Broadband Networks

Prof. Karandikar

Electrical Engineering Department

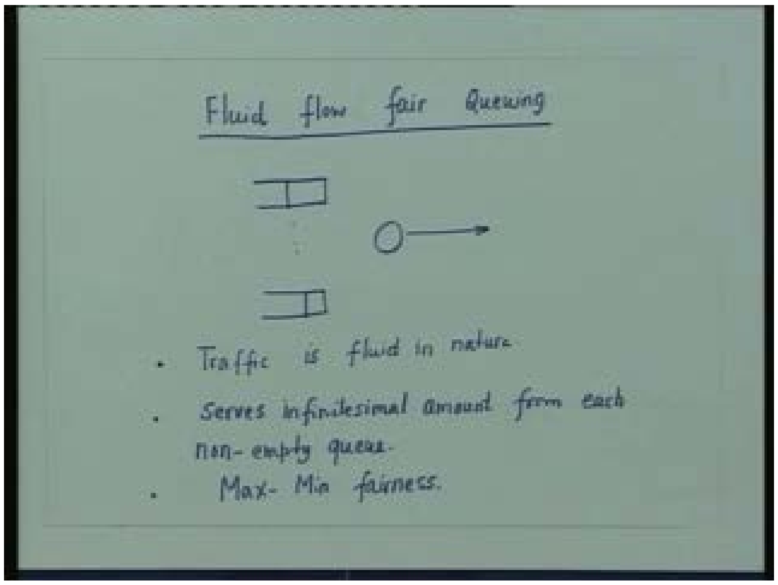
Indian Institute of Technology, Bombay

Lecture - 10

Fluid Fair Queueing & Weighted Fair Queueing

So, we were discussing about fair scheduling algorithms, the design of a fair scheduling algorithm and we have seen in the previous lectures what are the various attributes of such scheduling algorithms. So, let us today discuss how we can design such fair scheduling algorithms. So first of all to give you an idea how we can design the scheduling algorithms which results in some kind of max min fairness, we first assume that the traffic to each of the queues is fluid in nature. So, let us define the fluid flow fair queuing.

(Refer Slide Time: 1:24)



Fluid flow fair Queuing

- Traffic is fluid in nature
- Serves infinitesimal amount from each non-empty queue.
- Max- Min fairness.

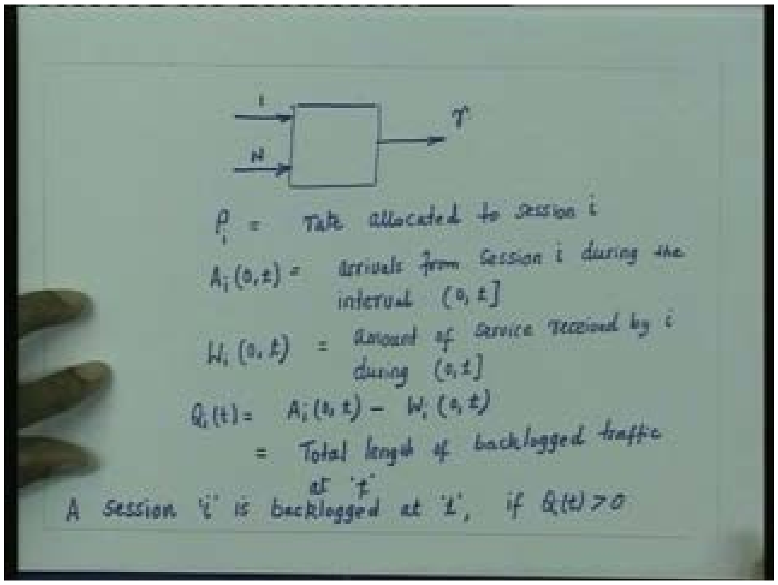
So, I will just define as fluid flow fair queuing algorithm. So, in fluid flow fair queuing algorithm, we assume suppose that there are several queues and here is a scheduler which is scheduling it onto the output link; so we assume that the traffic in each of the queues is fluid in nature, traffic in each queue is fluid in nature and then the scheduler visits each non empty queue in turn and serves an infinitesimal amount from each queue. It serves very small amount from each of the queue.

So, what we are assuming that each of these queues have the traffic which is fluid in nature and the scheduler will schedule, will serve each of the non empty queues and will serve very small amount - an infinitesimal amount from each queue.

Now, if you schedule the packets in this manner, then this will result in the max min fairness. If the different connections have different weights, then the scheduler will serve this infinitesimal amount of the fluid in proportion to its weights. So, what we are saying is that the schedulers serves an infinitesimal amount, very small amount from each non-empty queue. And, this will result in max min fairness. Now obviously, the traffic which we are transmitting it in the networks is not fluid in nature. So, this is definitely a theoretical assumption that we have made. But let us first discuss the fluid flow fair queuing with its formal definition and then we will see how we can have packetized versions of such fluid flow fair queuing.

Now, this fluid flow fair queuing we will also call it to be a generalized processor sharing or the GPS. Now, I would like to give you a formal definition of these fluid flows fair queuing. Qualitatively you have understood that each of the queues has a traffic which is considered fluid in nature and the server will serve an infinitesimal amount from each non-empty queue in turn and by this we will have some kind of max min fairness. So, let us have a formal definition of the fluid flow fair queuing.

