Real-Time Systems
Prof. Dr. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

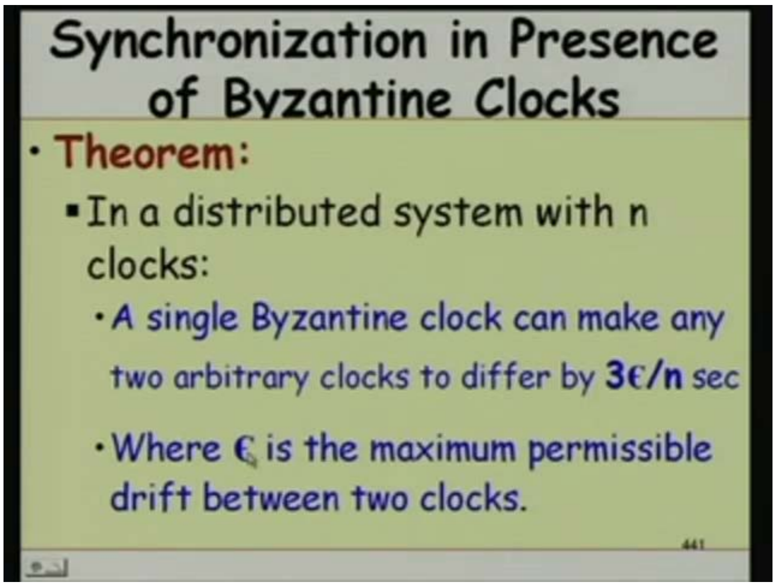
Lecture No. # 20
Internal Clock Synchronization in Presence of Byzantine Clocks

So, let us get started from what we were discussing last time. Last time we were discussing about clock synchronization, a distributed real-time system. And we had said that there are two main types of synchronization that can be done; one is external clock synchronization that is synchronized with respect to an external clock, and internal clock synchronization where some internal clock values are used to synchronize the clocks. And then, we had seen that the external clock synchronization is expensive, not used very commonly. And in the internal clock synchronization, we had seen two types; one is based on a master clock or time server keeps on periodically transmitting its time in the other clock set their time.

But that has the problem of single point failure, if what if that - know the that clock fails. So, a common implementation of clock synchronization is the distributed clock synchronization, where the clocks they keep on transmitting their own values, and its clock averages and sets its own value. And then we had seen about the effect of bad clocks; seen, bad clocks not very difficult to handle. It can be eliminated from computation.

But the real difficulty was Byzantine clocks, which shows one value to one clock and another value to another clock. And then, we are trying to we are trying to derive some results on the impact of a Byzantine clock on the value time value computed by all clocks. So, let us proceed from that point.

(Refer Slide Time: 02:05)



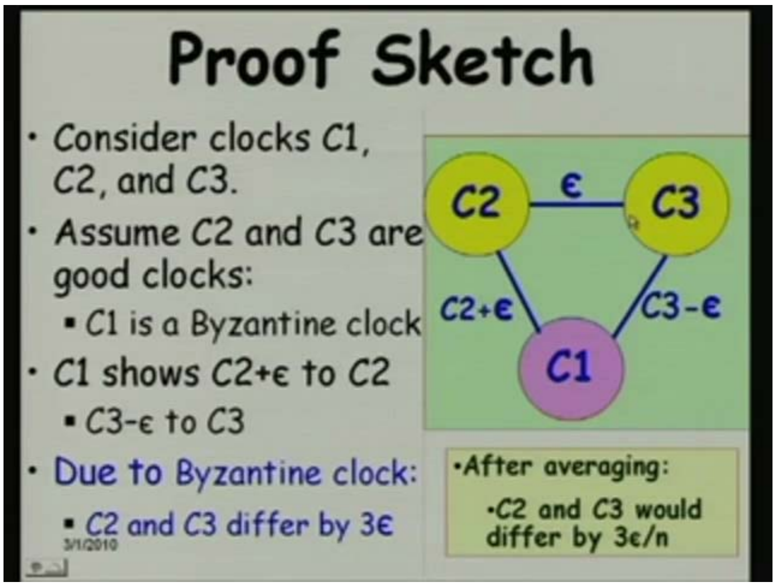
Synchronization in Presence of Byzantine Clocks

- Theorem:
  - In a distributed system with n clocks:
    - A single Byzantine clock can make any two arbitrary clocks to differ by $3\epsilon/n$ sec
    - Where ϵ is the maximum permissible drift between two clocks.

441

So, we we had said that a single Byzantine clock can make any two arbitrary clocks to differ by 3 epsilon by n second, where epsilon is the maximum permissible drift between two clocks.

(Refer Slide Time: 02:23)



Proof Sketch

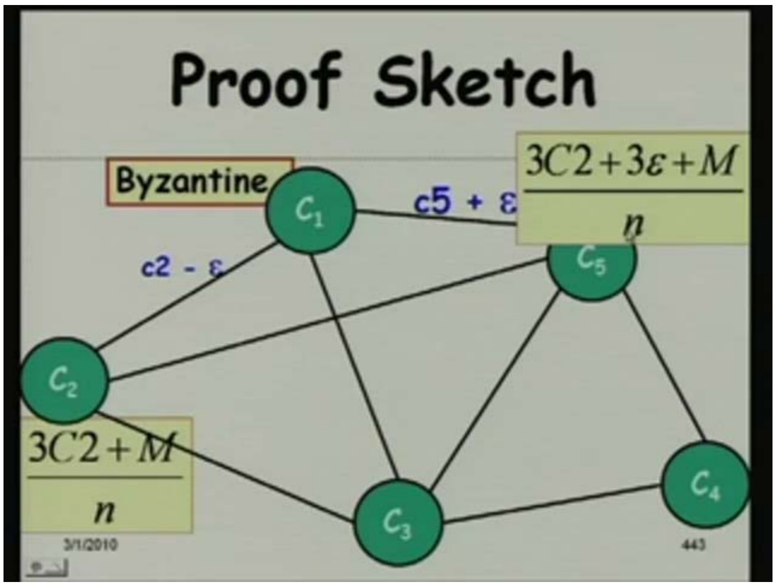
- Consider clocks C1, C2, and C3.
- Assume C2 and C3 are good clocks:
  - C1 is a Byzantine clock
- C1 shows C2+ϵ to C2
  - C3-ϵ to C3
- Due to Byzantine clock:
  - C2 and C3 differ by 3ϵ

- After averaging:
  - C2 and C3 would differ by $3\epsilon/n$

3/1/2010

So that we had seen a sketch of that proof by considering only three clocks. And then, we are trying to show that due to a Byzantine clock - two clocks will differ by at most 3 epsilon time. And when the compute average, the average time will be will differ by 3 epsilon by n. And we said that see, this is just a situation to explain the sketch of the proof rather than the proof itself, because with one Byzantine clock out of three we cannot even keep them synchronized within any bound.

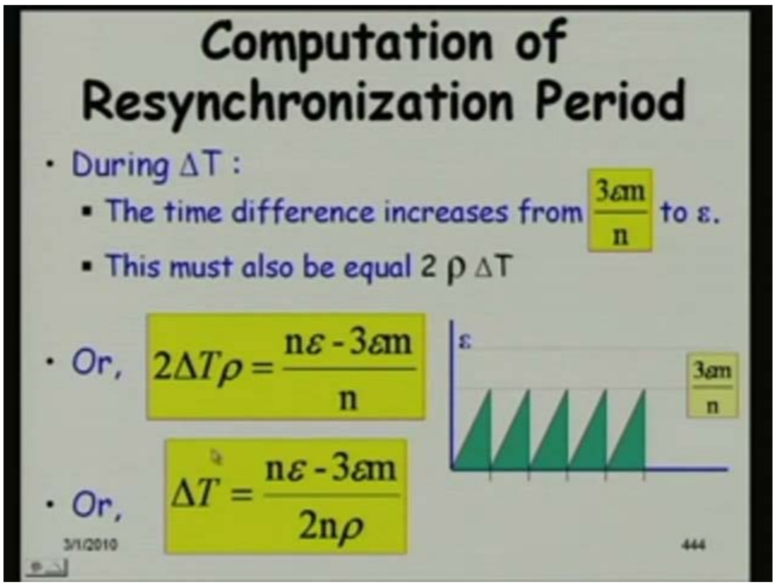
(Refer Slide Time: 03:11)



Proof Sketch

And then we said that see, if we understand three clocks, we can just add the other clocks, and because these are showing the same value to all other clocks. The sum of all their time values that is received by either C 2 or C 5 is a constant. It can be we can consider that is n, and then we are trying to show that the value computed by C 2. The sum of the clock values computed by C 2 will be 3C2 plus M by n, and that computed by C 5 will be 3C2 plus 3 epsilon plus M by n, and then they differ by 3 epsilon m. So that is the proof.

