Broadband Networks

Prof. Karandikar

Electrical Engineering Department

Indian Institute of Technology, Bombay

Lecture - 12

Fairness of WFQ and SCFQ Scheduling Algorithms

Now, we were discussing the weighted fair queuing algorithm in the previous sessions and what we saw is that the weighted fair queuing algorithms requires a computation of a quantity called virtual time and virtual time keeps track of the work done in the fluid flow fair queuing systems and then from the work done in the fluid flow fair queuing systems, we compute something called virtual finish time and that is something like a service tag.

Each packet is then stamped with this service tags and in the weighted fair queuing or the packetized versions of the fluid flow fair queuing, the packets are then served in the increasing order of these service tags or what is called as the virtual finish times. But we have also seen that the computation of this virtual time in real time can be difficult because the computation of this virtual time actually requires keeping track of the set of backlogged users. This quantity virtual time is inversely proportional to the number of users that are backlogged and whenever this set changes that whenever the set of backlogged user changes, there is a break point in the virtual time curve.

So, typically the virtual time is monotonically increasing with a slope which is inversely proportional to the number of backlogged users and whenever this set changes, the slope changes. If the number of backlogged users becomes more, then the slope decreases. Otherwise, if the number of backlogged users becomes less, the slope increases.

Now, the difficulty is that in generalized processor sharing or the fluid flow fair queuing, the number of users, there may be many number of users who may finish their service simultaneously or there may be many queues which may become backlogged simultaneously. So, as a result, the break points in the virtual time may approach the total number of sessions in the systems and the total number of sessions in the systems can be arbitrarily very large in a let us say in a core router or a wide area networking router.

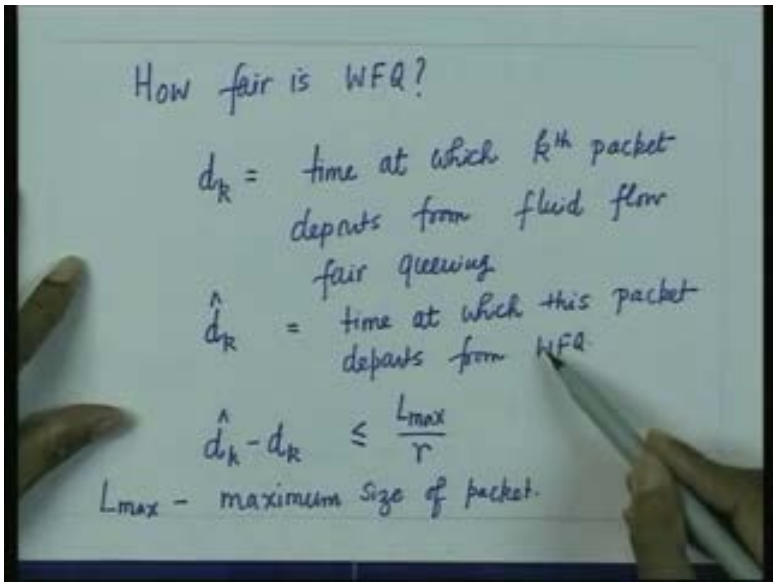
If this number of sessions are very large, then the computational complexity will become of the order of the total number of sessions. So therefore, the real time computation of the virtual time may become difficult. Now, today we will see what are the other methods of having packetized versions of the fair queuing algorithms which are simple to implement, which has less implementation complexity.

But before we go towards that, as we had seen last time that we wanted to ask this question that how fair is the packetized versions of the fluid flow fair queuing algorithm in the sense of max-min fairness? And, we had seen that the fluid flow versions of the fair

queuing algorithm is max- min fair and there were several packetized versions which we have conceived of or thought of like one is weighted Round Robin or the Deficit Round Robin and we found that these techniques were unfair over shorter time scales and that is the reason we suggested the weighted fair queuing algorithms.

And, we state and we wanted to know that how fair is the weighted fair queuing algorithms and then we have this result where we said let d_k is the time at which the k'th packet departs from the fluid flow fair queuing and d_k hat is the time at which this packet departs from the weighted fair queuing.

(Refer Slide Time: 4:18)



Then the difference between their departure times in the two scheduling systems is bounded by the L by L_{max} by r, where L_{max} happens to be the maximum size of the packet and r happens to be the output link rate.

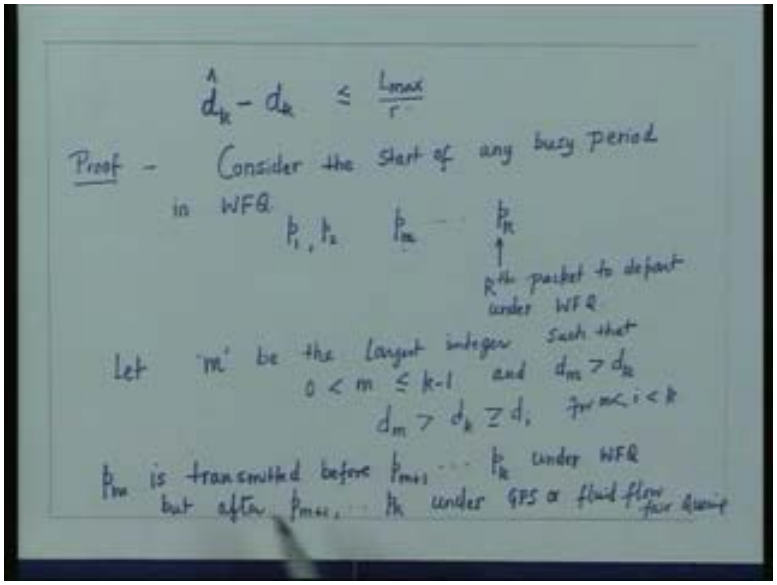
So, what we are saying is that the difference between the difference between the departure times of the packets in the two systems: one is the fluid flow versions which is just a abstract or a hypothetical systems and the other which is the packetized systems. The difference between their departure times will always be bounded by the maximum size of the packet divide by the output link capacity.