(Refer Slide Time: 5:05)



ρ_i = rate allocated to session i

$A_i(0,t)$ = arrivals from session i during the interval $(0,t]$

$W_i(0,t)$ = amount of service received by i during $(0,t]$

$Q_i(t) = A_i(0,t) - W_i(0,t)$

= Total length of backlogged traffic at 't'

A session 'i' is backlogged at 't', if $Q_i(t) > 0$

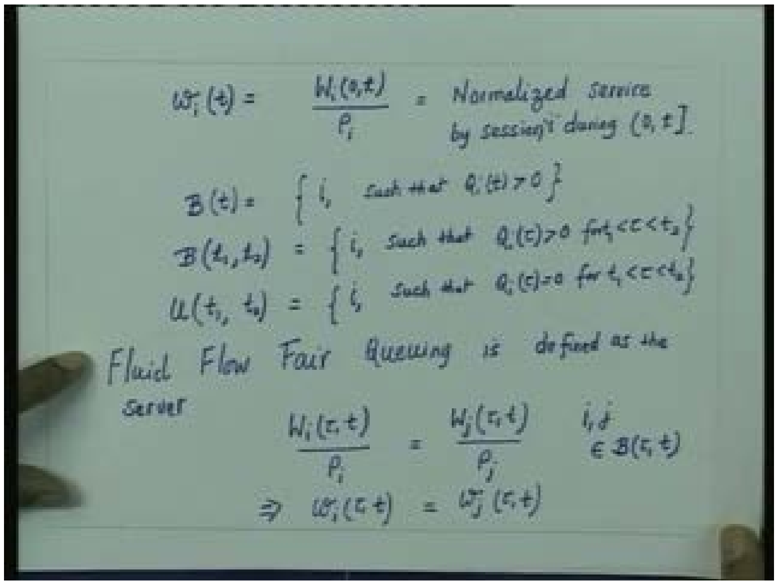
So, formally let us say that we are having as we have said N queues. So, this is let us say, a fluid flow fair queuing scheduler. So, let us say that there are 1 to N queues and this is an output link with the capacity r. Now, let us say that rho i is the rate which has been allocated to session i. Now, let us define A $_i$ (0, t) to be the arrivals, traffic arrivals from session i during the interval 0 to t. Now, if it is a packetized arrival if it is a packetized arrival, then the arrival time of the packet will be determined by the reception of its last bit. So, for a packetized traffic, this arrival function A $_i$ will be considered as a step function, the staircase functions.

We will assume that some arrivals have occurred only when the last bit of that packet has entered the queue if it is a packetized traffic. So, now let us define W $_i$ (0, t) to be the amount of service received by session i during 0 to t. So, this is the amount of the traffic fluid which has been served and W $_i$ (0, t), we consider it to be a continuous function; whether the arrivals are packetized or whether the arrivals are fluid.

Now, let us define Q $_i$ (t) to be A $_i$ (0, t), that is the total number of arrivals that has occurred minus the total fluid that has been served. So, then what is Q $_i$ (t) in that case? The Q $_i$ (t) will be nothing but the total length of the backlogged traffic at time t. So, Q $_i$ (t) will denote as the total length of backlogged traffic at time t. So, we say that a session is backlogged; so a session i is backlogged at time t if Q $_i$ (t) is backlogged at time t if Q $_i$ (t) is greater than 0. Now, W $_i$ (t) is the amount of traffic fluid that has been served. A $_i$ (0, t) is the traffic arrivals that have occurred during 0 to t.

So, A $_i$ (0, t) minus W $_i$ (0, t); this is some sort of queue length, that is the total length of the traffic which is lying in the queue of session i. Note that each of these sessions have independent different queues. So, this is the total length of the backlogged traffic at time t. Now, we say that this session, it is backlogged at time t if Q $_i$ (t) is greater than 0.

(Refer Slide Time: 9:00)



$$w_i(t) = \frac{W_i(0,t)}{\rho_i} = \text{Normalized service by session } i \text{ during } (0,t]$$

$$B(t) = \{ i, \text{ such that } Q_i(t) > 0 \}$$

$$B(t_1,t_2) = \{ i, \text{ such that } Q_i(\tau) > 0 \text{ for } t_1 < \tau < t_2 \}$$

$$U(t_1,t_2) = \{ i \text{ such that } Q_i(\tau) = 0 \text{ for } t_1 < \tau < t_2 \}$$

Fluid Flow Fair Queuing is defined as the server

$$\frac{W_i(\tau,t)}{\rho_i} = \frac{W_j(\tau,t)}{\rho_j} \quad i,j \in B(\tau,t)$$

$$\Rightarrow w_i(\tau,t) = w_j(\tau,t)$$

Now, we define something called as the normalized service which we define with respect to let us say small w $_i$ (t) is equal to W $_i$ (0, t) divide by rho i. So, this is actually equal to the normalized service by received by session i during 0 to t interval. Now similarly, we define as we have said that a session i is backlogged at time t if Q $_i$ (t) is greater than 0, so let us define the set of sessions which are sort of backlogged.

So, we say that B (t) denotes the set of all such sessions such that Q $_i$ (t) is greater than 0. That means B (t) is that set which includes all the sessions which are backlogged at time t. Similarly, we define B (t_1, t_2) happens to be the set of all such sessions such that Q $_i$ tau is greater than 0 for tau lying from t_1 to t_2. That means B (t_1, t_2) consists of all the sessions which were continuously backlogged during t_1, t_2. That means there Q was greater than 0 during the interval t_1 to t_2.

