Broadband Networks

Prof. Karandikar

Electrical Engineering Department

Indian Institute of Technology, Bombay

Lecture - 11

Virtual Time in Scheduling

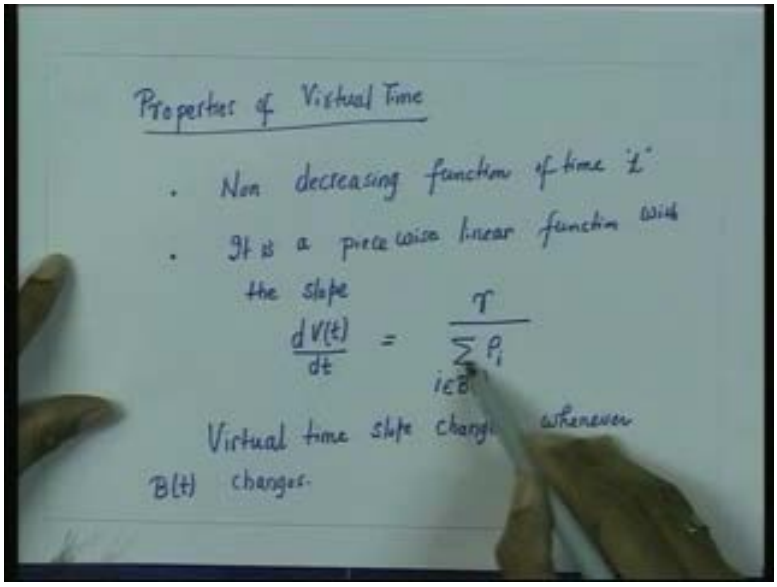
So, we were discussing about the packet scheduling algorithms for providing fairness as well as quality of service guarantees in internet. So, what we have seen is that a fluid flow fair queuing or generalized processor sharing algorithm guarantees fairness to the traffic queues in the sense of maximum fairness. But as we know that a fluid flow fair queuing operates only on the traffic arrivals which are considered as a fluid and in practice, the traffic arrival will be in packetized forms; so then we were looking for the packetized versions of the fair queuing algorithms.

So, we saw various versions. First is the round robin scheduling algorithms or the weighted round robin scheduling algorithm; then, we also saw the deficit round robin algorithms. We have seen one of the disadvantages of the weighted round robin scheduling algorithm is that it is unfair first of all over shorter time scales but more than that it requires knowledge of the mean packet length in advance if you have to make the weighted round robin scheduling algorithms to be fair. But even then it is unfair; it may be unfair over shorter time scales.

The disadvantage of having to know the mean packet size in advance was addressed in the deficit round robin by maintaining some kind of credit. But then, even then the disadvantage of the deficit round robin continues to be that it is unfair, it could be unfair over shorter time scales. So, then we wanted to see that what kind of packet scheduling algorithms we can derive which approximates the fluid versions of the fair queuing algorithm that is the fluid flow fair queuing algorithms.

So, we were working towards that and first we try to see that how we can keep track of the progress of the work done in the fluid flow fair queuing algorithms and accordingly then we take a decision of scheduling the packets in the packetized versions of the fluid flow fair queuing algorithms. So, we define a property called a virtual time of the system.

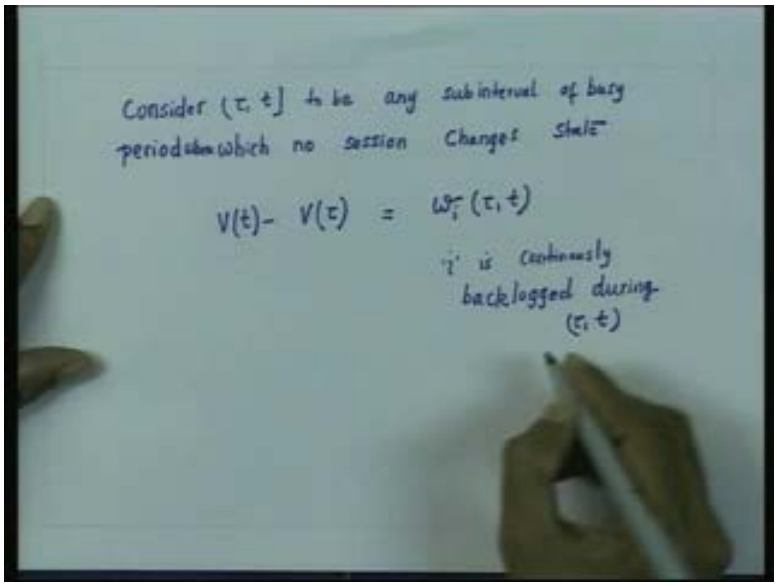
(Refer Slide Time: 3:02)



So, that is what we are seeing that a virtual time is a non decreasing function of time t and it is a piece wise linear function with this slope d v(t) by d(t) is equal to r divide by summation over rho i's where rho i's are the rates which have been allocated to the session i and this summation over this summation is over the set of all sessions which are backlogged at time t and the r is the output link rate. So, we can see that the virtual time, the slope of the virtual time, it changes whenever the set of backlogged users changes. Essentially, the slope of the virtual time is inversely proportional to the number of backlogged users.

So, first of all we will try to prove this result; how do we get this result. So, let us first prove this result. Now, for that let us consider any arbitrary subinterval of the busy period. Let us consider tau to t to be any arbitrary subinterval of the busy period.