So, to that extent now, L_{max} by r is the time it will be taken for a scheduling the maximum size packets. So, to that extent the difference between the two departure times is bounded by the time taken to transmit the maximum size packet, the time taken to transmit the maximum size packet and that is the best that can be apparently done in the packetized versions of the fluid flow fair queuing algorithms.

But let us prove this result and try we will try to prove this result and then we will see whether we can have algorithms which are simple to implement in terms of the virtual time computations and as well as they are somewhat fair.

So, first let us prove the fairness of the weighted fair queuing algorithms. So, we will try to prove this result. Now, one thing that we have observed before we prove this result is that since both the weighted fair queuing algorithm and the fluid flow fair queuing algorithms are work conserving, since they are work conversing; their busy periods will coincide. So, this is one important thing that we must keep in mind that a busy period of the weighted fair queuing algorithm will be equal to or if the same, this will be equal to the busy period of the fluid flow fair queuing algorithms. So, we will prove this result over any busy period.

(Refer Slide Time: 6:41)



So, let us prove this result that is d hat k minus d_k, this is bounded by L_{max} by r. So, we would like prove this result. So, let us consider the start of busy period and we call it to be the time 0. So, let us consider the start of any busy period. Let us say we are considering the start of any busy period in the weighted fair queuing algorithm which is also the same in the GPS or the fluid flow fair queuing and let us say in the weighted fair queuing algorithm, p_1, p_2 these are all packets which depart in this order. So, p_k is the k'th packet to depart. So, this is like saying that k'th packet to depart under PGPS.

Now, what we say is that let m be the largest integer such that m lies between 0 to k minus 1 and d_m is greater than d_k and also that d_m is greater than d_k is greater than or equal to d_i for all. So, what we are trying to say is that m, let us say m is the largest integer. So, it happens for the first time at m such that all these packets afterwards; p_m plus 1, p_k etc, they will depart in the fluid flow fair queuing algorithms before sorry after before p_m.

So, now what is happening is that what we are saying is the d_m is the d_m is the departure time in the GPS. d_m is the time at which the packet departs in the fluid flow fair queuing. What we are saying is that m is the largest integer. Now, look at this way that p_1 p_2 p_k, in this order the packets are departing in the weighted fair queuing.

Now, m is the largest integer where d_m is greater than d_k. That means the packet p_m that departs in weighted in fluid flow fair queuing after p_k, it departs after p_k. In PGPS, it departs before p_k. So, in weighted fair queuing, it is departing before p_k. But in fluid flow fair queuing, it is departing after p_k.

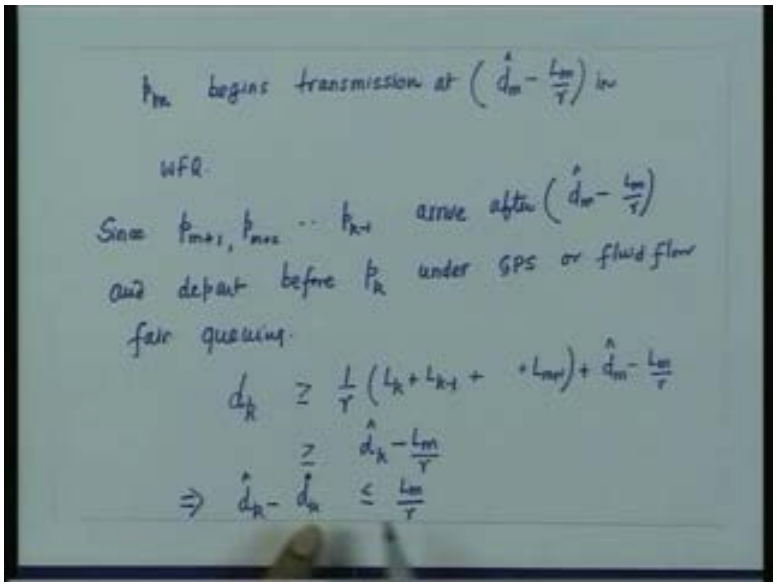
Now, this happens for the first time in m. Before that before that what we are considering is that all the packets, they depart in this order either earlier in weight GPS or at least at the same time. So, now this is the first time it is happening at the m'th instant where the departure time d_m of the m'th packet happens to be greater than the departure time of the k'th packet.

So, we will so what we are trying to say is that the packet p_m is transmitted before p_m plus 1 so on upto p_k under WFQ but after all these packets - p_m plus 1 p_k under GPS or fluid flow fair queuing. So, that is what we are trying to say.

Let m is the largest integer. That means it happens for the first time in m where the packet p_m happens to be transmitted before p_m plus 1 p_k. So, this packet obviously is transmitting before all these packets in the weighted fair queuing. But this packet gets transmitted after p_m plus 1 p_k etc in the fluid flow versions of the fair queuing algorithms. If no such m exits, then obviously m is equal to 0. So, we will prove this results for m greater than 0.