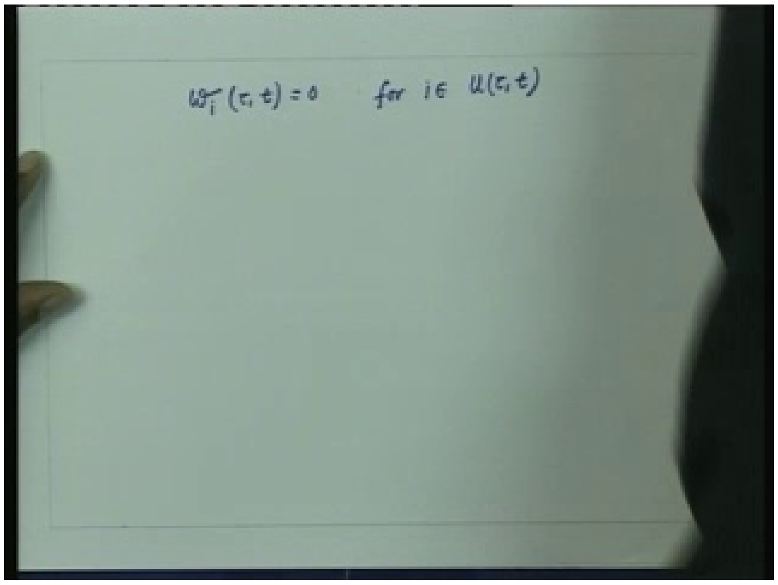
Similarly, let us define as U (t_1, t_2) which are sessions which are unbacklogged as consisting of all i such that the Q $_i$ (t) is 0 for t_1 less than tau less than t_2. That means during the interval t_1, t_2, the queues were empty. So, such sessions, we can call it to be the sessions which are absent sessions which are continuously backlogged. Then the important definition of the fluid flow fair queuing is fluid flow fair queuing is defined as the server as the server for which the normalized service received by 2 sessions let us say i and j which are continuously backlogged during an interval is equal. That means, if you consider 2 sessions, any 2 sessions; during an interval and if during this interval, if these 2 sessions are continuously backlogged, then the normalized service received by these 2 sessions will be equal.

So formally, we will define as the fluid flow fair queuing is defined as is this is defined as the server for which W $_i$ tau t divide by rho I, so this is the normalized service received by the session i during the interval tau to t. This will be equal to W $_j$ tau t divide by rho j. This is the normalized service received by the session j during the interval tau to t and both i and j sessions are such that they are continuously backlogged during the interval tau to t.

So, because we have defined these B (t_1, t_2) to be consisting of those sets which are continuously backlogged during the interval, t_1 t_2; so therefore, we have this B tau t is consisting of those sets which are continuously backlogged during the interval tau to t. So, we take any such 2 sessions and for this, we can show that the normalized service received by these 2 sessions will be equal.

In other words, we can also say that W $_i$ tau t, this is denoting the normalized service will be equal to W $_j$ small W $_j$ tau t and the sessions which are during this interval tau to t are the absent, they will not receive any service. So, for such sessions, the normalized service is actually 0. So, we can say that for the sessions which are absent, the normalized service is 0.

(Refer Slide Time: 14:04)

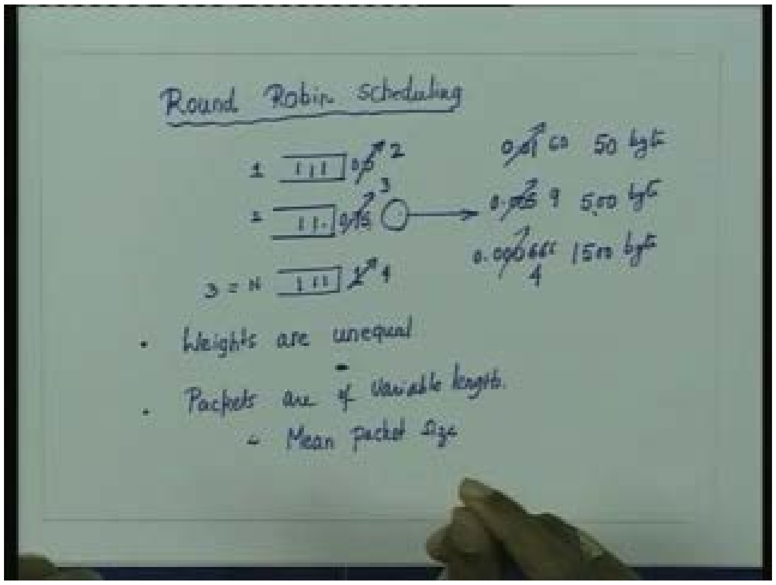


$w_i(\tau, t) = 0 \quad \text{for } i \in U(\tau, t)$

That means W $_i$ tau t is 0 for i belonging to u tau t which comprises of all those sessions which are absent during the interval tau to t. So, this is the formal definition of the fluid flow fair queuing where we say that the normalized service received by the 2 sessions i and j which are continuously backlogged during the interval tau to t that is equal. On the other hand, so how do we serve this?

The traffic, we assume it to be fluid in nature and then the server will visit each non-empty queue in turn and it will try to serve an infinitesimal amount of fluids from each of these queues. So, having defined the fluid flow fair queuing in its formal way, now our question is however that the traffic is not fluid in nature but the traffic is packetized in natures. So, how do we in that case ensure the fairness for a practical case where the traffic is packetized in natures? So, let us say that we implement a round Robin scheme. So, we call it to be let us say that we implement round Robin scheduling algorithms and then we will try to see whether it is fair or not.

(Refer Slide Time: 15:32)



Round Robin scheduling

- Weights are unequal
- Packets are of variable length.
  - Mean packet size

We will see what is wrong with the round Robin scheduling algorithms. So, now what is in the round Robin scheduling algorithms? We have again these 1 to N queues. Traffic is of course packetized in nature. So, round Robin scheduling algorithms is simple; you serve each packet from the queue and send it onto the output link.

Now, the packets are of the fixed length. Now, note that in a packetized traffic, packet is a smallest unit that can be transmitted. In a fluid traffic, you can serve an infinitesimal amount of fluid. It is not possible to do in the packetized traffic. So, packet is the smallest unit. Now, if these packets are of the fixed length and if you are using a round Robin scheduling algorithm, then you serve one packet from the first queue, serve another packet from the second queue and so on. This way you will result in fairness and this is the best that you can do.

The problem arises when the packets are of variable length. Now, what do we do if the packets are of variable length? If the packets are of the variable length or else if these connections have different weights; in either of these 2 situations, we need to see how we can implement the round Robin algorithms.

So, first I will assume in the round Robin algorithms that the packets are of the different length or the weights are unequal. So, we say that let us say that the first case is that the weights are unequal. An example, let us say that there are 3 connections. So, N is equal to actually 3 here. So, this connection for example … So, this is a second connection. So, let us say this connection has a weight of 0.5. This connection has a weight of let us say 0.75 and this connection has a weight of 1. So, the weights are unequal. What do we do?