(Refer Slide Time: 4:10)

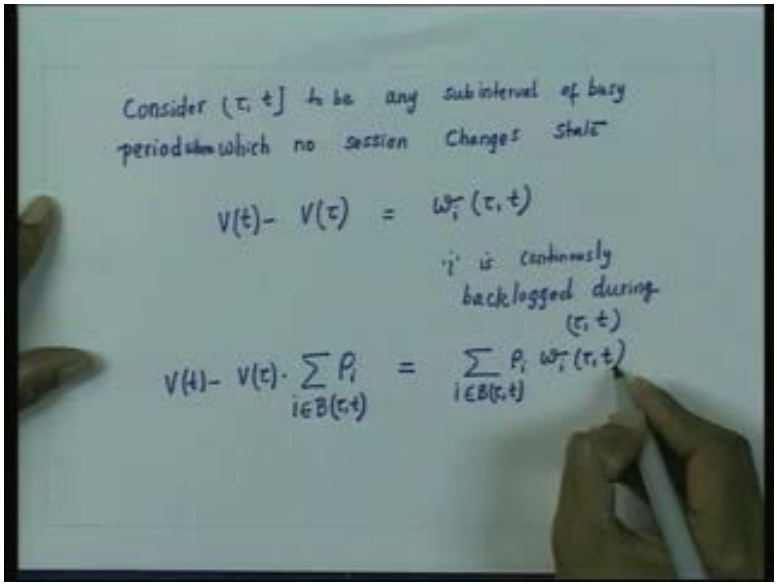


So, we say that consider tau to t to be any subinterval of the busy period during which no session changes states, where or during which no session changes state. If that is so, then by definition we know that v (t) minus v tau, the difference of the virtual time is by definition equal to the normalized service received by the session i where the session i is continuously backlogged during the interval tau to t. So, session i is continuously backlogged during continuously backlogged during this interval tau to t.

So, we have seen that the change in the virtual time of the system is equal to the normalized service received by the session i which is a backlogged session. So, now multiply both sides by rho i and sum over all the sessions which are backlogged during this interval. So, we multiply by rho i and then sum it over all such i which are backlogged during this interval tau to t.

So, if we do this then this becomes i belonging to b tau t where b tau t is the set of all sessions which are backlogged during the interval tau to t, rho $_i$ w $_i$ tau t.

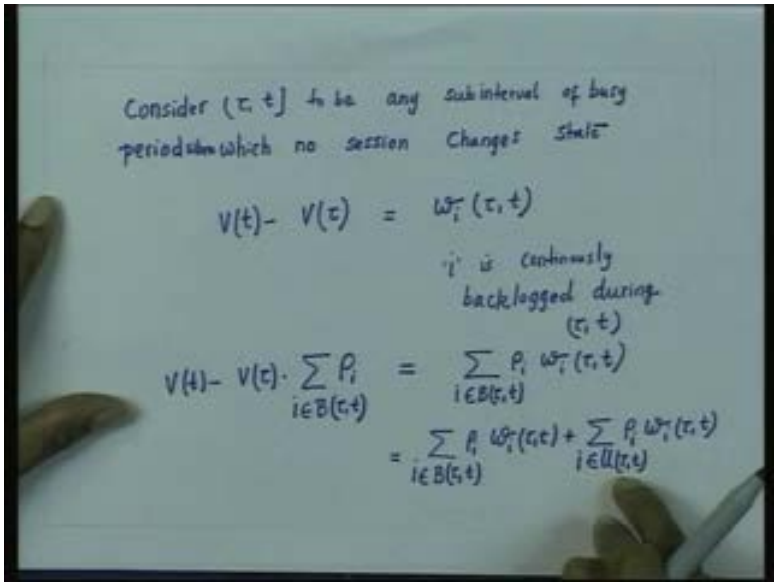
(Refer Slide Time: 6:35)



Now, we have considered this interval tau to t to be an interval where no session changes state. If that is so, then we can also write to be this equal to i belong to the set of all sessions which are backlogged, rho $_i$ w $_i$ tau to t plus rho $_i$ w $_i$ tau to t over all sessions which are absent or non backlogged during the interval tau to t.

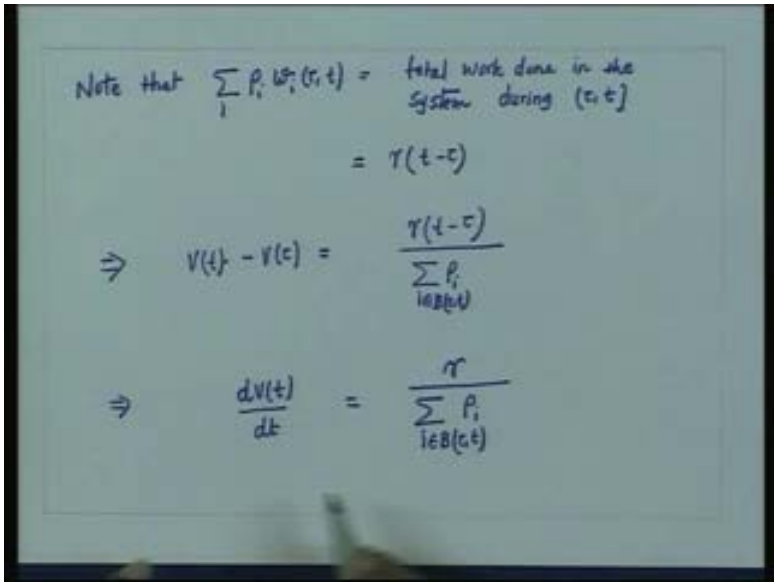
Now, note that the sessions which are absent during the interval tau to t, w $_i$ (t) is actually 0 because they do not receive any normalized service. This actually is 0 for all the sessions which belong to this set. So basically, we have added 0 to this and this means that this is also equal to sum over all i rho $_i$ w $_i$ tau t where this sum is now over set of all sessions.

(Refer Slide Time: 7:25)



Remember again, tau t to be an interval where no session is changing states. So, it is an interval where no session is changing state. So, if you now again see in this equation, then v (t) minus v tau becomes equal to rho. Now, what is this, rho $_i$ w $_i$ tau t?

(Refer Slide Time: 8:19)



It is the total work done in the system during the interval tau to t and therefore this will be equal to… Note that rho $_i$, summation rho $_i$ w $_i$ tau t, this is equal to the total work done in the system, total work done in the system during this interval of tau to t.