So, what we were saying is that the packet p_m is transmitted before p_m plus 1 p_m plus 2 p_k under weighted fair queuing but after p_m plus 1 p_m plus 2 p_k under the fluid flow fair queuing.

(Refer Slide Time: 12:47)



Now, let us see that the packet p_m, it begins transmission at d hat m minus L_m by r in weighted fair queuing. Now, d hat m is the time at which the m'th packet that is the p_m packet departs in the weighted fair queuing. L_m is the length of the packet and L_m by r is the transmission type. So therefore, the p_m packet will begin transmission at this time.

Now, note that this packets that is p_m plus 1 p_m plus 2, all these packets, these p_k - all of these packets obviously arrive after d hat m minus L_m by r. However, they depart they depart before - this is p_k minus - before p_k under the generalized processor sharing or the fluid flow fair queuing. And, this is easy to see but assume that the packet p_m begins transmissions at this time and it is very easy to see that this packets m plus 1'th packet, m plus and upto k minus 1 packet, they all arrive after d hat m minus L_m by r. But all these packets depart before the k'th packet under the fluid flow fair queuing.

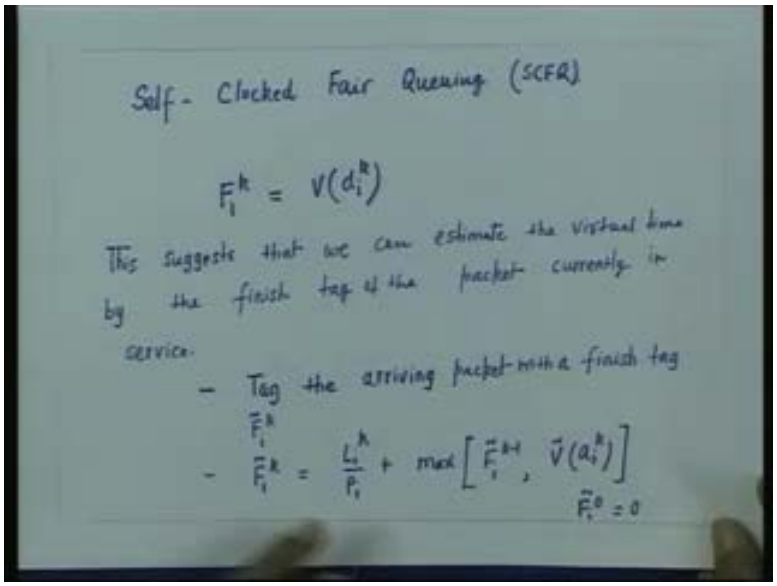
So therefore, d_k will be greater than all. Now, d_k is the time at which this k'th packet departs in the fluid flow fair queuing system. So therefore, d_k is greater than or equal to 1 upon r L_k plus L_k minus 1, these are all the length of the packets of the respective packets so on till L_m plus 1. These are all the times it will take to transmit the k'th packet, k minus 1'th packet and m plus 1'th packet plus d hat m minus L_m by r because all these packets, they are all arriving after d hat m minus L_m by r. And therefore, this should be equal to and if i add these d hat m to all that will become equal to d hat k minus L_m by r and this shows that d_k minus d hat k is less than or equal to L_m by r and the maximum size of the L_m could be L_{max} by r.

So therefore, we can say that the difference between the departure times of the packets that is d_k minus d hat k sorry this will be d hat k minus d_k. That is the departure times of the packets in the fluid in the weighted fair queuing and the fluid flow fair queuing or the GPS will be bounded by the transmission time of the maximum sized packets. So, this we have proved, we know the result that how fair is the weighted fair queuing algorithm.

Now, as we have seen, we would like to see whether there is a mechanism by which we can reduce the computational complexity of the weighted fair queuing algorithm. And, Golestani in 1994 proposed a way of implementing fair queuing algorithm in the packetized versions which reduces the computational complexity of computing the virtual time. That algorithm is called self clocked fair queuing algorithm.

So, let us see how we can implement a computationally efficient packetized fair queuing algorithm which is called self clock fair queuing algorithm.

(Refer Slide Time: 17:29)



Now, before we go to the self clock fair queuing algorithm which is actually trying to see whether we have an approximation to computing the virtual time or a computationally efficient way of computing the virtual time; so we have self clocked fair queuing which is also called as abbreviated as SCFQ.

Now, one thing we should note that F_i^k, how are we computing the virtual finished time in the weighted fair queuing algorithm F_i^k? Note that F_i^k, we are computing with the virtual time when this packet will depart in the fluid flow fair queuing. What we are doing What we are doing is that we were keeping track of the virtual time and we would find out what is the virtual time when this k'th packet departs in fluid flow fair queuing and at that time, whatever the time that we note, we call it the service tag or the virtual finished time. So, that is how we were computing the virtual finish time.

Now, this suggests that we can estimate the virtual time. It should be possible for us to estimate the virtual time by the finished tag of the packet currently in service. So, this suggests because what we are doing is that since we are serving the packets in the increasing order of their finished tags and our finished tag that are being computed in the weighted fair queuing, our philosophy of computing the finished tag in the weighted fair

queuing is that the finished tag is actually the virtual time and this packet will depart in the fluid flow fair queuing algorithm.