So, the weights are unequal. We normalize the weights in such a manner that they become integers. So, what do we do? We normalize these 2 weights, these 3 weights: 0.5, 0.75 and 1 such that they become integers. So, in that case, they become 2, 3, and 4. So, how do we implement round Robin algorithms?

We serve 2 packets from this first queue, then the server will serve 3 packets, from the third queue and second queue and then the server will serve 4 packets from third queue. So, that way round Robin algorithm will be implemented. This will still be fair this will still be fair if the packets are of fixed length. If all the packets are of fixed length, then of course each of these queues were having different weights. So obviously, we were serving different packets.

Here, we were serving 2 packets, here we are serving 3 packets and in this, from this queue, we are serving 4 packets. The problem is suppose if the packets are of variable length; so what happens if the packets are of variable length? Now, if the packets are of variable length, then obviously there is a problem. So, we in order to implement the round Robin scheduling algorithms, we need to know the mean packet size in advance and then we will normalize this weights with respect to the mean packet size.

So, if the packets are of variable length, we need to know the mean packet size. Now, suppose we know the mean packet size; let us say we know the mean packet size of the connection 1, let us say the mean packet size of the connection 1 is something like 50 bytes, the mean packet size of the connection 2, let us say is the 500 bytes and the mean packet size of the third connection let us say is the 1500 bytes. So, these are the mean sizes of the 3 queues – 50 bytes, 500 bytes and 1500 bytes.

So, we normalize theses rates which were 0.5, 0.75, 1 with respect to these we divide them and what really we get by 0.5 divided by 50, we get the weight of 0.01, 0.75 divided by 500 we get the weight of 0.005 and of course 1 divided by 1500 we will get something like 0.00066 and so on.

So now, we again make these rates to become integers and for that we will, this will turn to 60, this will become 9 and this will become 4. So, as a result what do we do? This queue has a less mean packet size of 50 and at a weight of 0.5 we serve 60 packets from this queue first and then we serve 9 packets from the second queue and then we serve 4 packets from the third queue.

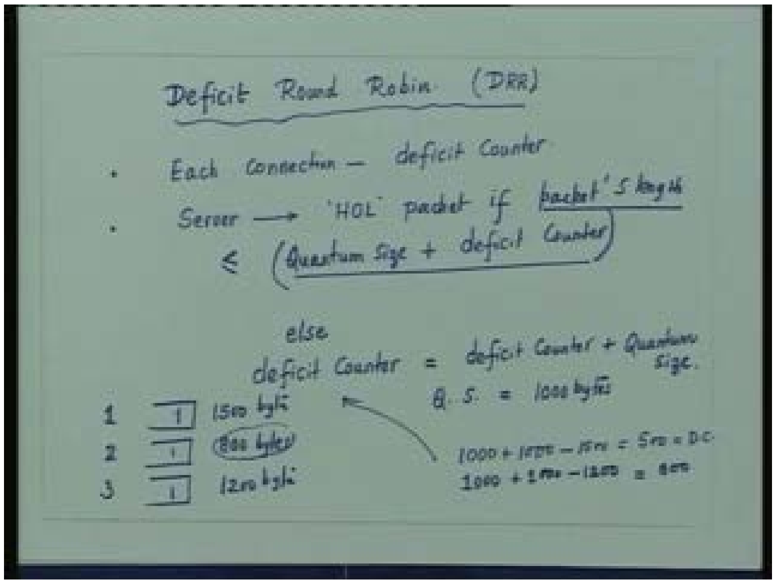
So, this way we can try to maintain the fairness. So, now this scheme therefore has the 2 disadvantages. One is we need to know of course the mean packet size in advance and then only we can determine how many number of packets we need to serve from each queue by normalizing the weight with respect to mean packet size. But even if we know the mean packet size, the scheme is actually unfair over shorter time scales.

Just to give you an example how the schemes will be unfair over a shorter time scale, in the sense that in this case, in this particular example that we had considered, where the scheduler has to serve 60 packets onto this output link from this queue; now, when the transmission of these 60 packets is going on onto the output link, we are not serving anything from queue 2 and queue 3 and therefore over the interval for which the transmission is going on for the 60 packets, the scheme becomes unfair to 2 and 3 and if the transmission time is little bit large, then the interval over which this unfairness also increases.

Now, note that in the fluid flow fair queuing, we were serving infinitesimal amount of fluid from each of these queues. So, the scheme was really fair over an instant of time. Of course, when the traffic is packetized in nature, it is not possible to serve a bit from each of these queues or an infinitesimal amount from each of these queues. But even then we would like to reduce the unfairness interval, the interval over which the algorithm is unfair that should be reduced.

Now, in this case we have find that it could be unfair over a larger interval of time as well. So, we are therefore we need to therefore look for different more algorithms. A variation of this round Robin scheme could be or the weighted round Robin scheme that was suggested is called deficit round Robin. So, let me just explain what is a deficit round Robin algorithms.

(Refer Slide Time: 23:44)



Deficit Round Robin (DRR)

- Each connection – deficit Counter
- Server → 'HOL' packet if packet's length ≤ (Quantum Size + deficit Counter)

else
deficit Counter = deficit Counter + Quantum Size.

Q. S. = 1000 bytes

1 1500 bytes
2 800 bytes
3 1200 bytes

1000 + 1000 – 1500 = 500 = D.C.
1000 + 1000 – 1200 = 800

So, this is a deficit round Robin scheme. So, deficit and also called as DRR. Now, in the deficit round Robin what you are doing is that each connection has a deficit counter. So, each connection has some kind of a deficit counter which is initially 0. So, this could be initialized to 0. And then, server will serve the packet. The first packet serves server serves the first packet; we call it to be head of the line, HOL. The first packet from the queue, it serves if the packets length is less than or equal to something called as quantum size plus the deficit counter value.