(Refer Slide Time: 03:59)



Computation of Resynchronization Period

- During ΔT :
  - The time difference increases from $\frac{3\varepsilon m}{n}$ to ε.
  - This must also be equal $2 \rho \Delta T$
- Or, $2\Delta T\rho = \frac{n\varepsilon - 3\varepsilon m}{n}$
- Or, $\Delta T = \frac{n\varepsilon - 3\varepsilon m}{2n\rho}$

3/1/2010

444

But if a single Byzantine clock can make two clocks differ by 3 epsilon m. Then, during this resynchronization interval, the 3 epsilon by m 3 epsilon m by n should increase to epsilon, and then the resynchronization should occur, is it not? We want to keep them synchronized within epsilon. And we know that they can differ by 3 epsilon m, we can do anything about it. So, from 3 epsilon m by n to epsilon, we can allow they to diverge, and then we need to resynchronize. See earlier we are trying to resynchronize from 0 to epsilon; is it not?

m is the number of Byzantine clocks.

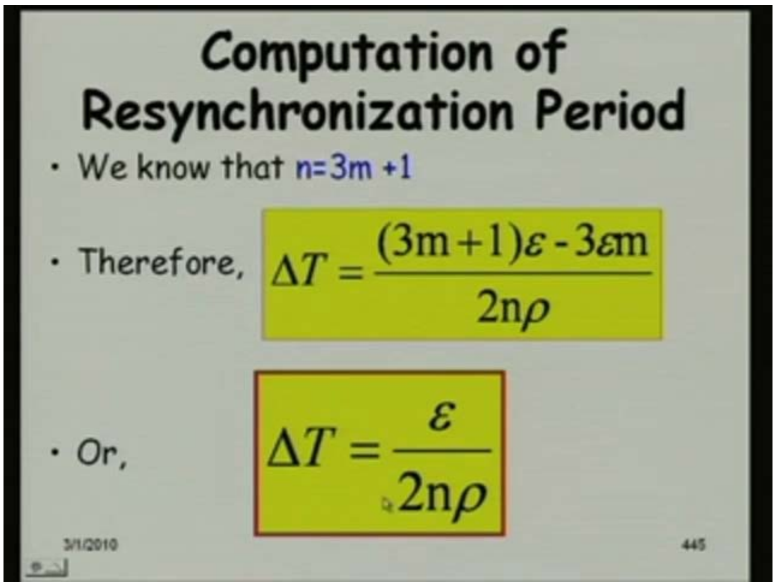
m is the number of Byzantine clocks, yes. So, 3 epsilon by n is the time value that a single Byzantine clock will make 2 clocks differ by n, and m is the number of Byzantine clocks. So, the total time difference due to m Byzantine clocks will be 3 epsilon m by n.

So, we can set 2 delta T rho is equal to n epsilon minus 3 epsilon m by n. So, that is basically epsilon minus 3 3 epsilon m by n or delta T is equal to n epsilon minus 3 epsilon m by 2 n rho. So, this is the idea here. See, the the time values by themselves differ from the differ by 3 epsilon m by n right, you can do much about it. So, from 3 epsilon… I should not have drawn it here actually; I should have drawn it here. So, I have done a mistake here, see… or maybe I should have drawn 3 epsilon m by n here and epsilon here. So, in anywhere 3 epsilon m by n they will differ due to presence of Byzantine clocks. So, from 3 epsilon m by n… So, this I should have brought here.

So, from 3 epsilon m by n to increase to epsilon, by the time they increase to epsilon we should resynchronize right. Or in other words, epsilon minus 3 epsilon m by n is the divergences that will that can we can allow between clocks within the delta T interval.

So, in delta T interval, the divergence will be 2 rho delta T; that we know. So, 2 rho delta T will be epsilon minus 3 epsilon m by n. So, this diagram is a big wrong, I would drawn it wrongly. It should have it 3 epsilon m by n some… If this is the 3 epsilon m by n, then this should have shifted here. So, because they will we have to synchronize as soon as they are difference the drift is equal to epsilon. So, we have to compute how much time that take to drift from 3 epsilon m by n to epsilon. So that is basically the permissible drift is epsilon minus 3 epsilon m by n which is n epsilon minus 3 epsilon m by n which is should be equal to the drift that is possible due to rho - that is 2 delta rho. One might be drifting in the positive side; another might be drifting in the negative side. So, the total drift in delta T will be 2 delta T rho or delta T is equal to n epsilon minus 3 epsilon m by 2 n rho.

(Refer Slide Time: 08:17)


Computation of Resynchronization Period

- We know that n=3m +1
- Therefore, $\Delta T = \frac{(3m+1)\varepsilon - 3\varepsilon m}{2n\rho}$
- Or, $\Delta T = \frac{\varepsilon}{2n\rho}$

3/1/2010 445

But we know that n is equal to 3 m plus 1, because less than one-third the clocks are Byzantine. And therefore, delta T is equal to 3 m plus 1 epsilon minus 3 epsilon m by 2 n rho. So, we can take out this 3 m epsilon here. So, it will leave us with epsilon by 2 n rho. Is that ok?

So, in the presence of Byzantine clocks - m Byzantine clocks, where m is less than one-third of n, then we need to resynchronize after every epsilon by 2 n rho.

That is delta T is independent of m.

See, he he is asking that delta T is independent of m. So, in does anybody have any answer to that question? See, he says that delta T that we have computed, that is the frequency of resynchronization is independent of m. It has become independent of m. Why why is that?

(Audio not audible. Refer Time: 09:36) So, they are committed the maximum possible.

Exactly, so, we are assuming that any number of clocks between 0 to m can be Byzantine. At best there are if there are m Byzantine clocks, we can keep them synchronized; at best there are m byzantine clocks, we can keep them resynchronized by transmitting within epsilon by 2 n rho. Of course, if you had no Byzantine clocks or let us say just one or two Byzantine clocks, then your resynchronization interval can be longer. Is it not? This is the worst-case.

Sir, this with this we can say even if all the n are Byzantine. We can keep it synchronize.

No.

From this expression.

See, this is the assumption that is given, n is equal to 3 m plus 1. Under that assumption, we have substituted here n is equal to 3 m plus 1. But given that n is equal to 3 m or n is equal to m, let us say n is equal to m if you make it. So, m minus 3 epsilon m by 2 n rho. So, that will become minus 2 epsilon by 2 n rho, which is a impossible period for resynchronization right. So here, this expression says that - as long as in the number of Byzantine clocks is less than one-third. You can keep them synchronized by transmitting by every clock transmitting it is clock value, every epsilon by 2 n rho time units. Looks ok?

Sir,

Yes.

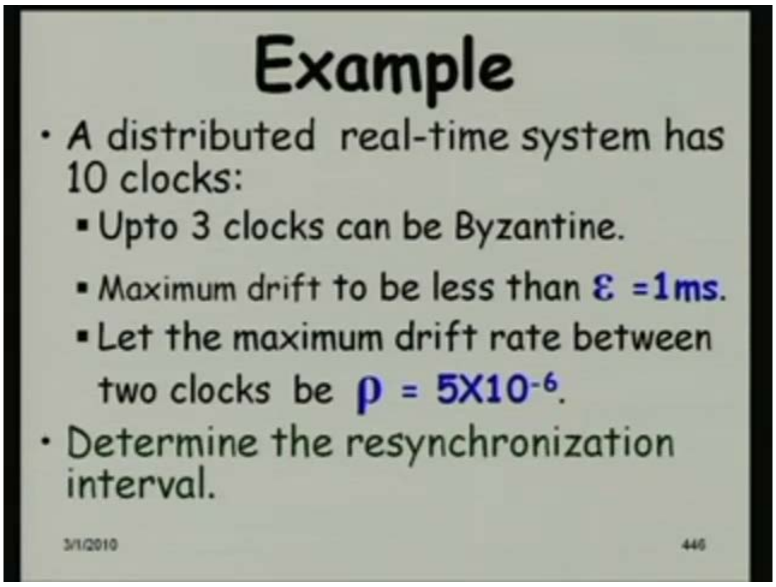
Sir, can we so, in the previous class, we had discussed at least no more than one-third, if it is Byzantine then only we can keep.

Yes yes.

Sir, can we derive this statement from this expression.