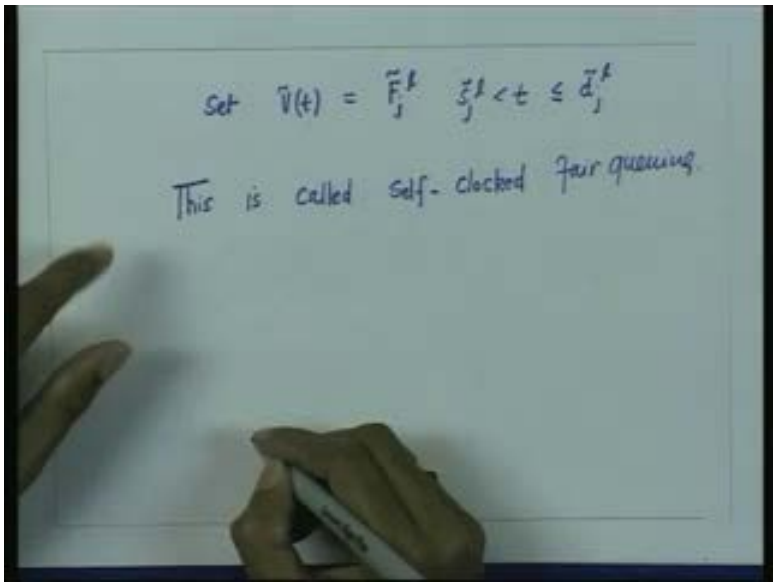
So, if you really want to have a scheduling algorithm which closely approximates in terms of fairness to the fair queuing algorithm or the weighted fair queuing algorithm but does not require the computational complexity of the weighted fair queuing algorithm; then they suggest that suppose somehow we know the finish tag of the packet which is currently in service, so if we know the finish tag of the packet which is currently in service, then by following the philosophy by which we had computed the virtual time in the weighted fair queuing algorithm, it should be possible for us to estimate the virtual time by looking at the finish tag of the packet which is currently in service. And, after that we can use the same recursive formula which we had used in the weighted fair queuing algorithm for computing the finished tags.

So, let us say now that the j'th packet of some lath session or the l'th packet of some j'th session l'th packet of some j'th session is currently in service, let us say. Now, we find out what is the finish tag of this packet which is currently in service. By finding out this finish tag, we estimate the virtual time and if you estimate the virtual time and then subsequently the finish tags of the i'th session, we can compute in the same manner in which we were computing it in the weighted fair queuing algorithm.

Specifically, what we are trying to say is that we will tag the arriving packet, let us say the k'th packet. Let us say that a k'th packet has arrived in the i'th session, then we will tag the arriving packet with a finish tag of let us say F_i^k tilde. Now, this tilde denoting that it is not the virtual finish time in the manner in which we had computed in the weighted fair queuing but it is an approximation, approximate way of computing that and then the finish tags will be computed as L_i^k upon rho $_i$ plus max of F_i^k minus 1 tilde. That is the finished tag of the k minus 1'th packet and the virtual time of the packet which has arrived, estimated virtual time, so the tilde denoting that the estimated virtual time when this k'th packet has arrived.

Obviously, our starting point is initial condition is that F_i^0 is 0. So, we initialize this virtual time and the server becomes ideal and similarly this finished tag of the queue that is of the 0’th packet will start with the 0. The question now is how we estimate this virtual time when the packet arrives.

(Refer Slide Time: 23:39)



Now, we say that we set the virtual time at time t to be the finish tag of the l'th packet of the j'th service which is currently in service. So, that means if the time t lies between… for all the time which lies between the starting time of the packet in service, the time when it starts its service in the packetized fluid flow fair queuing algorithm which is denoted by s hat l j. That means this is the starting time of the packet for the service of the L'th packet of the j'th session.

So, for the time t, for all time t lying between this start times and the departure times that is d, for all these times; we will set the virtual time to be equal to the finish tag of the packet which is F_j^l and after putting this we will compute we will compute the finish tags of the i'th sessions packet.

Now, the situation is something like this that some l'th packet of the j'th session is currently in service. Now, let us say that this packet starts its service at time s j and it finishes its service at time d j. So, between all those times we will compute the virtual time equal to the finish tag of this packet.

Now, some packet, k'th packet has arrived let us say in the i'th session. So a_i^k, what we are tying so say that this a_i^k is the time which lies between this is let us say s_j^l and v_j^l. So, if a_i^k lies between these times, then we will put this virtual time to be equal to the finish tag of the packet which is currently in service. So, in some sense what we are trying to say is that the computation of the finish tag is since determined from the finish tag of the packet which is currently in service, so this algorithm is therefore called self clocked fair queuing algorithm.

This algorithm, note that it does not require an emulation of the fluid flow fair queuing algorithm, this algorithm does not at all require an emulation on the fluid flow fair queuing algorithm which was required to be done in the weighted fair queening algorithm

and that was the lead cause of the computational complexity of the weighted fair queuing algorithm. This algorithm does not require any emulations, this is self clocked. So, that is the reason this algorithm is called self clocked fair queuing algorithm.

Obviously, the question is same, that is the self clock fair queuing algorithm fair? So, for this, Golestani introduced a notion of what is called as relative fairness. So, we will study the notion of relative fairness later. But just try to understand the fairness notions somewhat differently.

Now, note while computing the finish tag in the self clock fair queuing algorithm, we had put this finish tags to be recursively computed by L_i^k upon ρ_i. Now, note that L_i^k is of course the length of the packet and ρ_i is the rate which has been allocated to it. So therefore, this is the transmission time plus we are adding these. So, the question obviously that arises is that why we add this quantity like this? Why do not we do something like this that F_i^k that is the finish tag, we compute as L_i^k upon ρ_i plus F_i^k minus 1?