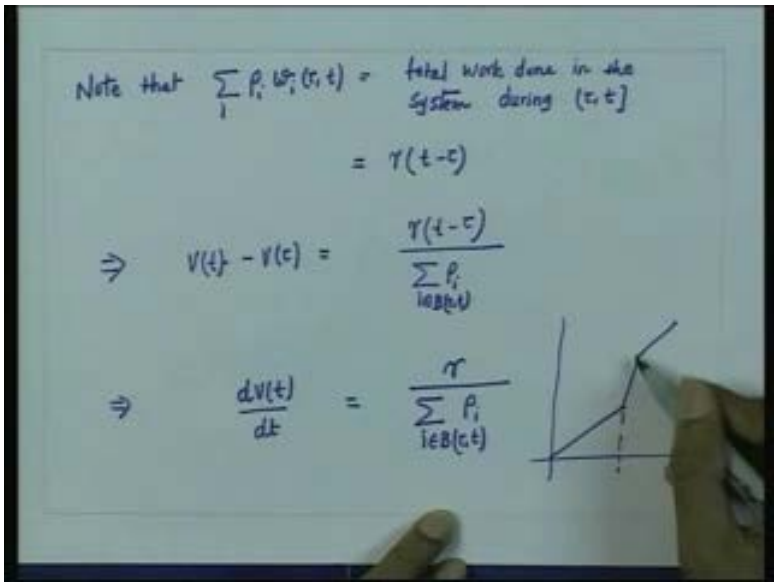
So therefore, this becomes equal to r into t minus tau and this tau to t - remember is a sub subinterval of the busy period and this is a work conserving scheduler. So, the scheduler is continuously working. This is the interval, this is the output link rate and therefore this is the amount of work which has been done by the scheduler and hence from the previous equations if you see, from these equations, this becomes now equal to r $_i$ into t minus tau.

So therefore, we get is v (t) minus v tau - this becomes equal to r of t minus tau upon rho $_i$ where i is the sum over the set of all backlogged sessions. So, this means that the slope d v (t) by d (t), this is given by r upon summation over set of all sessions which are backlogged, summation of rho i's over all sessions which are backlogged.

So, you can see that remember tau to t is the interval where no session changes state. Now, during such an interval where no session is changing state; the virtual time, the slope of the virtual time is inversely proportional to the number of users which are backlogged and it is equal to r $_i$ into summation of all the rates which have been allocated to this different sessions over the set of backlogged users. So, it is equal to r upon summation rho i.

Now, whenever this set changes, that means whenever one of these backlogged users becomes unbacklogged or one of the unbacklogged users become backlogged; then the slope changes. So, therefore the virtual time is actually a piece wise linear function and its slope changes whenever the set of backlogged users changes.

(Refer Slide Time: 11:05)



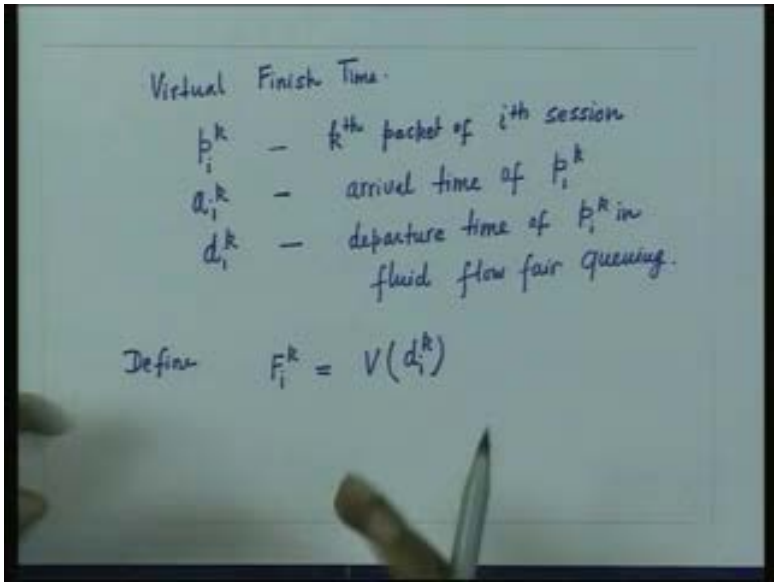
It is something like something like this that this is the slope and which is equal to r upon summation rho i's and if some users becomes unbacklogged, then slope increases if some users becomes ore backlogged and slope decrease and so on. So, it is a piece wise linear function. This could be an interval this could be an interval where no session is changing state. The users which are backlogged are backlogged and the users which are non backlogged are backlogged.

Now, it is at this point some session changes state. So, then you go into another slope where the set has changed and so on. So this is therefore, the virtual time v (t) is a piece wise linear function. Now, we will try to define something called as the virtual finish times. What is our objective? Our objective is trying to define the packetized versions of the fluid flow fair queuing that is what our objective is.

What we have done now is; we have defined something called virtual time which is keeping track of the work done in the system. How? The virtual time increases in proportion to the normalized service received by backlogged session. Now in fluid flow fair queuing, remember that a normalized service received by all sessions which are backlogged is equal. It is equal at every instant of time; the normalized service received by all backlogged users.

So therefore, the virtual time by defining the virtual time in this manner, the virtual time is keeping track of the total service which is being given by the scheduler or the server in the fluid flow fair queuing systems. So now, we will try to attempt defining the packetized versions of this fluid flow fair queuing. So, let us see.

(Refer Slide Time: 13:24)



Now, we define some quantity which we call it to be as the virtual finish time. Let us say that p_i^k is the k'th packet of i'th session, a_i^k is the arrival time of the packet p_i^k and d_i^k is the departure time of p_i^k in the fluid flow fair queuing. This is the d_i^k - denotes the actual departure time, it is the actual time at which the packet departs in the fluid flow version of the fair queuing algorithm.

Now, we define some quantity and this quantity is F_i^k. This quantity is for the k'th packet of the i'th session and defined to be v of d_i^k. So, what it is? It is the virtual time of the system when the packet p_i^k departs. So, at the time of departure of the k'th packet of the i'th session, we see what is the virtual time and we call that virtual time to be F_i^k. So, this we call it to be as the virtual finish time.