Of course, of course, so, may be that is thing that to think about. See, what he says is that we had claimed to start with without any proof. Saying that if less than one-third clocks are Byzantine, then we can keep them synchronize to any bound. If more than one-third, if 3 out of 9 are Byzantine, we cannot even keep them synchronized to any given bound, whether you tell 10 second or whatever, we cannot keep them synchronized. So, what he says that - this can be the basis and which you can claim. So that time we just stated. That see, one-third less than one-third should be byzantine for us to keep them synchronized within any bound. So, this maybe the basis and which we can show that. See, if more than one-third is bad sorry Byzantine, then we cannot keep them synchronized. There retransmission period will be impossible to meet. That is negative right.

(Refer Slide Time: 12:41)



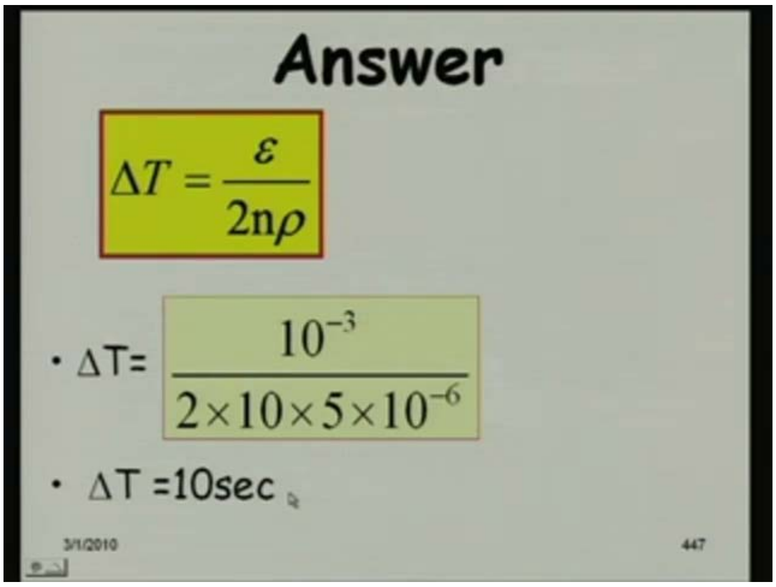
Example

- A distributed real-time system has 10 clocks:
  - Upto 3 clocks can be Byzantine.
  - Maximum drift to be less than $\varepsilon = 1$ms.
  - Let the maximum drift rate between two clocks be $\rho = 5\text{X}10^{-6}$.
- Determine the resynchronization interval.

3/1/2010 446

So, let us look at an example. A distributed real-time system has 10 clocks. And given that up to 3 clocks are Byzantine, and the maximum drift is to be kept less than 1 millisecond. And the clock manufacturer has specified the drift rate between 2 clocks is 5 into 10 to the power minus 6. Determine the resynchronization interval.

(Refer Slide Time: 13:12)



Answer

$$\boxed{\Delta T = \frac{\varepsilon}{2n\rho}}$$

- $\Delta T = \dfrac{10^{-3}}{2 \times 10 \times 5 \times 10^{-6}}$
- $\Delta T = 10\text{sec}$

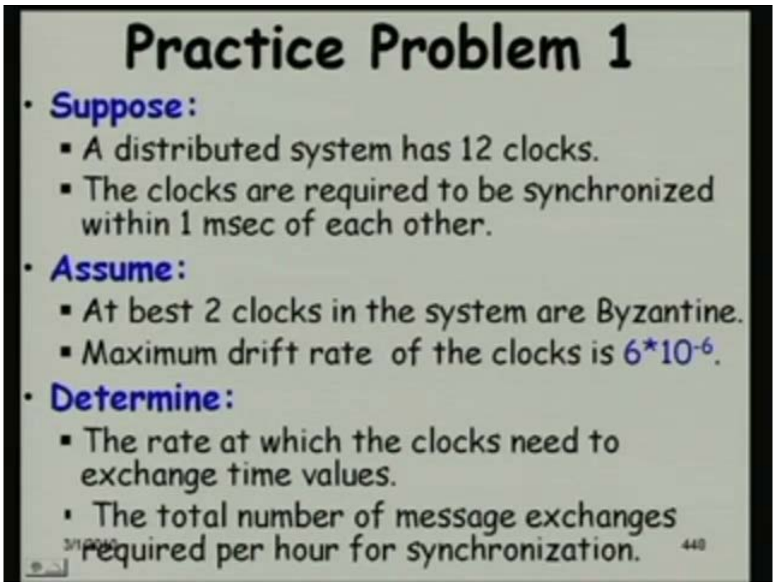
3/1/2010 447

So, this is simple. You know the expression delta T is epsilon 2 n rho, 2 and 10 is the number of clocks, 5 into 10 to the power minus 6 is the drift rate, and 1 millisecond is the maximum drift permissible, and then delta T is 10 second. So, as long as less than 3 clocks are Byzantine, less than or equal to 3 clocks are Byzantine, then as long as the time values are broadcast by every clock in every 10 second, then the clocks will become synchronized. But what if, we know that no clock is Byzantine. How do you do that? What is the synchronization interval?

(Audio not audible. Refer Time: 14:11)

We can earlier we had asked we we were set 2 rho delta T - that is what he says. The 2 rho delta T is the time we took for the clock to diverge from 3 epsilon m by n 2 epsilon, says that now we need to equate that 0 to epsilon right. Simple thing.

(Refer Slide Time: 14:38)


Practice Problem 1

- **Suppose:**
  - A distributed system has 12 clocks.
  - The clocks are required to be synchronized within 1 msec of each other.
- **Assume:**
  - At best 2 clocks in the system are Byzantine.
  - Maximum drift rate of the clocks is $6*10^{-6}$.
- **Determine:**
  - The rate at which the clocks need to exchange time values.
  - The total number of message exchanges required per hour for synchronization.

Now, let us look at another problem. So this, please try to do it. So, we have a distributed real-time system with 12 clocks. The clocks are required to be synchronized within 1 millisecond of each other. At best 2 clocks in the system are Byzantine. We know that at best 2 can become Byzantine. And the maximum drift rate is 6 into10 to the power minus 6. Now, find the rate at which the clocks need to exchange their time values, and also the total number of message exchanges required per hour for synchronization of the clocks. So here…

Sir, at best means.

At best means it can be 0 clock can be Byzantine, 1 clock can be Byzantine or 2 clocks can be Byzantine. And we know that as long as less than 3 clocks are Byzantine, they can be kept synchronized is less than one-third. So, less than 3 clocks, up to 3 clocks they can be kept synchronized. So please try.

(Audio not audible. Refer Time: 16:07 to 16:12) At best 2.

At best 2, so, if you compute assuming that less than one-third that is 3, then your resynchronization interval will be much smaller. It will be smaller, then what it should be actually. So, this is an additional fact that is given - that at best 2 clocks are Byzantine. So, please try to do this. Do not assume that up to 3 clocks will be Byzantine, because this is additional fact here. So, you need to compute. The effect of 1 Byzantine clock that is the most crucial step or crucial result, if we know that we can do that easily that is what is the impact of one Byzantine clock on all clocks in the system? And then, we can easily show that what will happen, if there are 2 clocks are Byzantine. So, one person has completed anyone else has completed. Some two, three persons have completed. So, need their result of one of you, so that you can display that. (No audio from 18:26 to 18:52)

Nothing very difficult, is it not? So, why why you are taking so much time? We know the result for 1 clock - 1 Byzantine clock. How does it impact all other clocks? So, 2 clocks, what will be its impact on the other clocks? And then the drift should be between that value the impact of 2 clocks, because beyond that we cannot keep them synchronized. So, from that point, until they increase to epsilon which is given to be 1 millisecond. That should be equal to the drift that is possible under the specification that the drift rate is 6 into10 to the power minus 6. (No audio from 19:49 to 20:02) So, that should be 1 millisecond minus 2 into epsilon which is 1 millisecond by n, n is 12. So, 1 millisecond minus 2 epsilon by 12.

Sir, we can use that expression 2 rho delta T is equal to epsilon minus 3 epsilon.

No, you do whatever, we need the result; you do whatever. What I am saying is that do not assume 3 as the Byzantine, assume 2 Byzantine clocks is given.

(Conversation not audible. Refer Time: 20:49)

So so, need a need a answer, we will display that. So, your name is what?

Manoj.