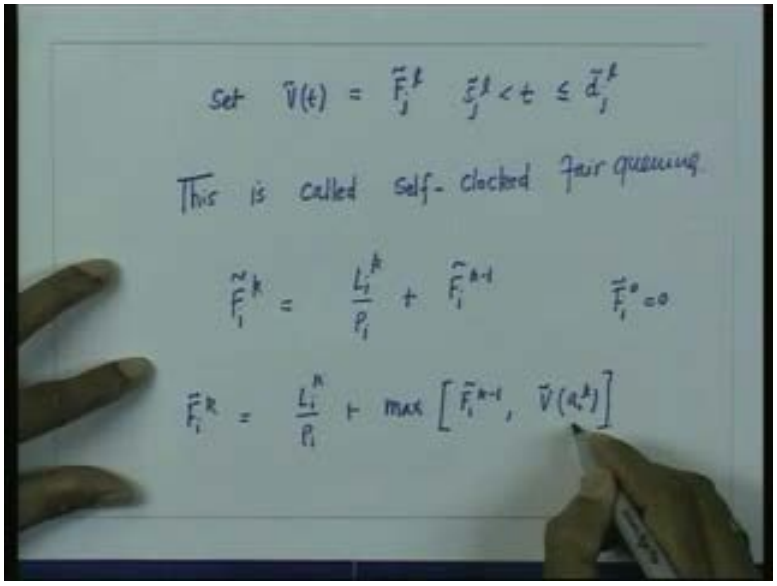
That is if we know the finish tag of the k minus 1'th packet which is the packet ahead of us, then if we know the finish tag of that, then we keep on simply adding the length of the packet divided by ρ_i that is the transmission time and then compute the finish tags. So, what is the, what is the difficulty or what is the problem in having like this? The difficulty is that suppose right now a packet is in service and let us say that its finished tag is F. Now, let us say that there was one session when this session, when this packet whose finish tag F was taken for service. Obviously, among all the packets which are available, this must have been the smallest tag and that is why this packet has been taken for service.

Now, let us say that there was a session which was empty. Now, suddenly it sends a burst of packet which are back to back. Now, if you follow this method, then obviously in this case as you know that F_i^0 was 0. So, the finish tags of all these packets will start from 0. So, F_i^0 will be 0 and F_i^1 then will be L_i^1 upon ρ_i. So, that way it will increase.

Now, as a result, till the finish tag becomes equal to F, all these finish tags will be lower than F. So therefore, these packets will be served as after F. All these packets will be taken up for service and the other session which are anyway backlogged and which has the right to transmit the packets, so called; they will be now starved, their packets will not be transmitted.

What is happening? Why it is happening? This is happening because this particular session which was un-backlogged and which were absent; suddenly when it transmitted up some amount of back to back packets, then this session which was absent tries to accumulate that credits or tries to get the service which is at which it has missed. And, that should not happen. The normalized service missed by this session should not happen and that should be added to that while computing the virtual finish time. So, that is the reason, what we need to do is that we need to add we need to add on this F_i^k while computing the virtual finish tags L_i^k upon ρ_i plus max of F_i^k minus 1 v hat a_i^k.

(Refer Slide Time: 30:53)



Now, this is actually denoting that the normalized service which is missed by the session, this v hat $_i^k$, the normalized service missed by the session should be added while computing the virtual finish times. Otherwise, the sessions which are un-backlogged or the sessions which are absent get an undue advantage by accumulating the credits for the service which they have missed and then try to transmit now or trying to transmit more packets and therefore become causing unfairness to the other sessions.

So, this is really the concept that a session will never get such service; the service, the normalized service which it has missed while it was un-backlogged. So then, while asking this question that whether the self clocked fair queuing algorithm is fair or not, it was argued that we will introduce a notion of something called as the relative fairness.

What is the relative fairness? In the relative fairness, we will try to see what is the difference between the normalized services received by 2 sessions. What is the difference between the normalized services received by 2 sessions which are backlogged in the self clocked fair queuing algorithms? And, not try to compare between the normalized services received by this session in the self clocked fair queuing algorithm with respect to the fluid flow fair queuing algorithm, no.

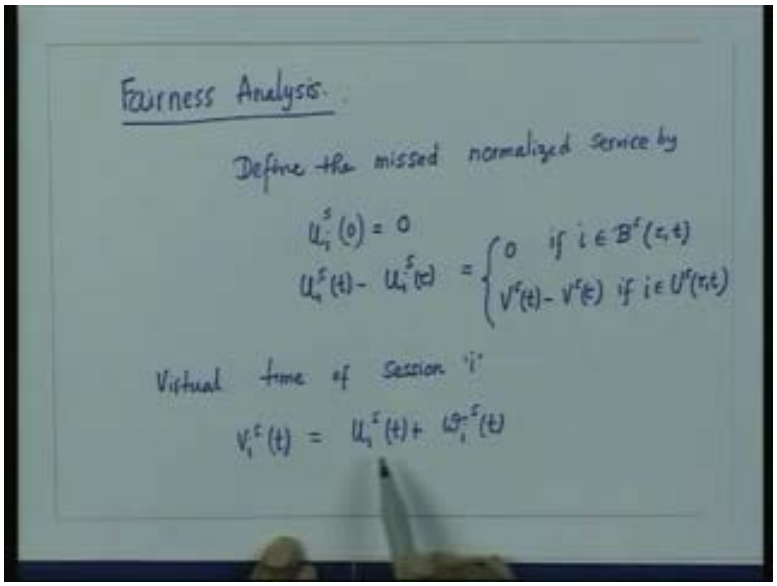
We will try to see what is the difference between the normalized services received by 2 backlogged sessions in the self clocked fair queuing algorithm itself. Now, the philosophy is that note that the self clocked fair queuing algorithm is not based upon the emulation of the fair queuing algorithm, it is not based upon… It is a packet based fair queuing algorithm and our philosophy is that we are designing a packet based fair queuing algorithms in such a manner that the normalized service received by 2 backlogged sessions, will if it remains bounded, if it remains bounded by a small amount if it remains bounded by small amount; then obviously this algorithm can be said to be