So, if the packet length is less than or equal to the quantum size plus deficit quantum, the deficit counter; then only we serve the packets. Otherwise, the quantum is added to the deficit counters; else the quantum is added to the deficit counter, else the deficit counter becomes equal to whatever the value was - the old deficit counter plus this quant size. So, this way actually, the principle of the deficit round Robin scheme is to avoid the difficulty of having to know the mean packet sizes in advance in the weighted round Robin or in the round Robin scheme that we had just considered. So, the deficit round Robin scheme works on the principle that we do not know the mean packet size in advance. So, we maintain a deficit counter and then try to maintain the fairness among the flows. Just to give you the example; again the previous example of 3 queues we will consider.

So, let us say that there is a queue; there is a connection 1, 2, 3. Now, the connection 1 has a packet and the connection 2 has another packet and connection 3 has another packet. This packet is of let us say 1500 bytes and this first packet is of 800 bytes in the connection 2 and the third packet is of 1200 bytes.

Now, let us say that quantum size that we had considered here, so let us say that the quantum size that we have considered is 1000 bytes and the deficit counter initially is 0. So, deficit counter initially was 0; so now if you see 1500 byte, the length of this packet is less than the quantum size plus the deficit counter. So therefore, this packet is not served. This packet is not served and this quantum size is now added to the deficit counter. So, the deficit counter becomes actually 1000.

Now, you go to the second packet, second queue; 800 bytes is definitely less than the 1000. So therefore, this packet is served. So, this packet is served. Third you come, here it is 1200 bytes. Again, 1200 bytes is less than 1000. So therefore, you do not serve this packet. So, now you come so, in the first round, in the first you have served only this packet of 800 bytes. Now, in the second round, then when you come here in 1500 bytes; your deficit counter was how much? 1000, because that was earlier, it was added. So, in the second round you had here 1000 was the deficit counter. You add again 1000 into the quantum size that becomes how much? 2000. So, obviously, 1500 is less than or equal to the quantum size plus the deficit counter that is it is less than that.

So therefore, this packet is served and so the deficit counter now becomes equal to how much? 500. So, the deficit counter 0 has become 2000 minus 1500 which is equal to 500. So, this is the new deficit counters value, so new deficit counter value. Of course, this was of course 0, so there was no queue, there was no packet.

Now, we come to third queue which has 1200 bytes of packets. The deficit counter was 1000 here again, add the quantum 1000 no 2000 minus 1200 and that becomes equal to 100 800. So, this will be your new deficit counter value. The recommended quantum size, as you can see each time when you are adding the counter is actually equal to the maximum packet size because otherwise you can see here that if the maximum packet size was 3 connection itself was something like 2000 bytes; then initially itself when the quantum size is 1000 bytes, none of these packets would have been served. It was possible that none of these packets would have been served. So therefore, the recommended value the recommended value of the quantum size is always actually equal to the maximum packet length.

So, the principle of the deficit round Robin is that you serve the packet only if the packet length is less than or equal to your credit. So, it is actually maintaining some kind of credit. So, if your packet length is less than or equal to the credit that you have accumulated, then you serve the packet. If it is greater, then you accumulate that credit and then you will be served only in the next round.

Of course if a connection is empty, if a queue is empty, then your credit will be initialized to 0. Otherwise, you will keep on accumulating the credits for having remained absent for a long time. So, to prevent this you can initialize the deficit counter to 0 once your queue becomes empty. Again, the disadvantage of the deficit round Robin is unfairness over a shorter interval of time, over a smaller interval of time. That is actually the disadvantage.

So, now we have seen 3 kinds of scheduling algorithms. One is the fluid flow fair queuing algorithm where we assume the traffic to be fluid in nature and the server serves an infinitesimal amount of fluid from each of queues and then maintains the max min fairness. We then see how we can have packetized versions of the scheduling algorithms. One simple scheme was round robin algorithms where you serve the queue in a round Robin fashions; pick up the packets from each of the queues and serve the packets in a round Robin manners.

Obviously, if the different connections have different weights, then you need serve them in proportion to their weights. So, you normalize these weights so that they become integers and accordingly serve as many numbers of packets from each of the rounds. Now, this scheme can be considered as fair, provided the packets are of fixed length. If the packets are of variable length, then we need to normalize these weights with respect to the mean packet sizes and that is where the disadvantages of this scheme comes in because then we need to know the mean packet sizes in advance if you want to be fair.

So now, to avoid this problem, to address this problem that we need to know the mean packet sizes in advance. The deficit round Robin scheme was proposed and the deficit round Robin scheme what really you are doing is that you are trying to accumulate the credits and then serve the packets only if you have enough credits.

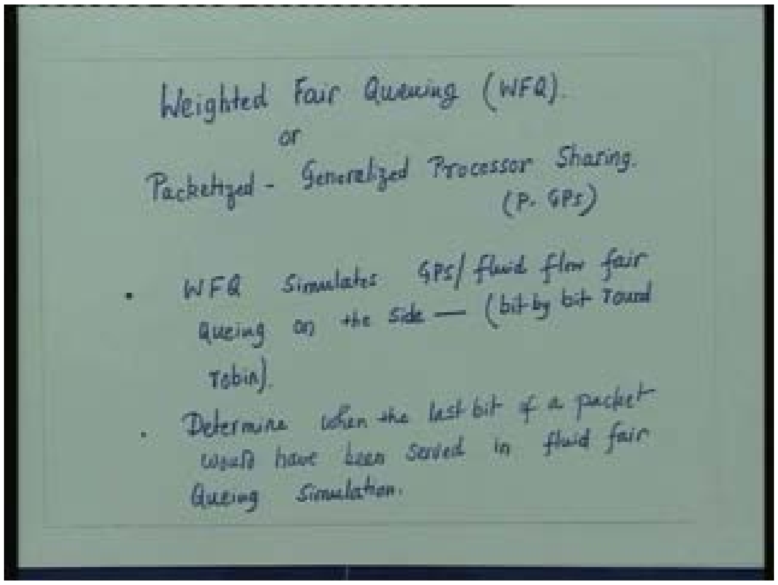
Now, this avoids having to know the mean packet sizes in advance. The only thing you need to you need to ensure that you will definitely serve even if the packets is of a maximum size, the quantum size, the value of the quantum size should be at least equal to the maximum packet size. So, this way you can ensure that you will always serve a packet even if it is of a maximum size.