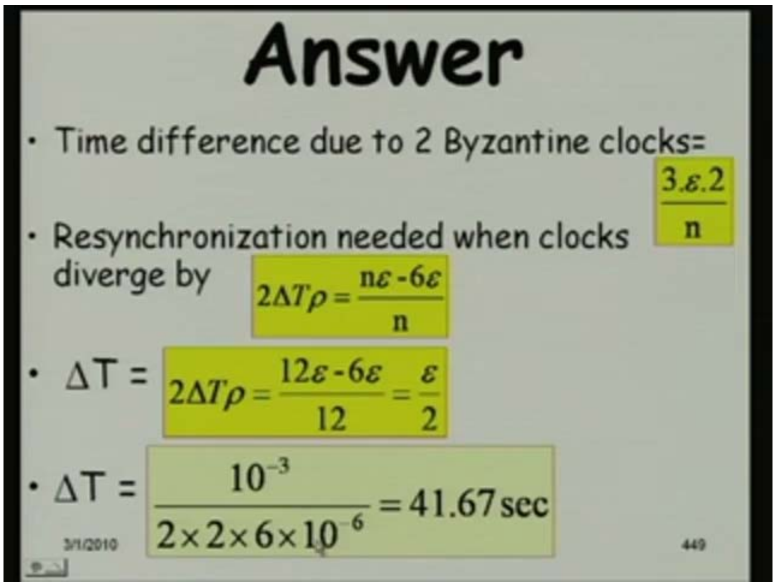
Manoj. Let us, look at the answer of manoj. So, let them focus here. So, are you able to see this? So, looks very front; anyway, so, we have 2 rho delta T is n epsilon minus 3 epsilon m. Why 3 epsilon m?

1 clock 1 Byzantine clock (Audio not clear. Refer Time: 21:44) 3 epsilon m, m is, it is number of Byzantine.

Number of Byzantine; so each epsilon will have 3 epsilon by m, 3 epsilon by n, as the impact on other other clocks. So, the time for a clock to diverge from 3 epsilon to 2 by n 2 epsilon will be epsilon minus 2 into 3 epsilon by m sorry n which is 12, is it not; which is equal to 12 epsilon minus 6 epsilon by 12 which is equal to 6 epsilon by 12 which is epsilon by 2. Is it not? So, epsilon by 2 should be equal to 2 rho delta T, so based on that he has done it and got 1 by 2 into 10 to the power 1 by 24 is it not 1 by 24.

So, 2 rho delta T should be equal to epsilon by 2 or delta T should be equal to epsilon by 2 rho sorry 4 rho, delta T should be equal to epsilon by 4 rho right. So, is that ok with everybody? Let us proceed from that point, let me see, if I had computed that. I think I have computed here.

(Refer Slide Time: 23:25)



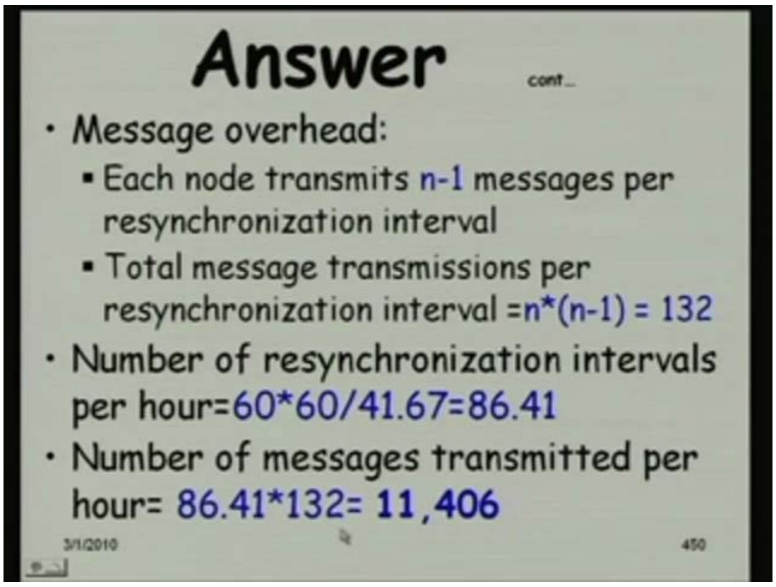
Answer

- Time difference due to 2 Byzantine clocks= $\frac{3.\varepsilon.2}{n}$
- Resynchronization needed when clocks diverge by $2\Delta T\rho = \frac{n\varepsilon - 6\varepsilon}{n}$
- ΔT = $2\Delta T\rho = \frac{12\varepsilon - 6\varepsilon}{12} = \frac{\varepsilon}{2}$
- ΔT = $\frac{10^{-3}}{2\times2\times6\times10^{-6}} = 41.67\,\text{sec}$

3/1/2010 449

So, time difference due to 2 Byzantine clocks is 3 into epsilon into 2 by n, and resynchronization is needed. So, this is equal to 6 epsilon by n, and resynchronization is needed when the clocks diverge by epsilon minus 6 epsilon by n which is n epsilon minus 6 epsilon by n right. Or 2 delta T rho is equal to n epsilon minus 6 epsilon by n or delta T is equal to 10 to the power minus 3 by 4 into 6 to the power minus 6 is equal to 41.67. Just check what is the value you are getting? 41.67 second.

(Refer Slide Time: 24:28)



Answer cont...

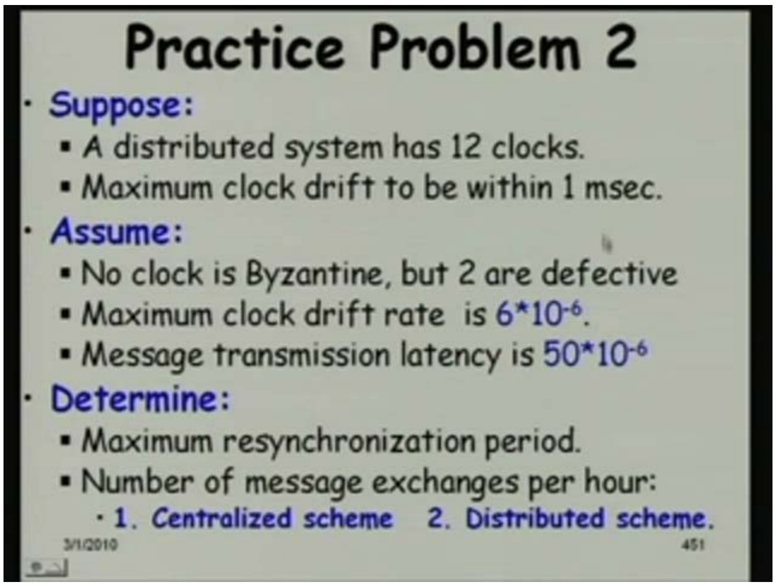
- Message overhead:
  - Each node transmits n-1 messages per resynchronization interval
  - Total message transmissions per resynchronization interval =n*(n-1) = 132
- Number of resynchronization intervals per hour=60*60/41.67=86.41
- Number of messages transmitted per hour= 86.41*132= 11,406

3/1/2010 450

Now, what about the message overhead? . (No audio from 24:31 to 24:41) Per hour, how many clock how many times the clock value needs to be transmitted? . (No audio from 24:48 to 24:58) Each node to transmit n minus 1 messages per resynchronization interval, there are 12 nodes, so, it will send 11 messages. Each node will send 11 messages per resynchronization interval. And we know that the resynchronization interval is 41.6 or something.

So, each node sends n minus 1 and there are 12 nodes. So, 12 into 11 will be 132. So, per resynchronization interval 132 messages need to be transmitted. And we know that there are every 41.67 second, 1 transmission, 1 round of transmission should occur. So, that should be 60 into 60 that many seconds are there in an hour divided by 41.67 is 86.41. And in each of this retransmission each of these 86.41 retransmissions 132 messages are sent. So, the total you will find some 11,406.

(Refer Slide Time: 26:17)



Practice Problem 2

- **Suppose:**
  - A distributed system has 12 clocks.
  - Maximum clock drift to be within 1 msec.
- **Assume:**
  - No clock is Byzantine, but 2 are defective
  - Maximum clock drift rate is $6*10^{-6}$.
  - Message transmission latency is $50*10^{-6}$
- **Determine:**
  - Maximum resynchronization period.
  - Number of message exchanges per hour:
    - 1. Centralized scheme 2. Distributed scheme.

3/1/2010 451

Now, just a small variation of that problem. No clock is Byzantine. See, same problem actually, distributed system has 12 clocks; maximum drift is to be kept within 1 millisecond. No clock is Byzantine, but 2 are defective. Maximum clock rate is 6 into 10 to the power minus 6. Message transmission latency is 50 into 10 to the… Here, there is another factor given here, just checks here. Not only that we have no Byzantine clocks and 2 are defective or bad clocks, we have a message transmission latency also given. Find the maximum resynchronization period - the centralized scheme and the distributed scheme. (No audio from 27:19 to 27:26)

So, first do for the centralized scheme, what is the maximum resynchronization period. . (No audio from 27:33 to 27:38) And then, we need to do for the distributed scheme. (No audio from 27:41 to 27:58) So, are you able to make progress on this, I mean how will you do, first let us find out how will you do - the centralized scheme. Under this data that 2 are defective, no Byzantine, and the message transmission latency is 50 into 10 to the power minus 6, a constant message latency is assumed here between any 2 clocks, 50 into 10 to the power minus 6, need to keep all the clocks synchronized within 1 millisecond.