somewhat fair. That is that is how the definition of fairness is; that we should not unfairly service other sessions at the expense of the others at the expense of some other sessions.

So therefore, the idea was that we will introduce the notion of the relative fairness which will try to measure the difference between the normalized services received by 2 backlog sessions in the self clocked fair queuing algorithm itself. Now, let us see, let us try to analyze how much is this bound? So, to do that what we are trying to say is that there are normalized service opportunity, the normalized service opportunity which has been missed by a session while it was un-backlogged should also be taken into considerations. So, that is the basic philosophy of defining this relative fairness.

So, let us define few terms and then we will state the result which will tell us that how much is the relative fairness of the self clocked fair queuing algorithm. So actually, therefore, what we are saying is that it is not fair just to compare the normalized service normalized service received by 2 backlogged sessions but we should also see how much is the service opportunity which has been missed by the sessions while it was un-backlogged.

(Refer Slide Time: 34:45)



So, we will define. So, to do this fairness analysis, let us define the missed normalized service, the service opportunity which has been missed by the session - u_i^s and s, I am just denoting it to be for some packetized scheduling algorithms or any scheduling. s So, if s equal to WFQ, then it will be indicating WFQ. If s is equal to GPS, then it will be indicating the fluid flow fair queuing algorithm. If s is equal to SCFQ, then it will indicate it for the self clocked fair queuing algorithm.

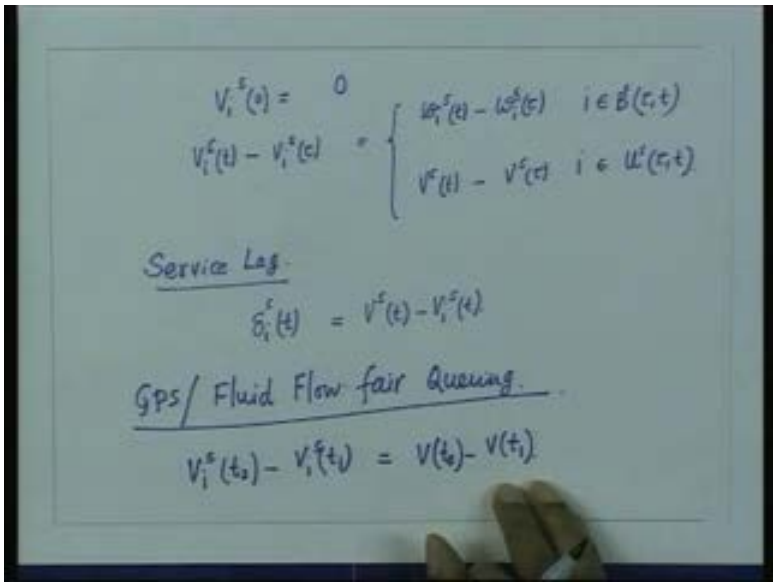
So, define u_i^s to be equal to 0 at time 0 and $u_i^s(t)$ minus $u_i^s(\tau)$ to be equal to 0 if the i'th session is backlogged in that fair queuing algorithm between tau to t and this is

equal to $v^{s}(t)$ minus $v^{s}(\text{tau})$ if this session is un-backlogged. So, what it is trying to say is that the normalized service missed by the session during tau to t, that is equal to 0 if this session was backlogged. Obviously, this session was backlogged; the session was receiving some service. However, this is equal to the difference between the system's virtual time if the session was not backlogged.

What it is trying to say that system's virtual time is keeping track of the work done by the system. So, if this session was not backlogged during an interval of tau to t, then in that case the normalized service opportunity missed by it will be equal to the increase in the virtual time, the system virtual time that has occurred. The second thing is that we now define that so obviously, the interval tau to t is any subinterval of the busy period whether this where the session i is either absent or it is backlogged.

Now, the virtual time, so we define this quantity which is called virtual time of session i. What it is trying to say? The virtual time of the session i is given by the normalized service opportunity missed plus the normalized service received by the session. So, it is the sum of these virtual time of the session i. Obviously, the session is backlogged at time t, then this quantity will be 0 and it will denote how much is the service that the user has received; otherwise, it will be equal to the normalized service opportunity missed by the session i.

