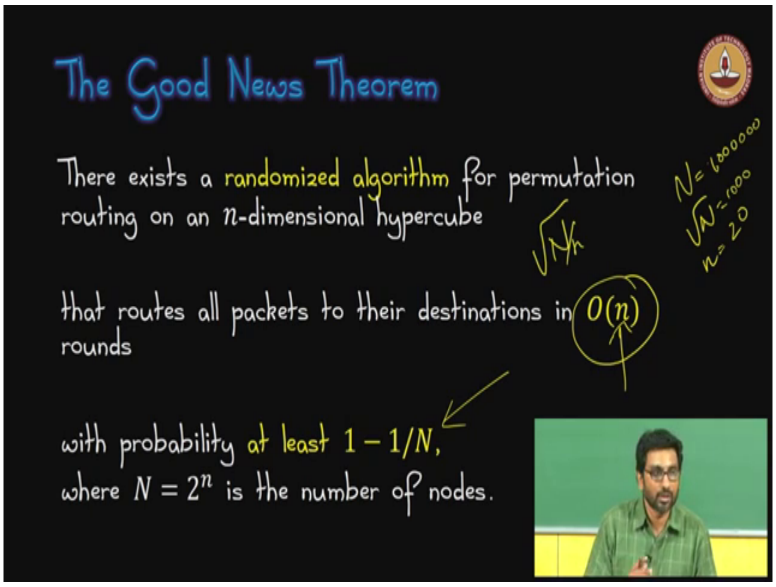
And let us focus first on deterministic algorithms and let me state the bad news. If, so let us fix any deterministic. Algorithm you come up with any algorithm that you can think of. And any network of out degree n, so the degree is bounded by this lower case n. And of course, there are total of upper case n number of nodes they always exists a permutation basically this pi that defines for each starting vertex where the packet should end, there always exists a permutation which will force the algorithm to take at least omega of uppercase I mean square root of uppercase N over lowercase n.

So, you can ignore this is for understanding purposes; the lowercase n is a small quantity its log of the uppercase n. But what matters mostly that you need to focus your attention on this its dependent on the square root of uppercase N ok, so that can be quite large ok. So, we want to try and minimize that ok. This is uncomfortably large.

(Refer Slide Time: 14:12)

So, naturally we want to try to see if there is some randomization we can do.

(Refer Slide Time: 14:17)



The Good News Theorem

There exists a randomized algorithm for permutation routing on an n-dimensional hypercube

$\sqrt{N/n}$

that routes all packets to their destinations in $\boxed{O(n)}$ rounds

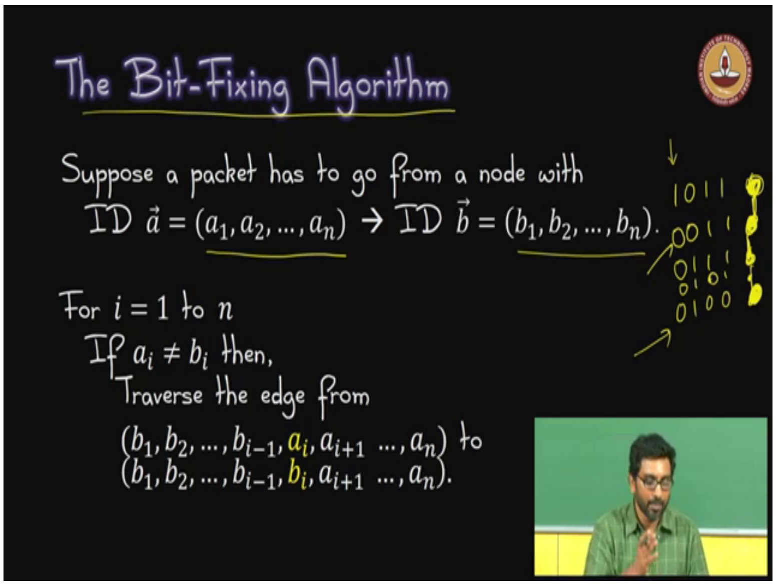
with probability at least $1 - 1/N$, where $N = 2^n$ is the number of nodes.

$N = 1000000$
$\sqrt{N} = 1000$
$n = 20$

So, here is what my good news will be. So, there exists a randomized algorithm for permutation routing on the n-dimensional hypercube, it routes all the packets to their destination in O of n a lowercase n number of rounds ok. What used to be square root of uppercase N divided by n is now become just log of that. So, if you if you want to roughly understand what the difference is, so let us say you have n capital N equal to 1 million ok, square root of N is 1000 ok.