Now, first of all note that if t_2 is greater than t_1, then v (t_2) will be greater than v (t_1). Virtual time is actually monotonically increasing. So therefore, if we schedule packets, therefore if we schedule packets in the increasing order of their virtual finish times, if you schedule packets in the increasing order of the F_i^k; then it is equivalent to saying that we are scheduling packets in the increasing order of their departure times in the fluid flow fair queuing. Because, if the departure time of let us say k'th packet of the i'th session is greater than say l'th packet of the j'th session that means if d_i^k happens to be greater than let us say d_j^l; obviously from the monotonousity property of the virtual time, F_i^k will turn out to be

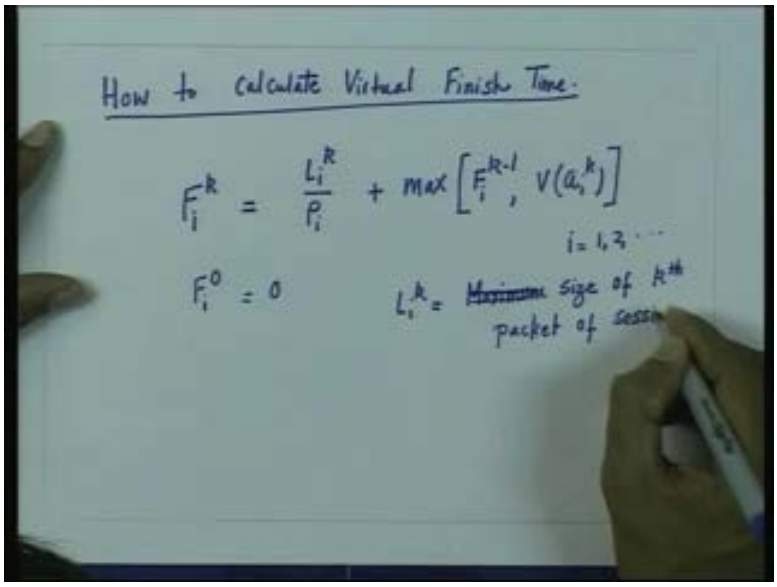
greater than F_j^l. So, if I schedule l'th packet of the j'th session first, note that in the fluid flow fair queuing it will depart first because its departure time is less.

So therefore, in the fluid flow fair queuing, it will depart first. So, if in my packetized version if I schedule this packet first, then I am likely to be fair. That is what I can assume. So, I am proposing a packetized version of the fluid flow fair queuing algorithm, where I am saying that I define something called a virtual finish time and the virtual finish time happens to be the virtual time in the fluid flow fair queueing system when this packet actually departs and then I am saying that I will serve the packets in the increasing order of their virtual finish times.

Now, the first question of course is that if I schedule packets in the increasing order of their virtual finish times, am I fair? That is first question. Second question is that how do I compute these virtual finish times? Do I have some better ways, some efficient way of computing this virtual finish time? Is there any easy way of computing the virtual finish times? So, we would see that. We will answer these 2 questions.

First we will try to see how it is possible to compute the virtual finish time in a very easy recursive fashion and then we will also prove that if we design a packet scheduling algorithm, where we are scheduling packets in the increasing order of their virtual finish times, this algorithm is likely to be more fair or closer in terms of fairness to the fluid flow version of the fair queuing algorithm. So, we will first see how to compute the virtual finish time in a recursive fashion.

(Refer Slide Time: 19:20)



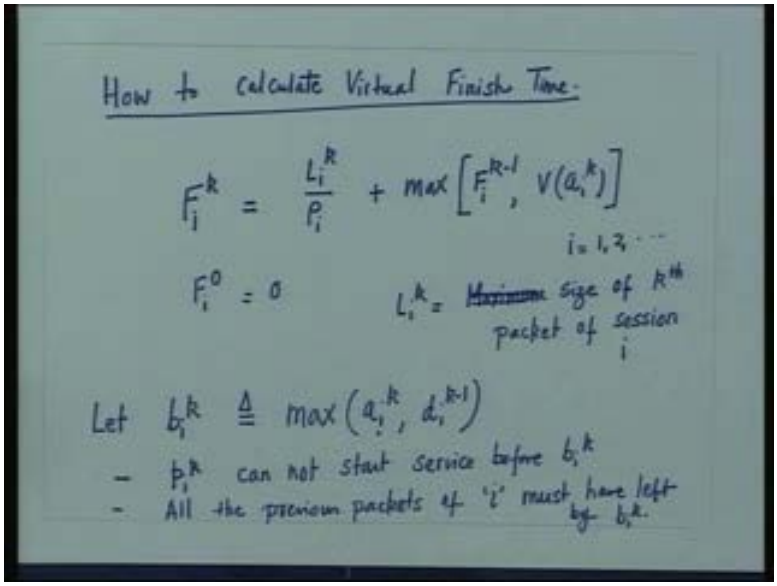
So, let me just define, let me just answer this question that how to calculate virtual finish time. Let us answer this question first. Now, it can be shown that this virtual finish time of the k'th packet of the i'th session satisfies a very easy and nice recursive relations where it is given by L_i^k upon rho $_i$ plus max of F_i^k minus 1 v of that is the virtual time of the p_i^k - this packet for all sessions i is equal 1, 2 and the initial condition which is F_i^0 is considered to be 0 and note that this L_i^k happens to be the maximum size packet maximum oh sorry this is the size of the this is not the max this is the size of the k'th packet of session i.

So, we can see that the virtual finish times can be computed in a very easy recursive fashions. All we need to know is to keep track of the virtual time is to find out the virtual time when a particular packet arrives and if a particular packet arrives and if we that time look at our fluid flow a fair queuing calculator and from that we determine, what is the virtual time when the packet arise. If we know that and if we plug into these equations, we can actually compute the virtual finish times of the packets. So, let us try to first prove this equation.

Now, let us say, let us define a quantity b_i^k is defined as max of a_i^k and d_i^k minus 1. That is what is a_i^k? a_i^k is the arrival time of the k'th packet of the i'th session and what is d_i^k minus 1? d_i^k minus 1 is the departure time of k minus one'th packet of the same session, i'th session.

Now, we define this b_i^k quantity to be maximum of either a_i^k or d_i^k minus 1. So, what is the significance of this b_i^k?