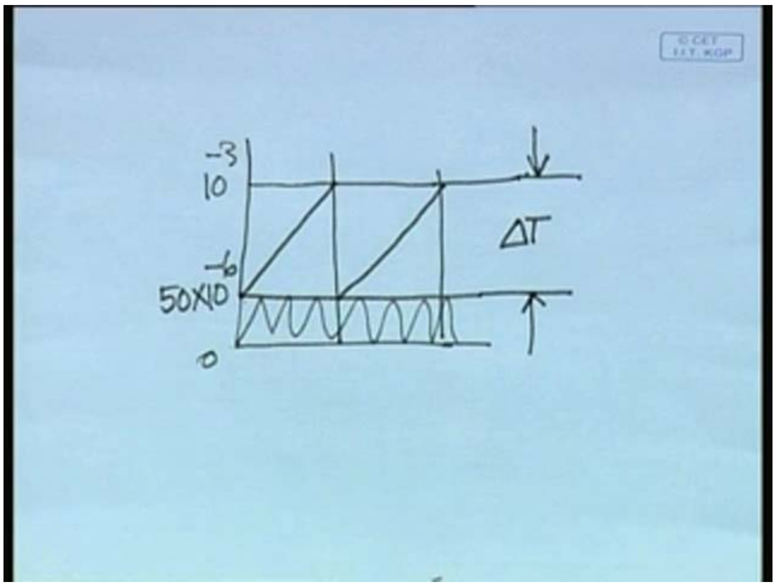
(Audio not audible. Refer Time: 28:44)

Yes yes, see, all the clocks have to be kept resynchronized, even with respect to the master. So, the actual drift that is allowed is 1 minus this value. This becomes more than that is it not? No, no, that is ok. So, this microsecond anyway.

So, it should be a 1 into 10 to the power minus 3, minus 50 into 10 to the power minus 6 right. That is the maximum drift that is permitted. And then the resynchronization should occur. Is it not? So, how much time does it take to drift from 1 minus sorry 10 to the power minus 3 minus 50 minus 10 to the power 6 to 10 to the power minus 3?

So, the drift should be between, so, we cannot keep them resynchronized within 50 into 10 to the power 50 into 10 to the power minus 6. Of course, we can deduct the time value and keep it with synchronized with respect to master that is a different scheme. Right? Since, we know this to be constant. See, if we know that this is constant. You can deduct that from the time value received from the master, and then set that time value, right.

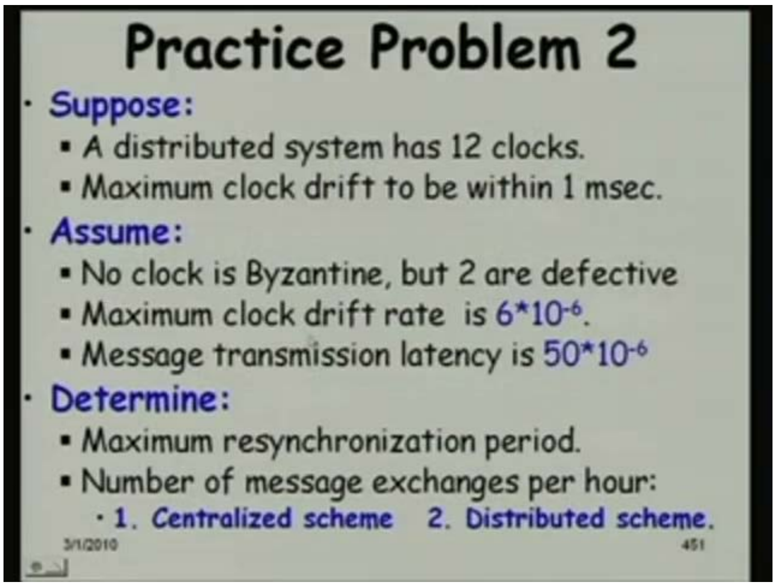
(Refer Slide Time: 29:58)



But let us assume that - this is just a bound. We do not know whether it will be 0 or 1 or 0 into 10 to the power minus 6 or 1 into 10 to the power minus 6, it is not constant let us assume. And then, we cannot keep the clocks synchronized with respect to the master, any finer than 50 into 10 10 to the power minus 6 right, any less than 50 into 10 to the power 6. We cannot keep them synchronized, because the time value can be anything between 0 to 50 into10 to the power minus 6, and this is 10 to the power minus 3. So, by the time, the time diverges to this one, should have a resynchronization right. So, how much over what duration delta T will this divergence from 50 into 10 to the power minus 6 to 10 to the power minus 3 will occur. So, that should be the answer.

So please, that may not be very difficult now to do, but what about the distributed scheme. In a distributed scheme, no clock is Byzantine, 2 are defective. So, 2 defective means 2 time values are simply ignored will be 10. So, the number of clocks becomes 10, because 2 are defective.

(Refer Slide Time: 31:51)



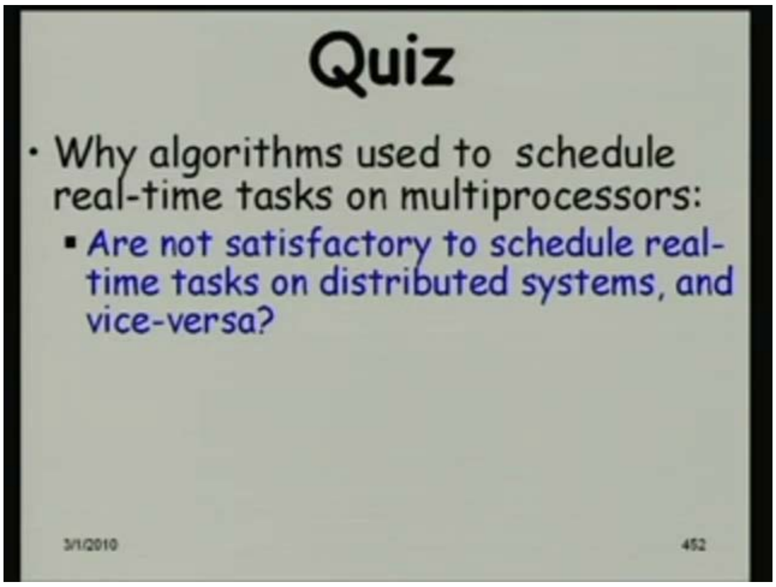
Practice Problem 2

- **Suppose:**
  - A distributed system has 12 clocks.
  - Maximum clock drift to be within 1 msec.
- **Assume:**
  - No clock is Byzantine, but 2 are defective
  - Maximum clock drift rate is $6*10^{-6}$.
  - Message transmission latency is $50*10^{-6}$
- **Determine:**
  - Maximum resynchronization period.
  - Number of message exchanges per hour:
    - 1. Centralized scheme 2. Distributed scheme.

3/1/2010 451

Now, the transmission latency is given. Now, if it is again not a constant latency, it is a bound. So, instead of the Byzantine, see, in the presence of Byzantine clocks we said that see, we cannot keep the clock synchronized, any less than 3 epsilon m by n. But here, we cannot keep them synchronized any less than 50 into 10 to the power minus 6. Right? So, based on that - you the divergence from 50 into 10 to the power minus 6 to 1 millisecond, that is 10 to the power minus 3, again you need to do the computation. So, please try that. (No audio from 33:01 to 33:10)

(Refer Slide Time: 33:11)



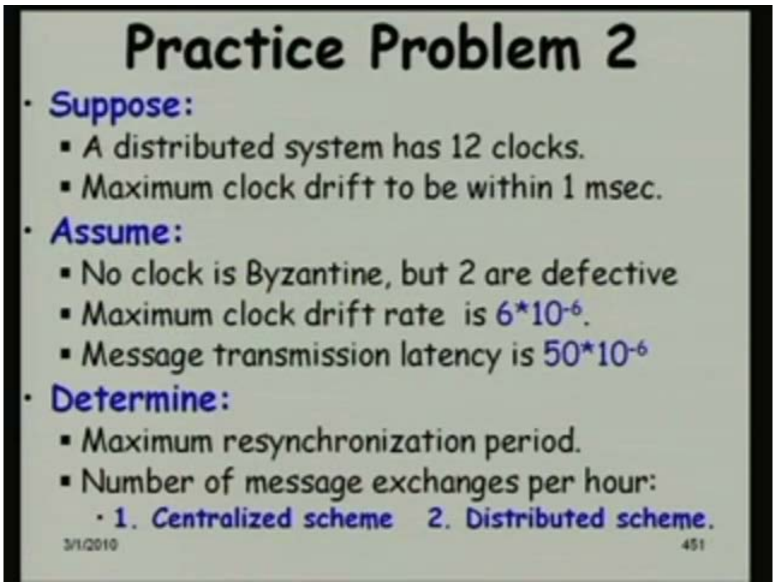
Quiz

- Why algorithms used to schedule real-time tasks on multiprocessors:
  - Are not satisfactory to schedule real-time tasks on distributed systems, and vice-versa?