Now, even though the round Robin algorithms and the deficit round Robin algorithms try to maintain fairness, even then they cannot be called perfectly fair in the sense that they are still unfair over a shorter time scales. Over a short time scales, they are still unfair. If you consider over a long time interval, then they could be fair because over a long time you would have seen that, you would have served the packets from each of the queues in proportion to their weights. But on a shorter time scales, they are still unfair. They are still unfair in the sense that in fluid flow fair queuing and fair, at every instant of time but in this packetized versions you are not still fair.

So, the question really is that can we design packetized versions of the fair scheduling algorithms which approximate the fluid flow fair queuing in terms of the fairness? Can we design, can we do that? Now to do this, packetized versions of the fluid flow fair queuing was proposed which is also called as weighted fair queuing, WFQ or which were also called as packetized generalized processor sharing or PGPS.

So, let us study what is the weighted fair queuing scheduling algorithm.

(Refer Slide Time: 35:30)



Weighted Fair Queuing (WFQ).
or
Packetized - Generalized Processor Sharing.
(P. GPS)

- WFQ Simulates GPS/ fluid flow fair queing on the side — (bit-by bit round robin).
- Determine when the last bit of a packet would have been served in fluid fair Queing Simulation.

So, we have weighted fair queuing WFQ or it is also called as packetized generalized processor sharing. Now, the general idea of the WFQ or the packetized generalized processor sharing is that it is also abbreviated as PGPS where the fluid flow fair queuing was called as GPS. So, what do we do is that WFQ actually it simulates fluid flow fair queuing on side. It simulates the GPS or the fluid flow fair queuing on the side. This is modeled as doing bit by bit round Robin service.

So, what we are saying is that in a fluid flow fair queuing in a fluid flow fair queuing, we were saying that we will serve infinitesimal amount of fluid from each of the queues. Now, what do we do? We say that suppose we serve the bit, we serve these queues in a round Robin manner as bit by bit round Robin that means you serve bit one from the first queue, then bit one from the second queue; so, in each of these bits.

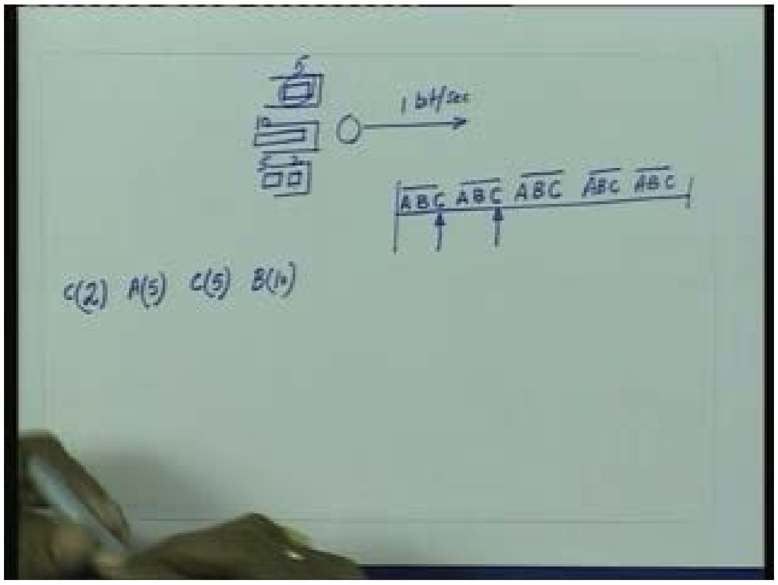
Obviously, in practice you cannot serve in this manner. But we are saying that suppose we are having a simulator which is simulating this bit by bit round Robin manner, then what a WFQ does is it determines when the last bit of a particular packet, when the last bit of a particular packet would have been served, determine when the last bit of a packet would have been served in the fluid flow fair queuing, in your fluid flow fair queuing simulation.

So, if you know that when if your simulation if you determine when the last bit of the packet would have been served, you call this as the virtual finish time. This is not the actual finish time, call it virtual finish time. WFQ, in your actual transmission, in actual WFQ, it will serve the packets in the increasing order of virtual finish time. You serve the packet in the increasing order of the virtual finish time.

Let me just again explain this. What we are doing is that in WFQ let us say that there are 1 to N queues and you want to implement weighted fair queuing; what you do is on side by side, you have a GPS calculator simulating this fluid flow fair queuing which is being implemented as serving round Robin manner but bit by bit round Robin manner.

So now, if the traffic which you have fed to your actual WFQ scheduler, you also feed it through your simulator, GPS simulator and that GPS simulator is serving the packets in a bit by bit round Robin manner. Now, you determine in the GPS simulator, when the last bit of a particular packet would have been served if you are serving in a bit by bit round Robin manner? Call this as the virtual finish time and then in your actual WFQ, serve the packets in the increasing order of the virtual finish times.

(Refer Slide Time: 40:25)



Just to give you a simple example, let us say that you have 3 queues. Let us say that your output link capacity is 1 bit per second. So, let us say this packet has 5 bits packet, this packet has 10 bits packet, this packet has 2 bits and 5 bits. So, this packet is 5 bits, this packet is 10 bits and this packet has 2 bits and 5 bits.

Now, if you implement in a bit by bit round Robin manner, so what will happen? One bit you will take, it will get transmitted; another bit you will take it, another bit you will take it. So, if you, if you really see in a time instant, the first bit of this gets transmitted. Let us say as A, then the first bit of the second this has to be transmitted. So, this becomes B and the third bit of this has transmitted, so this becomes C. So, 3 time units. Then again A is transmitted, then again B is transmitted and this C is transmitted.