(Refer Slide Time: 22:35)



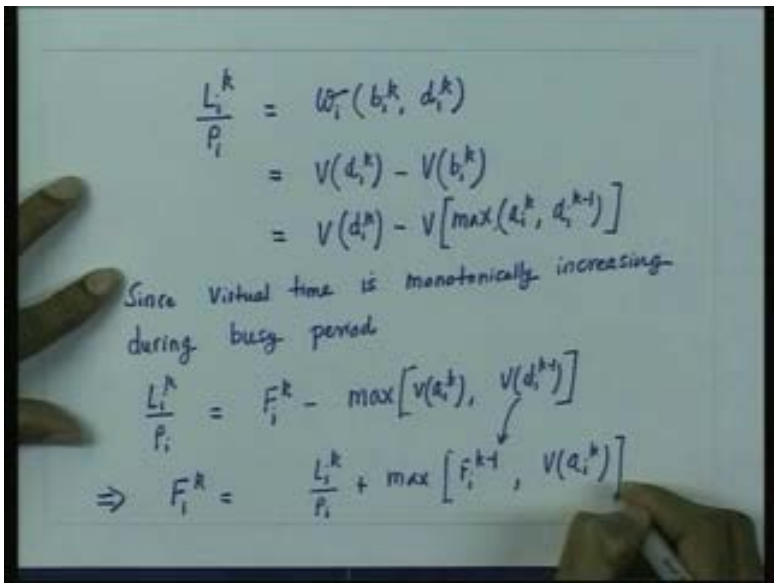
Significance of this b_i^k is that the p_i^k, this packet that is the k'th packet of the i'th session, it cannot start service, it cannot start service before b_i^k. It cannot start service before b_i^k. Why? Either the packet has not arrived if its arrival time happens to be higher; so, either the packet has not arrived or if a_i^k is less than this d_i^k minus 1, then the packet has arrived. But the previous packet, k minus 1'th packet has not yet left. So, it is still in the queue. So obviously, the packet p_i^k, it cannot start service before b_i^k.

Actually, it start service at b_i^k or later in some other system. But in the fluid flow fair queuing, as soon as the packet arrives and since it is a work conserving scheduler, it will start the service immediately.

So, another thing is that all the previous packets of this session i must have left, must have departed by this time - b_i^k. So now, let us see, we define L_i^k minus rho_i. So now, when this packet, the k'th packet of the i'th session that is a packet p_i^k, when it starts the service; the L_i^k is the length of the packet in bits and rho_i is the rate allocated to it. So obviously,

this will be in a normalized service received by this session during the interval for which it is backlogged and what is that interval? That interval is extending from b_i^k to d_i^k, d_i^k is the time when this packet will depart.

(Refer Slide Time: 25:03)



So, I write this to be equal to that this is equal to the normalized service received by the session b_i^k, d_i^k. Note that the session i is continuously backlogged during the interval b_i^k to d_i^k. During that interval b_i^k to d_i^k, the session is continuously backlogged and therefore I can write this to be virtual time v, virtual time at d_i^k minus virtual time at b_i^k. Now since, now, what is b_i^k? So, I just write here at in the presence of b_i^k, b_i^k is the virtual time of max of a_i^k into d_i^k minus 1.

Now since, we are considering during the busy period; since virtual time is a monotonically increasing function during this interval, during the busy period, since virtual time is monotonically increasing, monotonically increasing during busy period, so since it is monotonically increasing, I can change this operator max and v. So, what I get is that L_i^k upon rho $_i$ is equal to… what is v_d^k? v_d^k by our definition is F_i^k that is the virtual finish time minus if I change this max of, since virtual time is monotonically increasing, virtual time of maximum of this is same as first computing the virtual times and then taking the maximum quantity and what is this is F_i^k minus 1? So, we get this recursive equation that F_I^k is equal to L_i^k upon rho plus max of F_i^k minus 1 which is what, this is into v of a_i^k.

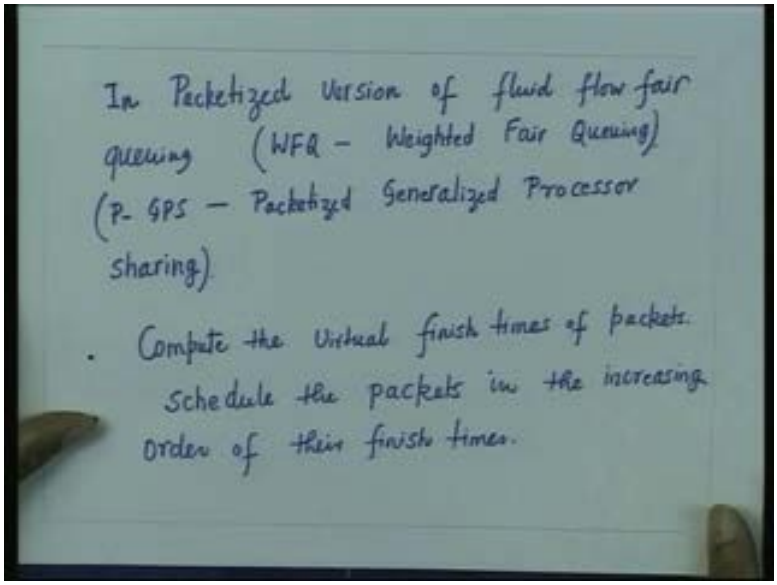
Now, what does it mean is it means that the virtual finish times of the packet can be computed recursively. This is this equation is easier because by definition the virtual finish time of a packet is the virtual time when the packet will depart in the fluid flow fair queuing. Now, that time when the packet will actually depart in the fluid flow fair queuing system is actually difficult to determine.

So, we have found a recursive way of computing the virtual finish time which states that the virtual finish time of the k'th packet of the i'th session can be recursively computed if you know the virtual finish time of the k minus 1'th packet of the i'th session and the virtual time when the packet actually arrives. When the packet actually arrives, you can easily see in the

fluid flow fair queuing system what is the virtual time that is currently going on and then plug that into this equations to time stamp a packet.

So, the question really is that now if I schedule the packets in the increasing order of their virtual finish times, then am I being fair? So, I will just state the result without proving it and then if in the repeat in the next lecture, we will try to prove this result.