3/1/2010 452

Let me see, if I have solution there, no, I do not have the solution here. So, please try that. Cannot really give you too much time to compute that, please try that at your home, and if you are difficulty we will discuss in the next class, not anything very difficult actually.

(Refer Slide Time: 33:27)



Practice Problem 2

- Suppose:
  - A distributed system has 12 clocks.
  - Maximum clock drift to be within 1 msec.
- Assume:
  - No clock is Byzantine, but 2 are defective
  - Maximum clock drift rate is $6*10^{-6}$.
  - Message transmission latency is $50*10^{-6}$
- Determine:
  - Maximum resynchronization period.
  - Number of message exchanges per hour:
    - 1. Centralized scheme 2. Distributed scheme.

3/1/2010 451

See here, we have a… See, if you had a constant latency there, if you had a constant latency then (Conversation not audible. Refer Time: 33:42)

So, so, he said that it will be the same. See, if you had the constant message transmission latency, then we would not be impacted at all, because every clock will basically subtract this from the time value received. Because it has taken this much time to come, right 50 into 10 to the power minus 6. So, the time value is older than this 1 or it should have been sent earlier by this time. So, what it will do is sorry it will not subtract it - it will add this time value is it not. The present time value at that clock will be the time value received plus 50 into 10 to the power minus 6, if it was a constant latency. But if it is just a bound that is given, that it can be no more delayed than 50 into 10 to the power minus 6, then the delay can be 0 into 10 to the power minus 6 or 1 10 to the whatever right. So, up to 10 to the power minus 6, and then you need to take that into account.

When the clock takes the message fix time value.

Yes.

Maximum delay.

Maximum delay exactly.

To what is of, we have to find the delta T normally and subtract this value from that.

No no no, see, what we are saying is that - the delta T is the time after which resynchronization occurs right. And by the time, the clocks are drifted up to epsilon. Now, but initially itself, let us assume that the initial difference between clocks is not 0. See after, even if you resynchronize, the difference can be up to this latency. The difference can be up to the latency. See, if it we know that the latency is l, then we can set minus l, but we do not know it can be 0 to l the message transmission latency.

(Question not audible. Refer Time: 34:55) Calculate delta T normally.

Yes.

And send we not consider the message that is period.

Yes.

Now only the extra thing is that, if the latency so, we will take that latency to account and transmit before…

How do you compute delta T?

That delta T minus…

I mean without assuming the latency.

Without assuming the latency.

Yes.

And subtract this value.

Not not really, because there in computing delta T you are assuming that the clocks are all synchronized right. So, starting with total synchrony they were diverging after epsilon, but here we are saying that the clocks can be different from each other by some amount. It is not 0 to delta T. So, we need to resynchronize them more frequently than delta T.

Delta T we can subtract this value, suppose normally.

Yes.

It is synchronize.

Yes.

Then we transmit every delta.

So, you are saying that if it is, let us say 20 second, and let us say the message transmission time is let us say, some microsecond, you will just subtract a microsecond is it. No, just check it out.

That much earlier if we send. So, by that time it will raise the…

May not be that value, because this is the time for the clocks to diverge, it is not that you know from that value subtracts and I do not think so please.

That limit before the diverge into that limit we will transmit by that.

No, see that transmission frequency may be 20 second right. That is a clock transmission. And here, the message transmission delay is of the order of few microseconds. So, you just subtracting few microsecond from 20 second may not be much different; is it not?

So that, transmission period we have computed, considering the case that clocks will not diverge beyond certain.

Ok.

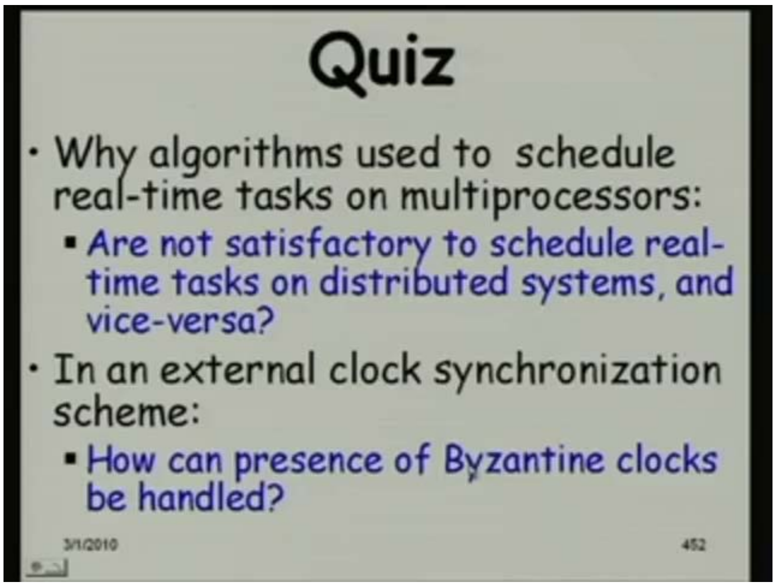
So, that is already taken into the transmission. Now, if we take into consideration the latency also that much we will subtract from the period.

Ok, please think about it. If think about it that whether… See, the argument that I am giving you is that see, the clocks are synchronize to start with, then you can no transmitted delta T, and then you can keep them synchronized. But, here the clocks to start are not synchronized by some amount. So, they have to diverge…

After every interval we are making it synchronize.

No, just think about it I am just telling you, think about it tell me in the next class right. Let us proceed.

(Refer Slide Time: 38:54)



Quiz

- Why algorithms used to schedule real-time tasks on multiprocessors:
  - Are not satisfactory to schedule real-time tasks on distributed systems, and vice-versa?
- In an external clock synchronization scheme:
  - How can presence of Byzantine clocks be handled?

3/1/2010 452

So, let me just ask few questions now that we have computed, we have completed our discussion on clock synchronization. And also the scheduling that we had computed last time and discuss last time, and I did not ask you any questions. So, why is it that algorithms used for scheduling on multiprocessors? I am not satisfactory for distributed systems.

(Audio not audible. Refer Time: 39:26)

Any any other answer.

Multiprocessors system can be at same whereas, distributed…

I mean, how does that impact the scheduling algorithm - tasks scheduling, how will it impact whether there at the same location or at different location.

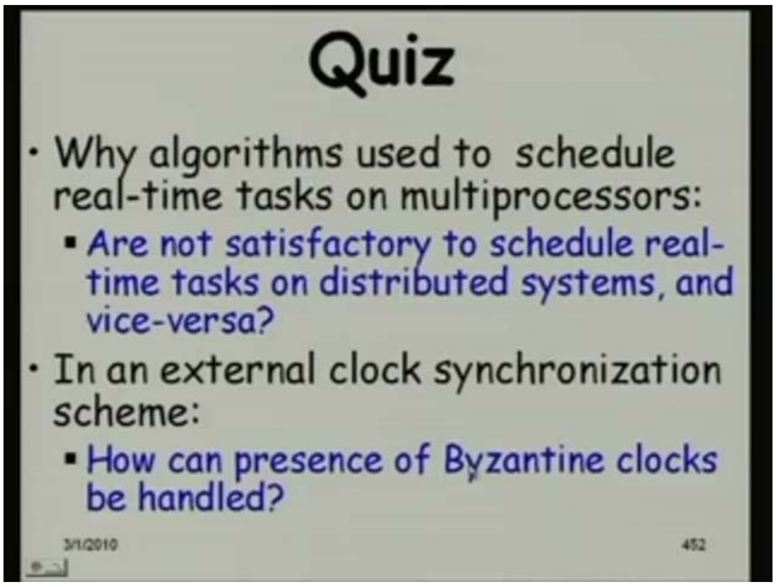
Message transmission overhead.

Ok, message transmission overhead. So we have to consider that in the scheduling, but see if it is static, then there is no overhead, is it not? If it is only dynamic. (( )) If it is dynamic tasks then there is a… But any any other thing that. (( )) I mean did we consider the ordering of events anywhere during hours scheduling algorithms - for distributed and multiprocessor system. Any anyone wants to answer anything. Yes, any other answer.