But lowercase n is just the number of bits needed to represent 1 million, so that should be like I do not know 20 something; like that so small. So, you see that this is a significant improvement of course, we are ignoring constants here, but constant should not play too much of a role here ok. And this is guaranteed with probability 1 minus 1 over uppercase Nm so that is with high probability. So, this is the good news.

(Refer Slide Time: 15:36)



The Bit-Fixing Algorithm

Suppose a packet has to go from a node with ID $\vec{a} = (a_1, a_2, \ldots, a_n) \rightarrow$ ID $\vec{b} = (b_1, b_2, \ldots, b_n)$.

```
For $i = 1$ to $n$
    If $a_i \neq b_i$ then,
        Traverse the edge from
        $(b_1, b_2, \ldots, b_{i-1}, a_i, a_{i+1} \ldots, a_n)$ to
        $(b_1, b_2, \ldots, b_{i-1}, b_i, a_{i+1} \ldots, a_n)$.
```

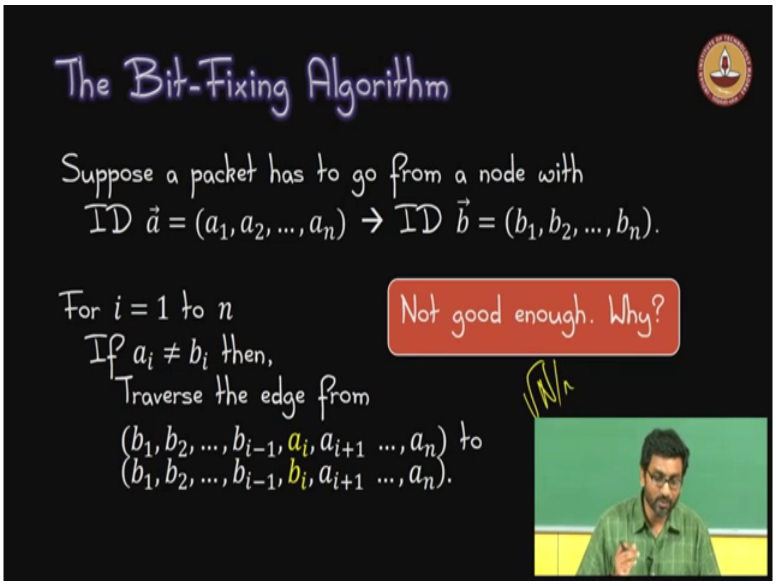
So, let me give the an algorithm ok. This is called the bit-fixing algorithm a very natural algorithm to route the packets. So, remember each packet is starting from a node and so let us look at a particular packet it is starting at a node that has this address a 1 through a n. And it has its destination is this location denoted by this address b 1 through b n ok. So, let us work through an example a simple example let us say is 1011, and it has to let us say its destination is 0100. These are two nodes right in the in the network, it should probably in the nodes ok.

So, from here, there will be four neighbors, but what the bit-fixing algorithm says is you start from let us let us count the bits from left to right. So, let us start from the first bit are the two bits different from where the packet is now and where it finally has to go yes. So, then you fix that bit that bit alone change you means that you have a neighbor where that bit alone is different right. So, you go to that neighbor ok. So, you go to that node. So, this neighbor of that first node where the first bit alone is different.

Second bit again is different between where it currently is and where it finally, has to go. So, then you again fix that bit. So, it becomes 01 1 1. The third bit again needs to be also pretty much all of them need to be fixed in this case. So, 0, it goes to 01001 and then finally, you fix the last bit you get to the destination ok. So, that in this manner you go from one node to its neighbor by just fixing bits from left to right ok. Why is this not good enough, what sort of an algorithm is this?

Student: Deterministic, deterministic.

(Refer Slide Time: 17:51)



The Bit-Fixing Algorithm

Suppose a packet has to go from a node with ID $\vec{a} = (a_1, a_2, \ldots, a_n) \rightarrow$ ID $\vec{b} = (b_1, b_2, \ldots, b_n)$.