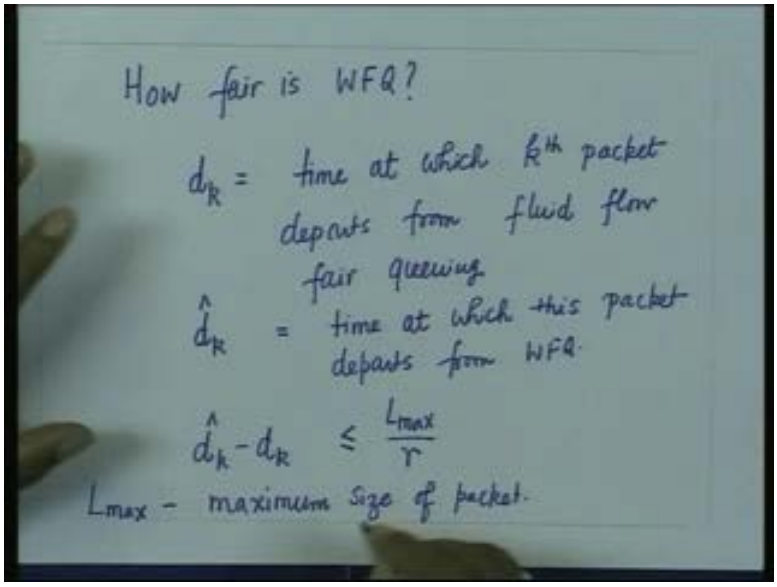
(Refer Slide Time: 29:16)



So now, I am defining the packetized versions of the scheduling algorithms. So, what I am saying is that in packetized versions of this fair in packetized versions of the fluid flow fair queuing algorithm which I also call as the WFQ or weighted fair queuing or it is also called as PGPS as packetized generalized processor sharing; so what I am saying is that if I compute the virtual finish time, compute the virtual finish times of packets and then schedule the packets in the increasing order of their finish times, so I schedule the packets in the increasing order of their virtual finish times, it turns out that this algorithm is quite fair.

How fair? Let me just state this by a result.

(Refer Slide Time: 31:24)



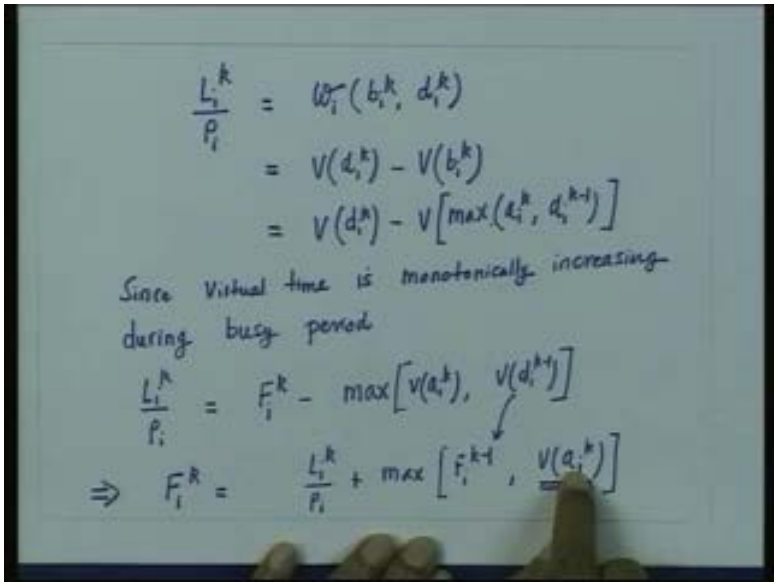
So, let us say, so the question really is that how fair is WFQ? Let us say we define as d_k to be the time at which some packet let us say the k'th packets departs from fluid flow fair queuing. d_k is the time at which the packet departs under the fluid flow fair queueing and let us say we define d_k head to be the time at which this packet departs, this packet departs from our WFQ. Then it can be shown that d_k head minus d_k that is the difference in their departure times will be bounded by L_{max} by r, where L_{max} is the maximum size of the packet.

So, what this algorithm? What the WFQ can do is that that the difference between the departure times of this WFQ and the departure time had this packet been served in a bit by bit round robin manner that is the fluid flow fair queuing, then the difference between this departure is actually bounded and it is bounded by L_{max} by r, where L_{max} is the maximum size of the packet.

So utmost, the difference in their departure times will be L_{max} by r, where L_{max} is the maximum size of the packet, not more than that. So now, here you can see that the algorithm is likely to be let us say unfair over an interval which is shorter than let us say L_{max} by r. But otherwise, the algorithm is likely to be fair. I mean roughly speaking; this is what this result is trying to say that over an interval less than L_{max} by r, this algorithm is unfair. But then we can do nothing more than that in a packetized versions of the fluid flow fair queuing algorithm.

In a fluid flow fair queuing algorithm, the normalized service received by backlog sessions is equal at every instant of time. This you cannot make it equal in the packetized versions because packet is the smallest entity that you need to serve. Now, we come back to our again, implementations of the WFQ and try to see are there any complexities in terms of implementing the virtual finish times.

(Refer Slide Time: 35:02)



So, now what do we see in terms of computing the complexity in the virtual finish time is that since in the WFQ essentially what it means is that you require the computation of the virtual time that is the v (t). Now, you can see that this v (t), it changes slope whenever the set of backlogged users changes. So, it changes its slope whenever the set of backlogged user changes whenever the set of backlogged users changes.

So, computing the virtual time means that you need to keep track of the backlogged users which is not so difficult because what is happening really is that a virtual time is monotonically increasing and whenever the set of backlogged user changes, the slope of the virtual time changes. But the difficulty is that suppose, right now, a packet is in service, this packet starts service at time let us say t_1 and it finishes the service at time t_2. During this interval t_1 to t_2, some more packets may arrive. Now, remember that we need to compute, we need to know the virtual time at every packet arrival. From this if you see, we need to know the virtual time at every packet arrived.

Now, what may happen is that during this interval t_1 to t_2, when that particular packet is served, it might be possible that the set of backlogged users during any arbitrarily small interval say t_1 to t_2, the set of backlogged users which may change may be of the order of the total number of sessions also because it may happen that in the fluid flow fair queuing, all sessions or a large number of sessions may finish service simultaneously. If that happens, then this set of backlogged users changes; if that changes, the slope of the virtual time changes.