Now, note that this at this point, these are ABC, so there are only 2 bits. So at this point, this packet has been transmitted. So, this is the first packet to get transmitted. Now, this has happened in the second round. First round comprises of 3seconds, the second round comprises of 3 seconds. So, here the second round finishes. So, we call this as a virtual finish time. So, the virtual finish time of this pack, the queue number 3, this first packet is 2.

Then again, now what happens is that you will transmit A again, B again and C again. But this is now this packet. Then, it will go on things like this. So, the virtual finish time of this… so this is like in the third round, this will happen ABC and again fourth round, it will happen ABC and in the fifth round, again it will happen ABC. So, if you see the fifth round, this packet would have been served. So, this is for the C's. So, first is like C's packet having 2, then A's packet having 5 bits and then after that you will serve again C's packet having 5 bits and again B's packet having 10 bits. So, this way…

So now, what happens is so first in actual WFQ, you will serve this packet first. So, the server will pick up this packet. Then, it will pick up this packet, then it will pick up this packet and after that it will pick up the last packet. So, you need to determine the virtual finish times of the packets that means you need to determine when the last bit of a packet would have been served in the fluid flow fair queuing simulations and that you call it to be the virtual finish time and serve the packets in the increasing order of the virtual finish times.

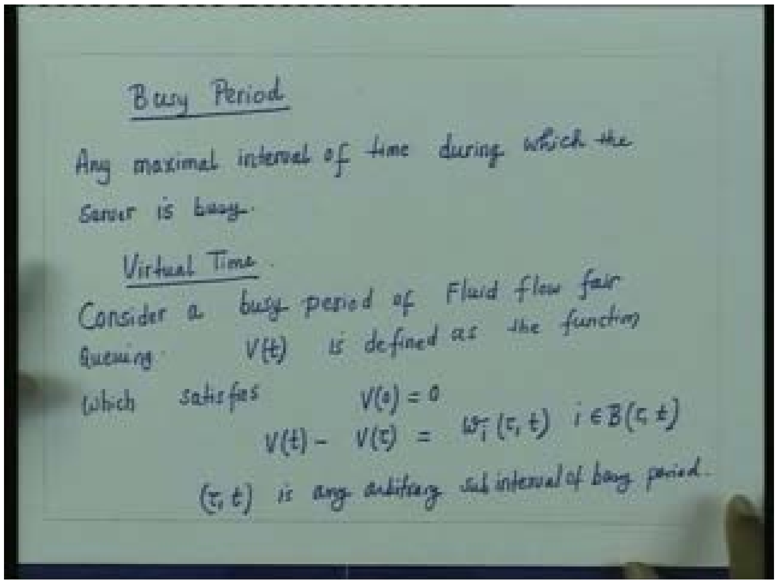
Now, in this particular example, it was very easy to visualize because what we assumed is that all the packets are there and then after that no further arrivals are there. But when the further arrivals are there, the situation is little more complicated and we need to actually determine the virtual finish time of the packet. So, we would therefore like to formalize this. I am just to give you an example here again; note that we had assumed here that this packet of length 2 and this packet of length 5. They are available at time 0 only.

Now, it is possible that this packet of length 2 and this packet of length 5 arrives only after sometime in the actual traffic arrival systems. But if you are doing the scheduling in a bit by bit round Robin manner, it is possible that even then this packet would have left earlier than this packet. That was possible. But see, this packet has not arrived you might have decided to transmit another packet on the queue.

So therefore, we need to carefully determine what will be the virtual finish time of a packet and towards that we need to formalize the definition of computing the virtual finish times. So, let me just formalize that so that we then know how to determine this virtual finish times and how to serve the packets. I will again explain that with an example so that this thing becomes cleared.

So, let me just define some few definitions for formalizing the weighted fair queuing. So, we define this definition in terms of a virtual time.

(Refer Slide Time: 45:46)



Busy Period

Any maximal interval of time during which the Server is busy.

Virtual Time.

Consider a busy period of Fluid flow fair Queuing V(t) is defined as the function which satisfies

$$V(0) = 0$$

$$V(t) - V(\tau) = w_i(\tau, t) \quad i \in B(\tau, t)$$

(τ, t) is any arbitrary sub interval of busy period.

So, but first we define the busy period of a server. So, busy period we define as any maximal interval of time during which the server is busy without any interruptions. This is called a busy period. Now, note that for the work conserving scheduling disciplines, since both the fluid flow fair queuing as well as the weighted fair queuing, both of them are work conserving; their busy periods will actually coincide. So, that is what the definition of the busy period.

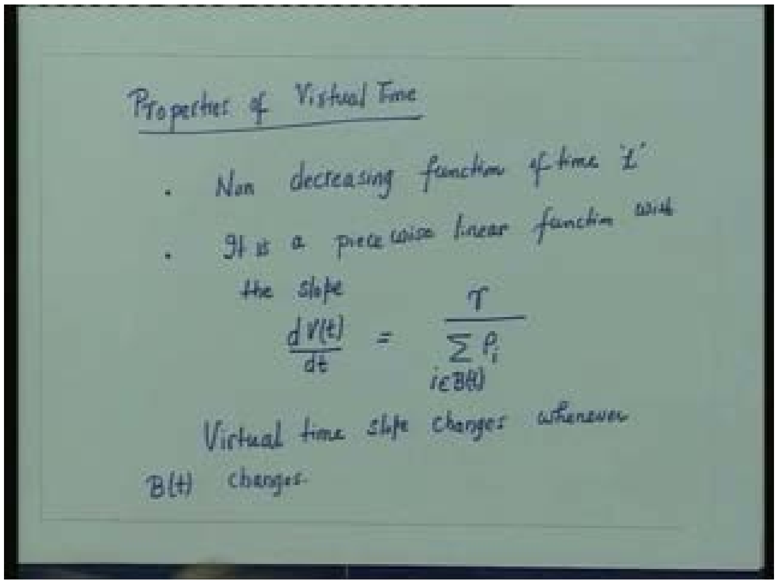
Now, we look at the definition of what we called as the virtual time. Now, consider a busy period of the fluid flow fair queuing system of a fluid flow fair queuing system. Now, let us define V (t) as the virtual time. Now, this is defined as the function which satisfies is defined as the function which satisfies the following property; V_0 is 0, so at time t, the initial time t_0 and V (t) minus v tau will be equal to W $_i$ tau t for i belonging to the backlogged sessions during the interval tau to t.