(Audio not audible. Refer Time: 40:38 to 40:47)

See, in a distributed system we do not have a knowledge of the global status right, we cannot have actually. So, in a multiprocessor system we have that and therefore, we can have a better scheduling algorithm. Easier I mean more efficient algorithm in a multiprocessor system than a distributed system. As far as the static scheduling is concern, it is there is no difference like he said. Because we know the status and each node just run some tasks. Of course, if we consider the communication between the tasks, then there may be a difference. But as long as these are static tasks running at different nodes, there is no difference between a multiprocessor and distributed system; only with respect to dynamic transfer of tasks, whether it can transfer to some node and what is the cost of transfer, time taken to transfer, and the global knowledge of the status of various nodes, so that will make the difference. That is the main reason, why we cannot use distributed tasks scheduling algorithm in a multiprocessor and vice versa.

(Refer Slide Time: 42:05)



Quiz

- Why algorithms used to schedule real-time tasks on multiprocessors:
  - Are not satisfactory to schedule real-time tasks on distributed systems, and vice-versa?
- In an external clock synchronization scheme:
  - How can presence of Byzantine clocks be handled?

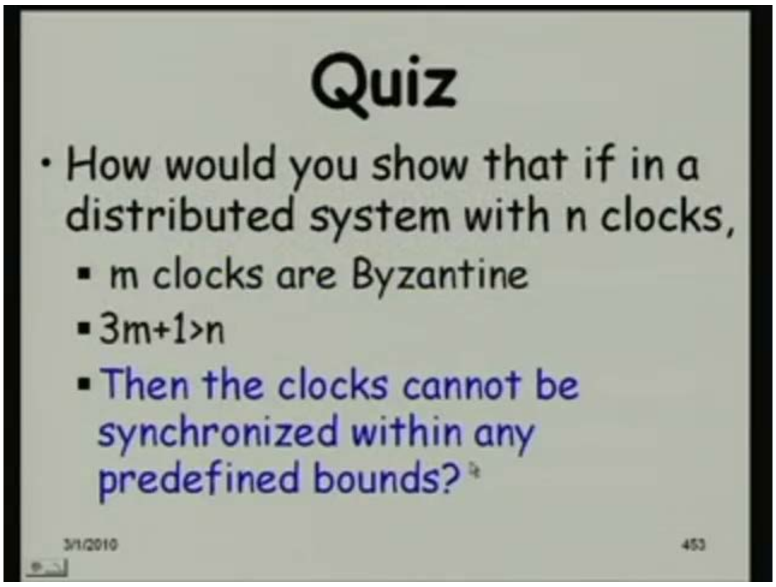
3/1/2010 452

Now, in an external clock synchronization scheme, how can Byzantine clocks be handled?

(Audio not audible. Refer Time: 42:16)

Ok right. So, see he has understood it fully. He says that see, Byzantine clocks are the clocks of the system right. The external clock is not Byzantine. So, the external clock just keeps on transmitting, and whatever be the clock it just sets its own value, Byzantine has no meaning that is what he said correct.

(Refer Slide Time: 42:39)



Quiz

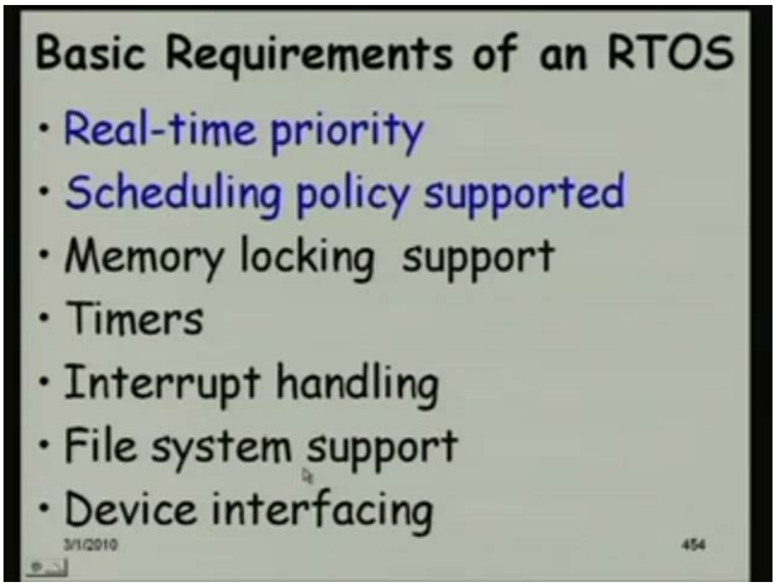
- How would you show that if in a distributed system with n clocks,
  - m clocks are Byzantine
  - 3m+1>n
  - Then the clocks cannot be synchronized within any predefined bounds?

3/1/2010 453

I think this he just argued out there that how would you show that if there are n clocks in a distributed system, and m are known to be Byzantine, where 3 m plus 1 is greater than n. So, that is more than one-third is Byzantine, then the clocks cannot be resynchronized within any predefined bounds. So that I think we need not discuss now, because he has already argued out that. See, if we try to keep them synchronized using the scheme that we discussed internal clock synchronization, then we need to retransmit or resynchronize them with negative frequency.

Now, since, we have discussed about the basics of scheduling in a uniprocessor multiprocessor and distributed systems, and we looked at the clock synchronization. Let us look at the real-time operating systems. First let us look at some very basic requirements of a real-time operating system.

(Refer Slide Time: 43:38)



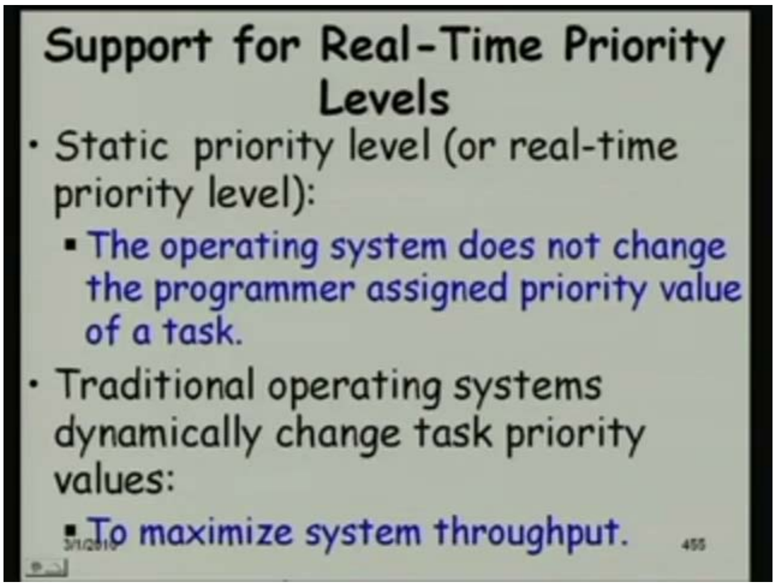
Basic Requirements of an RTOS

- Real-time priority
- Scheduling policy supported
- Memory locking support
- Timers
- Interrupt handling
- File system support
- Device interfacing

3/1/2010 454

The first thing is a real-time priority. We will see what you mean by a real-time priority, then what kind of scheduling policy supported. So, these two are important factors - that a real-time operating system must support. One is that it must support real-time priority levels, and the kind of scheduling policy to be supported - memory locking support. See, why we need a memory locking? Need timers, interrupt handling, I mean different from what we have been discussing in traditional operating system. Special file system support all these real-time file system, special arrangement for device interfacing.

(Refer Slide Time: 44:47)



Support for Real-Time Priority Levels

- Static priority level (or real-time priority level):
  - The operating system does not change the programmer assigned priority value of a task.
- Traditional operating systems dynamically change task priority values:
  - To maximize system throughput.