So therefore, during in any arbitrarily small interval of time, it might happen that the number of backlogged users, the change in the number of backlogged users may be of the order of the total number of sessions. So, the break points in your virtual time slopes may be arbitrarily very large approaching the total number of sessions and in a typical remember that this packet scheduling algorithms you are implementing in a router and in a typical router the number of flows may be as large as 1,00,000 or even if 10000; then in order to compute the virtual time, we actually then the complexity of computing the virtual time, if it approaches the total number of sessions, then the number of computations that may be required may be

very large during a small interval of time and as a result what may happen is that we may not be able to compute the finish tags or the virtual finish tags of the packets in a fast manner and therefore our packet scheduling algorithm may not work at high speeds So, this is the major problem.

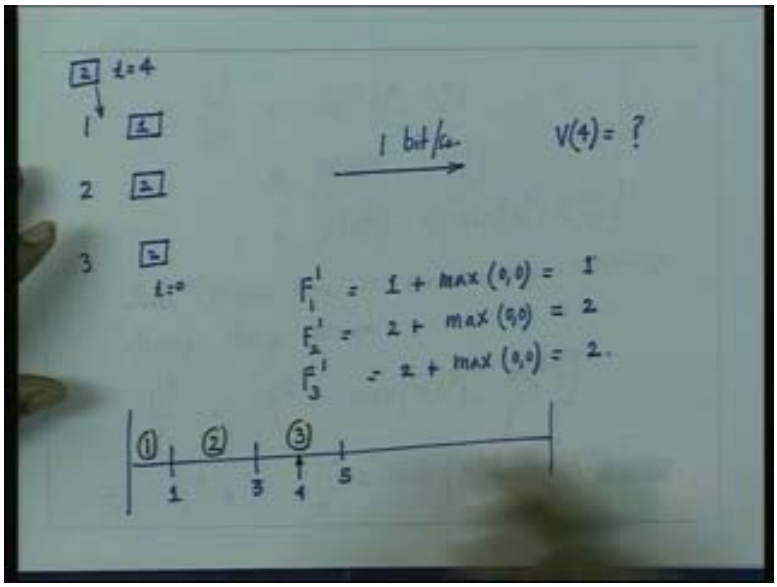
So therefore, now we are saying that the virtual time changes it slopes and whenever the set of backlog and users changes and the number of break points therefore, the number of break points in the v (t), the slope v (t) can be as high as the total number of as high as the total number of sessions and therefore this may lead to the real time computation, may be the real time computations of v (t) can become a problem.

So, even though it appears that to compute the time stamp of the packet that is the virtual finish time, we have found a very simple nice recursive algorithm, where the virtual finish time of the packet can be computed recursively by knowing the virtual finish times of the previous packets and the virtual time of the system when the packet arrives. So, we have found a very nice recursive way. Even then we find that computing the virtual time itself could be a difficult task because the number of break points can be as large as the total number of sessions and the total number of sessions themselves can be quite large and therefore the computation of the virtual time can be a problem.

Otherwise also, the computation of the virtual time is not so much of a tough task. All we need to know is that we need to keep track of the number of backlogged users; virtual time otherwise increasing monotonically and whenever the set of backlogged users changes, the slope changes. But the problem really is that the set of backlogged user, the change in the set of backlogged users can be very large and therefore the breakpoints could be very large and therefore the real time computation of the virtual time could become difficult.

Now, before we go to address this problem, let me just give you an example of how actually a weighted fair queuing algorithm will work by giving you an example and computing the virtual finish times. So, let us see with an example. In this example let us consider, a very simple example where we are saying that there are 3 sessions; now, session 1, 2, 3.

(Refer Slide Time: 41:25)



Let us say the output link rate is for the time being is 1 bit per second, this session gets a packet of 1 bit, this session gets a packet of 2 bits, this session gets a packet, all at time t is equal to 0. Then, we will assume that a packet of session, 2 bits arrived at time t equal to 4. Now, if you compute the virtual finish tags of these packets, so how do we how do we compute the virtual finish tags of these packets? F_1^1 as by giving this example of F_1^1 is assume the allocated rates of this row i's are all equal for them.

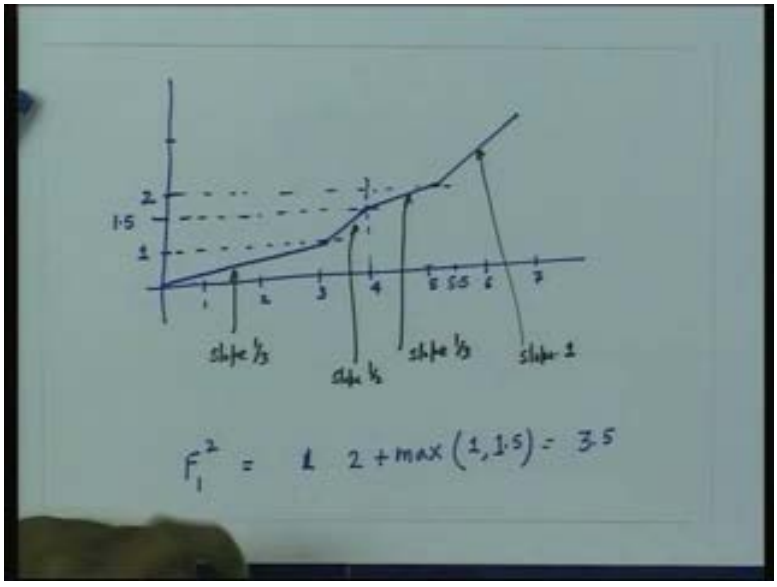
So, the weights are equal which becomes equal to let us say 1 plus max of 0 which becomes equal to 1. Then, F_2^1 which is equal to 2 plus max of 0 and this becomes equal to the 2. Then F_3^1 which becomes equal to let us say 2 plus…

So, now we need to schedule the packets in the increasing order of their finish times and we schedule the packets; first packet of this session 1, second packet of this session 2 and the third packet of this session 2.

So essentially, this packet gets served. So, if I serve these packets, this will finish its service – 1 bit per second at the time 1. If I serve this packet then this packet will finish its service at time three and this packet will finish its service this third packet of time 5 because there is a tie between these 2. So, we can choose any arbitrary packets. So, the session 1 packets get served here. So, this is session 1, this is session 2, this is session 3.