So, what we are saying is that the virtual time of the system the virtual time of the system increases in proportion to the normalized service received by any backlogged session. That is what it is. Now, note that the normalized service received by sessions which are all backlogged during an interval will be all same. So, the virtual time of the system is actually is indicative of what is the work done by the server.

So, the virtual time increases in proportion to the normalized service received by the sessions which are backlogged during an interval. So, let us now so this interval tau to t is any sub-interval for this busy period. So, tau to t actually lies in the busy period is any arbitrary sub interval of busy period. Now, having defined like this, the busy period; we would like to see what is the definition of this, what are the properties of this virtual time.

Now in general, the virtual time is a non decreasing function of time t. It is in general, it is a non decreasing function of time t because the virtual time will never decrease of the system as it is a representative of the total work done by the systems. So, the way we have defined here the virtual, the manner in which we have defined here is actually a monotonically increasing function of the time t. Second thing is it is a piece wise linear function.

(Refer Slide Time: 50:21)



Properties of Virtual Time

- Non decreasing function of time 't'
- It is a piece wise linear function with the slope

$$\frac{dV(t)}{dt} = \frac{r}{\sum_{i \in B(t)} \rho_i}$$

Virtual time slope changes whenever B(t) changes.

So, the properties of the virtual time is a non decreasing function of time t. Properties of the virtual time, it is a non decreasing function of time t and it is a piece wise, it is a piece wise linear function and this with the slope, this has the slope that is d V(t) by d(t) this is equal to r upon summation of rho i where rho i belongs to the sessions which are backlogged.

So basically, what does it mean is that virtual time will change its slope whenever the set of backlog session virtual time slope changes whenever the set of backlogged session whenever B(t) changes, whenever B(t), a set of backlogged session changes.

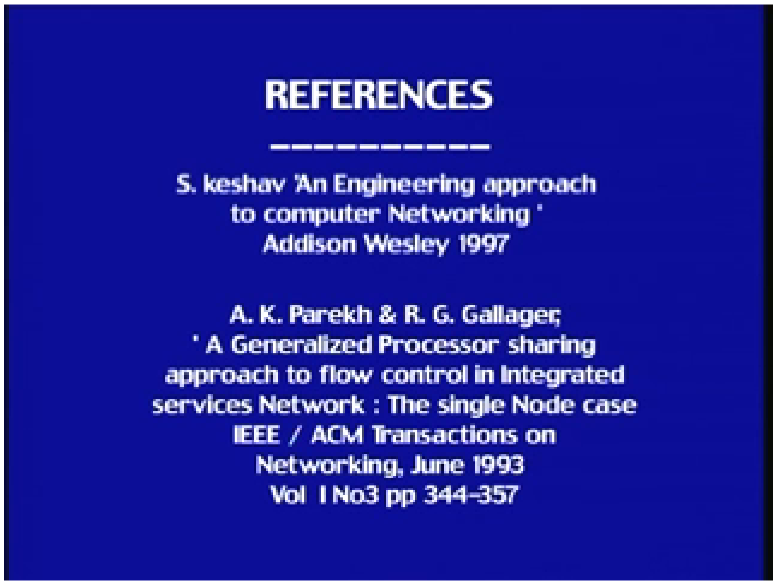
So, again let me just explain. What we are assuming right now is we are assuming that it is a fluid flow fair queuing system fluid flow fair queuing system where we are implementing bit by bit round Robin fashions. Now, we would like to keep track of the actual work done in the system and in the fluid flow fair queuing system, actual work done in the fluid flow fair queuing system and that is why we define certain quantity which we will call it to be virtual time. Now, we say what are the properties of this virtual time. The virtual time is initialized to 0 and after that the virtual time increases in proportion to the normalized service received by the sessions which are backlogged during any arbitrary sub interval of the busy period.

So, it increases and the slope of this increase - the virtual time is actually equal to, is inversely proportional to is inversely proportional to the number of sessions which are backlogged. So, if this number of sessions which are backlogged increases, then obviously the slope decreases. If the number of sessions which have become unbacklogged which have gone absent; then the slope in that case will increase, otherwise the slope decreases. So, the slope of this virtual time is inversely proportional to the number of backlogged sessions and we can show that this slope is actually given by the output link capacity r divide by the sum of all these rho i's were summed over the set of backlogged sessions and the rho i's are the rates which are allocated to the session i.

So, we will try to first prove this result and after proving this result, then we will try to relate it to how to derive the virtual finish time numbers and then determine the virtual finish time so that the packets can be served in the increasing order of their virtual finish times in the and the actual implementation of the weighted fair queuing. And, we will also illustrate with an example what is the importance of computing this virtual time in this manner.

So, that discussion will tell us that how we can keep track of the actual work done in the fluid flow fair queuing systems and relate it to the scheduling of the packets in the actual packetized versions so that we are as fair as or as fair in as the or, or at least try to be as fair as the fluid flow fair queuing or generalized processor sharing or the GPS algorithms.

(Refer Slide Time: 55:10)



REFERENCES

S. keshav 'An Engineering approach to computer Networking' Addison Wesley 1997

A. K. Parekh & R. G. Gallager, ' A Generalized Processor sharing approach to flow control in Integrated services Network : The single Node case IEEE / ACM Transactions on Networking, June 1993 Vol I No3 pp 344-357