(Refer Slide Time: 38:33)



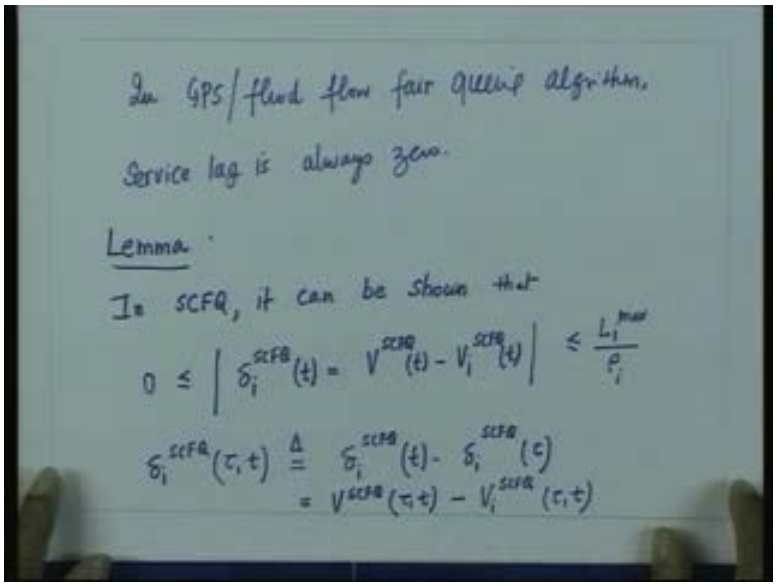
Now, one thing that should be noted is that in so session's virtual time if you see what we are trying to say is that this $v_i^{s}(0)$ is 0 at time t equal to 0 and $v_i^{s}(t)$ minus $v_i^{s}(\text{tau})$ that is the difference between the session's virtual time will be equal to the normalized service received during the interval tau to t if the session was backlogged and it will be equal to the system virtual time if this session was not backlogged. So, session's virtual time, the session's virtual time will be equal to the normalized service received if the

session is backlogged. Otherwise, it will be equal to the system's virtual time if it is not backlogged.

We define now the service lag. We define the service lag of the i'th session in the scheduling when the packet scheduling algorithm is s to be the difference between the system's virtual time and the session's virtual time. It actually indicates how far behind the session i is with respect to the systems virtual time that is the progress of the actual work done in the system. So, that will indicate how much session i is lagging behind service.

Now, let us look at the specific example of the GPS or the fluid flow fair queuing algorithm. Now, in the fluid flow fair queuing algorithms, the session's virtual time that is $v_i^s(t_2)$ minus $v_i^s(t_1)$, the difference between the session's virtual time that will always be equal to the system's virtual time for all sessions i, it will always be equal to the system's virtual time. That means what we are trying to say is that in GPS server, the session's virtual time, the session's virtual time is always equal to the system's virtual time. In the GPS server, the session's virtual time is always equal to the system's virtual time. That means the service lag, the service lag of the session i in the GPS server is always 0. So, in GPS server, the service lag of a session is always 0, the service lag is always 0.

(Refer Slide Time: 41:50)



So, the session's virtual time is always equal to the virtual time of the system in the fluid flow fair queuing algorithm and this leads to the service lag of the session, the service lag of the session being always 0 in the fluid flow fair queuing algorithm. Note that this is not true in the packetized versions of the fair queuing algorithm.

And, why the service lag is 0? Because, the session's virtual time is always equal to the system's virtual time because system's virtual time is increasing in proportion to the

normalized service received by the backlogged users and the sessions virtual time is also increasing proportional to the normalized service it receives if it is backlogged so therefore and the normalized service received by all backlogged sessions are equal. So therefore, the sessions virtual time becomes equal to the systems virtual time. Now, this is not true in the case of other packet based fair queuing algorithms like in the SCF queuing.

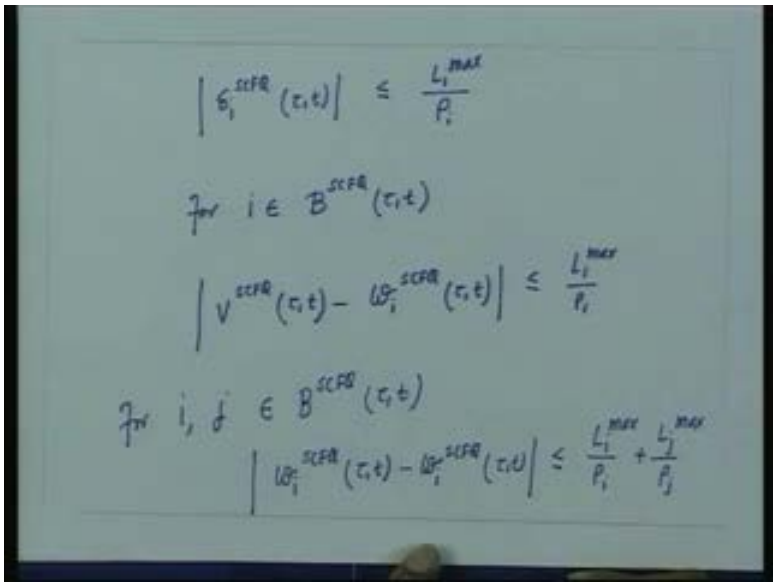
Now, it can be then proved. So, the important result that can be proved, even although we will not try to prove but let me just state at a lemma that in self clocked fair queuing algorithm; it can be shown that the service lag that is the w_i SCFQ I will write, that is the service lag which is equal to the difference between the session's virtual time and the system's virtual time which is equal to the v of SCFQ t minus v_i of SCFQ t that is the difference between the system's virtual time and the session's virtual time, this is the service lag is bounded by L_{max}, the maximum sized packet of the i'th session by rho_i.