Now, note that at 4, the session once this packet arrives and that time I need to know what is the virtual time. So, I need to know what is the virtual time at time 4. Now, this I need to compute so let us say what is happening in the fluid flow fair queuing. So, if you see in the fluid flow fair queuing, it is 1 bit per second one third, one third, one third for bits will be taken up from each of the from each of the flows.

(Refer Slide Time: 44:53)



So, the virtual time actually increases something like this that this is 1, 2, 3, then 4, then 5, then 6 and then 7 and so on. So, if you see here let us say 1, 2 and 3 and so on. So, if you see, so the slope of the v (t) is one third and this at time t, if you are serving in a fluid flow fair

queuing, then the virtual time slope is one third. And, if you see at time 3, in the fluid flow fair queuing, the session 1, the sessions one's bits would have been served.

Now, at time 3 since the session one's bit would have been served, this session would have completely served, now only 2 sessions - 2 and 3 are backlogged and therefore this it changes its slope. So now, it becomes half. It becomes half till about 4 so till about four and at that time, the value of the virtual time therefore so note that this slope is one third of the virtual time. So, the slope here is one third. Now, this slope changes and this becomes, the slope becomes half.

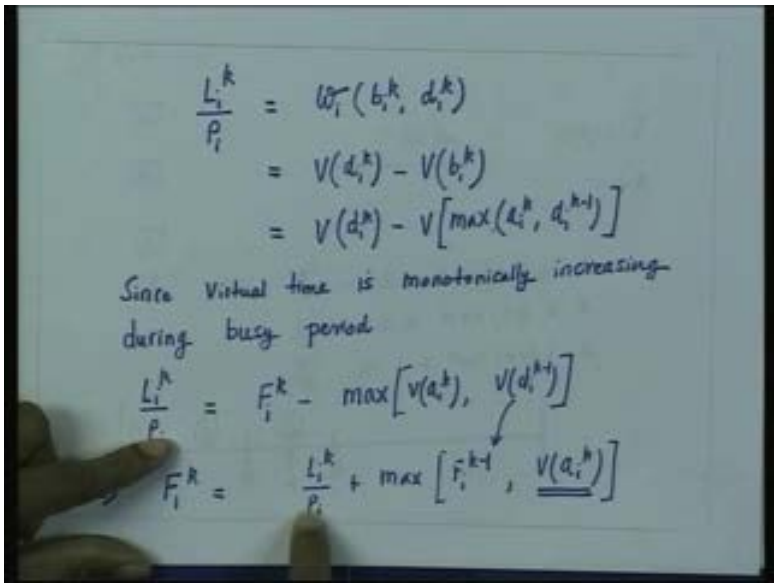
So, at time 4, the second packet at the session 1 has arrived and at that time, the value of the virtual time will be 1.5. So, we wanted to know what is the virtual time at time 4. So, this is actually we determine that 1.5. So, if it is 1.5 and if you wanted to calculate F_1^2 that is the finish time of the second packet of the first sessions; that becomes equal to a packet of length 2 arrives. So, 2 plus max of the finish time of the previous packet which was 1, 1.5.

So, this becomes equal to 3.5. So obviously, this is less than the finish time. So, this is greater than sorry this is the greater than the finish times of either of these that is the third session. So therefore, this packet only is likely to be served in the packet scheduling algorithms.

Now, let us look at again, after this what happens, the packet has arrived in the first session also, so the slope changes again and the slope now becomes equal to one third and at this point, it becomes one third again and at this point at 5.5 here both the sessions that is session 2 and session 3; they are completely served and after this the slope becomes 1 because only the user 1 is now backlogged.

So at this point, the slope again becomes one third from 4 to 5.5 and then at this point remember that both the sessions that is the session 2 and 3, they have been served completely, their both bits and then only this session is remaining backlogged with the packet 2. So therefore, this slope becomes now slope is 1 and this packet finishes service at 0.7. So, this way this slope is 1.

(Refer Slide Time: 49:14)



So, we can see that this way the virtual time changes its slope whenever the set of backlogged users changes. So here, I just wanted to clarify here that even though in our previous equations that we had considered that a virtual finish time is given by L_i^k upon the rho_i where rho_i is the set of, is the allocated rate is the allocated rate. But in computing this virtual finish times here, I have taken this rho_i's of all the 3 users to be 1. Essentially, they are considered as the weights, so I have given them the weights of 111 which means that all the 3 users have been allocated equal rights.

So, I have considered the specific values of rho_i's in place of rho_i's you could also consider the weights, the different weights which have been given to the 3 individual sessions. So, I mean that is the minor point that I want to make here because of the particular example that I have considered, really I have illustrated it by considering that a different weights have been given to them and they are not actually in terms of the allocated rate. So, weights I have considered, so the weights and the rates; they do not make a sort of much distinction in terms of computing the virtual finish times.

So, this gives an example and also an idea how to compute the virtual finish time. So, I again want to reiterate that what we were discussing is what we thought is that we wanted to have an algorithm which approximates the fairness characters fairness character of the fair queuing algorithm. So, we considered both the weighted round robin and the deficit round robin and found them to be approximate versions of the fluid flow fair queuing algorithms but not so fair as we would like to be.

Then we considered this packetized versions of the fluid flow fair queuing algorithms which we called as the weighted fair queuing algorithms and we saw that we can derive a fair version of the packetized fluid flow fair queuing algorithm provided, if you keep track of the actual work done in the system of the fluid flow fair queuing and then schedule the packets in the order of their virtual finish times and we can see that this algorithm is fair, in the sense that there is a difference between the departure times in the packetized versions and the fluid flow fair queuing versions will remain bounded by the maximum size of the packet divided by the output link capacity.

So, it turns to be very close to the fair queuing algorithm in terms of fairness. But the problem remains that we need to compute the virtual time for the implementations of the weighted fair queueing algorithms and computing the virtual time actually requires keeping track of the set of the backlogged users and that can pose a problem in terms of the real time computations.

So, we should look for algorithms which can address these problems in the sense that the real time computation of the virtual time can become easier and at the same time we retain the basic characteristics of the fairness of the packetized versions of the scheduling algorithms. So, that we will address in the subsequent lectures.

(Refer Slide Time: 52:37)