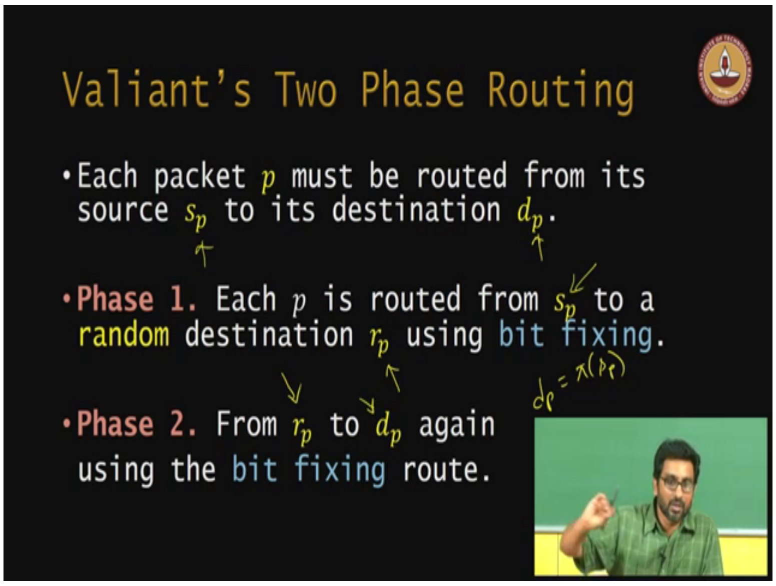
```
For $i = 1$ to $n$
  If $a_i \neq b_i$ then,
    Traverse the edge from
    $(b_1, b_2, \ldots, b_{i-1}, a_i, a_{i+1}, \ldots, a_n)$ to
    $(b_1, b_2, \ldots, b_{i-1}, b_i, a_{i+1}, \ldots, a_n)$.
```

Not good enough. Why?

$\sqrt{N}/n$

It is a deterministic algorithm, or what do we know about deterministic algorithm. So, it is always going to take square root of upper N upper case n divided by n number of rounds. So, it is a it is a very nice looking algorithm very simple clean algorithm, but it is not good enough why because its deterministic and as soon as it becomes a deterministic algorithm you have this bad news scenario ok. This of course, keeping just to keep in mind this is done by every packet because every packet has a starting location and a destination location.

(Refer Slide Time: 18:34)



Valiant's Two Phase Routing

- Each packet p must be routed from its source s_p to its destination d_p.
- **Phase 1.** Each p is routed from s_p to a random destination r_p using bit fixing.
- **Phase 2.** From r_p to d_p again using the bit fixing route.

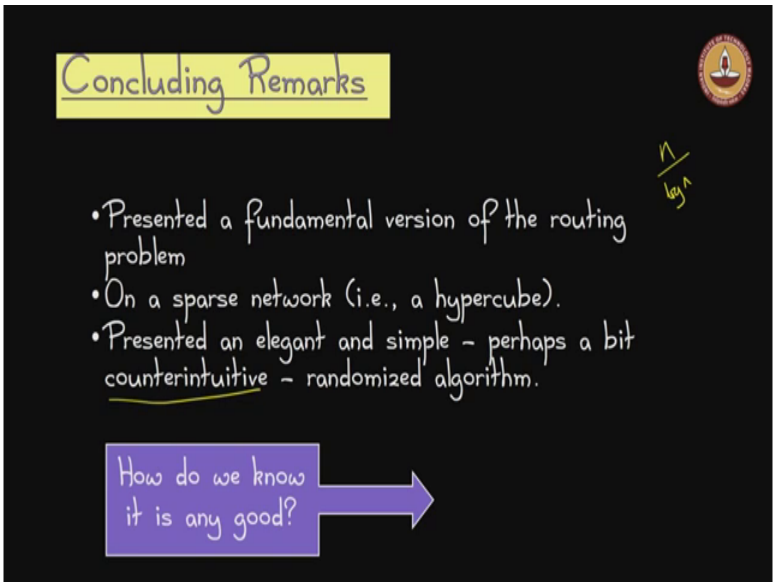
So, how do we salvage the situation; again now we have to insert some randomization into this right. So, how did valiant achieve this it is a very very nice algorithm it just basically takes advantage of bit fixing, but sets it up in a randomized fashion, but sets it up in two phases and this is the beauty of this algorithm. So, recall that each packet p must be routed from its source to its destination d p, s p to d p.