So, the service lag in the case of SCFQ will lie between 0 and L_{max} by rho_i. So, that result can be proved. Note that in the case of a fluid flow fair queuing algorithm, the service lag is 0. In the case of self clocked fair queuing algorithm, the service lag is not 0. The session i may lag behind, one particular session may lag behind the work done in the systems but that remains bounded and that is what this result is trying to prove.

So, let us define now the differential service lag. So, one corollary of this is that w_i SCFQ which is defined with this we call as a differential service lag is a difference between delta i sorry SCFQ t minus del i of SCFQ of tau.

So, the difference between the two service lags which we are calling it to be the differential service lag which will be equal to v of, that is the virtual time between tau to t minus v of v_i between tau to t. So, it follows that the differential service lag; it will be bounded, this differential service lag will be bounded by del i SCFQ tau t - that will be bounded by L_i^{max} by rho.

(Refer Slide Time: 46:33)



This result, we can prove it from this. Now, which means for a session i which is backlogged between tau to t, if you look at this result; then note that the session's virtual time, the session's virtual time if the session is backlogged that is equal is to the normalized service received by it.

So therefore, this indicates that this result will state that because this is the differential service lag, the differential service lag is the difference between the system's virtual time t tau minus the session's virtual time tau t. Now, the session's virtual time that is equal to the normalized service received. That is what we have replaced it. Now, this means if there are 2 sessions; i and j which are backlogged, continuously backlogged in the SCFQ during this interval, then from this result we will know that the difference between the normalized service received by these 2 sessions is bounded by L_i^{max} by rho $_i$ plus L_j^{max} by rho $_i$.

Now, this is an important result which is actually trying to prove the relative fairness of the 2 sessions which are continuously backlogged during this interval in the SCFQ. What we have actually stated is that the difference between the normalized service received by 2 sessions which are continuously backlogged will be bounded by the transmission time for the maximum size packet of the i'th session plus the transmission time of the maximum size packet of the j'th session for the these 2 sessions - i and j.

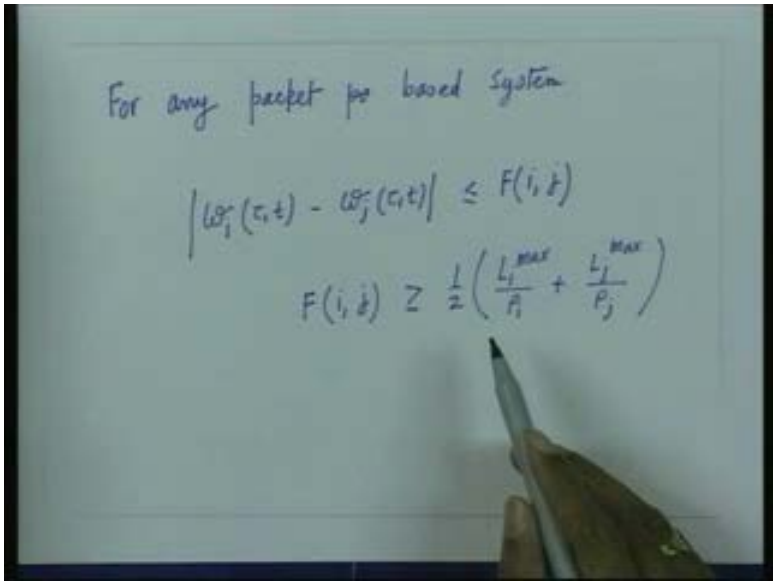
Now, this indicates how much is the fairness of the self clocked fair queuing algorithm? So, this is the result that we have proved in terms of the fairness of the fair queuing algorithm.

Now, this is obviously as I have already told you that in this case we are not trying to compare the fairness of the self clocked fair queuing algorithm with respect to the fluid flow fair queuing algorithm. This is some kind of a relative fairness notion that we have

introduced. The reason being that we are not doing, and the philosophy was that we are not doing any emulation of the fluid flow fair queuing algorithm here.

We are having a straight packet based queuing algorithm and we hoped that if the normalized service received by 2 backlogged sessions in this packet based queuing algorithms if it can be bounded by a small amount; then we have done the job, we are almost fair.

(Refer Slide Time: 50:37)



So, the question obviously is that what is possible in any packet based queuing algorithm? And, there is a result which states that for any packet based system; the normalized service, the difference between the normalized service received by the i'th i and the session j will be less than or equal to some quantity - F (i, j) which is not a function. Note that this is not a function of the interval tau to t. The times tau to t, it is just a function of the sessions i and j and where this F (i, j) will be greater than or equal to half of L_i^{max} upon rho $_i$ plus L_j^{max} upon rho $_j$. So, this quantity where F $_i$

Now, note that the self clocked fair queuing algorithm achieves this - L_i^{max} upon rho $_i$ plus L_j^{max} upon rho $_j$ - this is also we can see that the self clocked fair queuing algorithm is almost optimal. It is almost optimal by a factor of 2. So, we can say that as far as the packet based queuing algorithms are concerned; the design of a packet based queuing algorithm with the concept of this relative fairness, here we are trying to bound the difference between the normalized service received by 2 backlogged sessions, the self clocked fair queuing algorithm is almost optimal.

So, that is an important result that we have proved and interestingly turns out that the self clocked fair queuing algorithm does not require any emulation of the fluid flow fair queuing algorithm and therefore it is computationally very efficient. All you need to do is that to know, to keep knowing the progress of the work done in the system, you just need

to know the finish tag of the packet which is currently in service. So, this way we have designed a fair queuing algorithm which is quite optimal with respect to the relative fairness notion.

(Refer Slide Time: 52:56)