Here is what you do so that in the first phase what does p do this packet p, its routed from its source to a random destination completely randomly chosen destination using bit-fixing. Because now source is well defined it is it is already where the packet is starting from you just pick a random n bit string that tells you the random destination that this packet has to go to. And do you know how to go there yes you do not want to bit fix your way to that location.

And then phase two from that random location you make your way to the final destination. So, it is a very very simple algorithm; it is just two phases of the bit fixing. So, each packet just starts from its current location and goes to a random location from that random location it goes to the final destination, and both of them work via bit-fixing. So, bit-fixing itself then work, but it we have salvaged ok. It is it is a permutation routing right.

So, s p so basically d p is pi of s p each face is not a permutation routing you are absolutely right. So, there can be collisions on the receiving end. So, multiple nodes could choose the same r p yeah that is the possible possibility that is completely ok. So, that depends on the local memory of the of each node and that there is usually some because we have any way have to maintain the queues and everything in that local machine right in that machine. So, that is not the problem. The problem is in the as when you go through the links you can only send one by one.

(Refer Slide Time: 20:36)



Concluding Remarks

$\frac{n}{\log n}$

- Presented a fundamental version of the routing problem
- On a sparse network (i.e., a hypercube).
- Presented an elegant and simple - perhaps a bit counterintuitive - randomized algorithm.

How do we know it is any good?

So, this really is the algorithm any questions on the algorithms ok? So, hopefully everything is clear its basically let us just focus on the analysis in the next segment in this segment we have just basically seen a very simple algorithm and at least the first time I saw I saw it, it was counterintuitive to me. In some sense its intuitive in some sense, it is counter intuitive. So, so it depends there is a subtlety in here

Student: (Refer Time: 21:01).

Yeah, there is a subtlety here if you define the algorithm so that face one has to complete before phase two starts then they would have. Even if one reached before the other it both will wait in the destination until its time for phase two to start and then the phase two will start, but then you can you can actually the other version where the.

Student: Perfect.

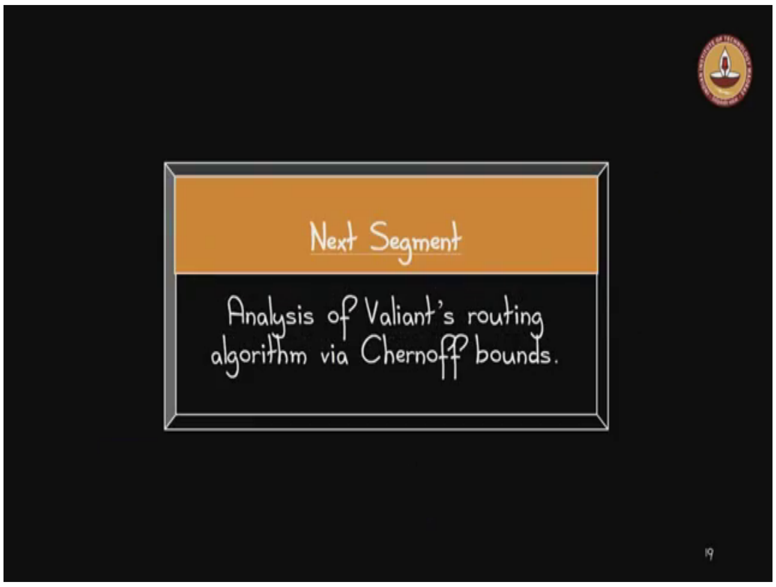
You do not wait is also perfectly legal.

Student: Perfectly legal.

It is cleaner, but even if there are collisions, it is not a problem. There will be collisions there was surely be collisions they will spread out, but we will actually be studying how to analyze the exact issue that you are talking about. There will be lots of nodes good fraction of them of the nodes will not receive any token at all any packet at all which means that some a good fraction of them will have collisions.

And in fact you can also analyze how many packets will end up in I mean the maximum number of packets that will end up in some node and that will turn out is it is actually something like n in this case lower case n over log n roughly speaking. And we will analyze these things in forthcoming lectures ok. So, this is basically what you are alluding to is balls and bins and alley.

(Refer Slide Time: 22:33)



Next Segment

Analysis of Valiant's routing algorithm via Chernoff bounds.

19

So, of course, we have to ask ourselves how do we establish that this is a good algorithm, it is a very clean algorithm neat algorithm, but that is what we are going to see in the next segment ok.