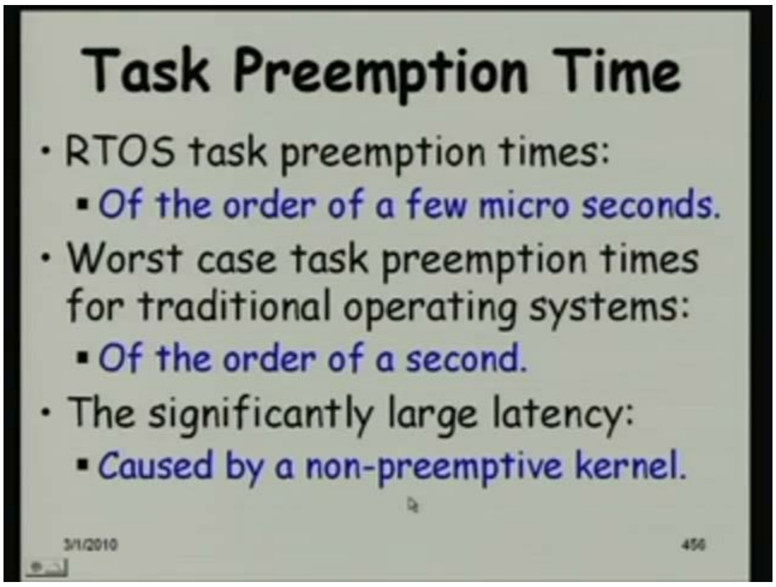
3/1/2010 455

But first let us look at what do you mean by real-time priority level? Actually, a real-time priority level which will also called a static priority level. Here, the operating system does not change the programmer assigned priority value of a task. Once a programmer sets a time value sorry priority value for a task, then it just continues that to remain that throughout the execution of the task.

But the traditional operating systems - UNIX, Windows or whatever they all once the programmer assigns a time priority value they keep on changing their priority value of the task. Why do they do that? It is a very fundamental question in any operating system discussion, that why an operating system needs to change the priority value of a task. We will see, we will discuss little further here, because normally not discussed in a first level operating system course. That why operating systems needs to change the priority of a task as it executes. We will see that the answer is to maximize the throughput, we will also give a formula they use to compute the priority dynamically that is both Windows and UNIX and all other operating systems.

Another requirement for a real time operating system will be the task preemption time. For a real-time operating system, we need a preemption time which is less than a few microseconds.

(Refer Slide Time: 46:17)



Task Preemption Time

- RTOS task preemption times:
  - Of the order of a few micro seconds.
- Worst case task preemption times for traditional operating systems:
  - Of the order of a second.
- The significantly large latency:
  - Caused by a non-preemptive kernel.

3/1/2010 456

But we will see that the worst case preemption time of the traditional operating systems like UNIX around those see some order of a second. We will see that this is caused by this large difference in the preemption time is caused by a non-preemptive kernel. We will see that - what is this non-preemptive kernel of a traditional operating system and why it is non-preemptive to start with, but what do you mean by task preemption time? Yes anybody.

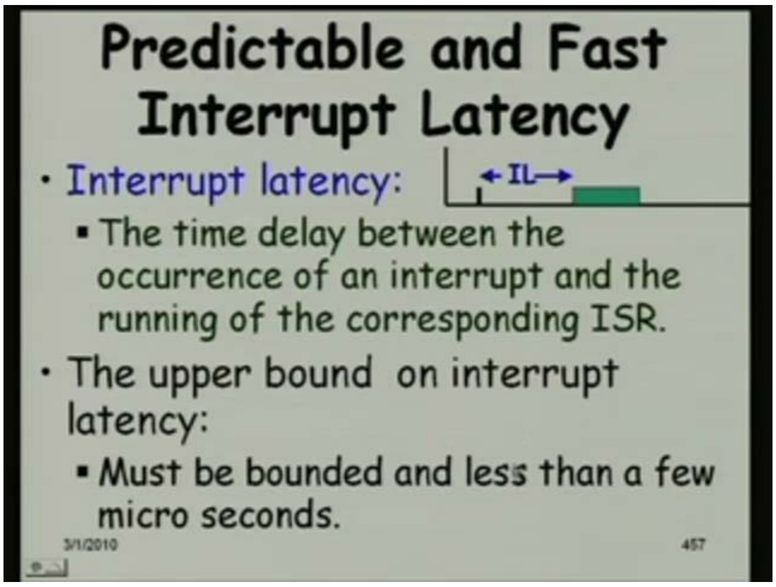
(Audio not audible. Refer Time: 47:20)

Ok, so, we were discussing a priority driven preemption preemptive scheduling mechanism. So, under some events a current task must be preempted. So, after determining that the current task will be preempted, how much time does it take too actually to the preemption? We will see that traditional operating system; we need a large preemption time. Because, the kernel is non-preemptable, once it is in the kernel mode, we have to wait until it comes out of that. And even the user program it can go into a kernel mode right. Do you agree with that - how will it go into kernel mode?

System calls.

System calls, exactly.

(Refer Slide Time: 48:20)



Predictable and Fast Interrupt Latency

- Interrupt latency:
  - The time delay between the occurrence of an interrupt and the running of the corresponding ISR.
- The upper bound on interrupt latency:
  - Must be bounded and less than a few micro seconds.

3/1/2010 457

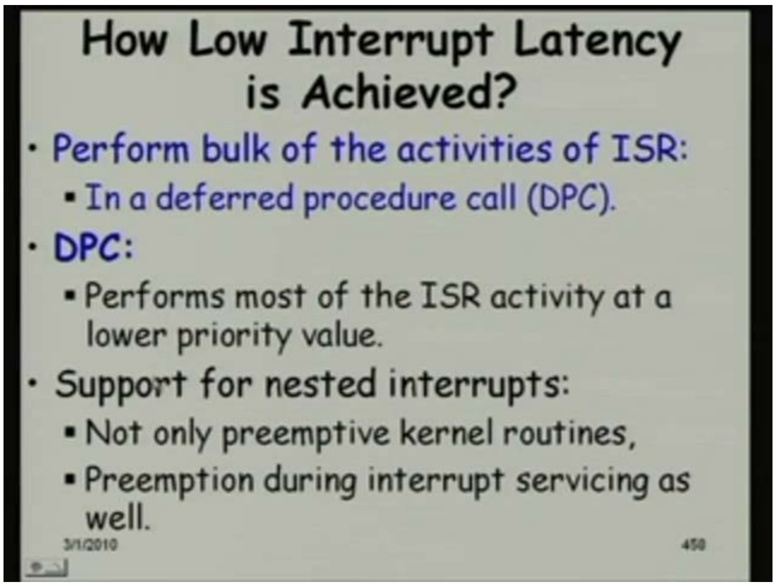
So, the interrupt latency is the time delay between the occurrence of an interrupt, and the running of the corresponding interrupt service routine. So, in real-time systems we will see that various types of devices, sensors and actuators, they give interrupts. And once the interrupt comes, how much time does it take to run the corresponding service routine? So, this is the event, the interrupt event, and then this is the interrupt service routine starts executing from here, the time that lapses between the occurrence of the event to the running of the interrupt service routine is the interrupt latency.

Now, later on we will see that there are various factors, which result in a large interrupt latency in a traditional operating system including the time to recognize the interrupt, and also if the we will see that the traditional operating systems they mask their interrupts. And also, if one ISR is running, so, the the ISR is non-preemptable, you cannot have a recursive ISR right, one ISR preempts another etcetera is not possible. So, under those situations that is that time to recognize the interrupt, and also, for the interrupt routine to run the time.

Higher priority and the interrupt... (Audio not audible. Refer Time: 50:18)

We will discuss more details of that - that the non-reentrant ISR's, we will we will discuss about that. So, this fixes the bound and the latency - interrupt latency, and we need this to be at least less than a few microseconds and for sensitive applications might need the order of few nanoseconds.

(Refer Slide Time: 50:42)



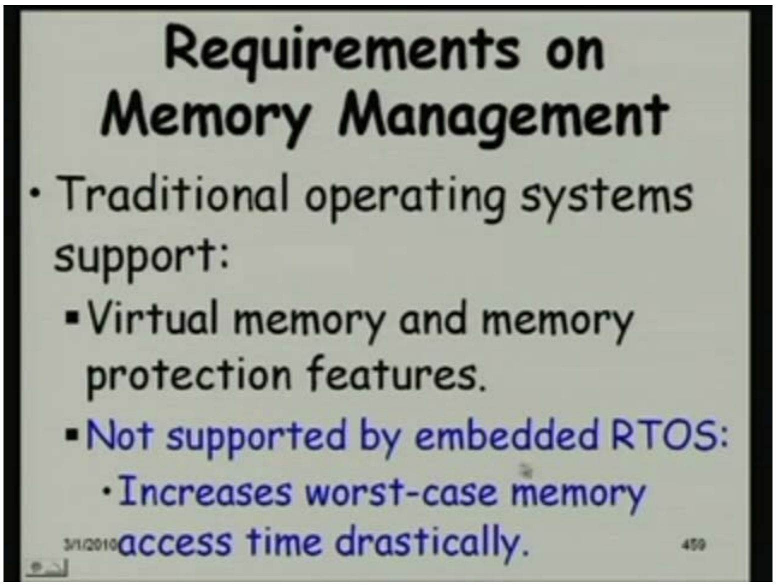
How Low Interrupt Latency is Achieved?

- Perform bulk of the activities of ISR:
  - In a deferred procedure call (DPC).
- DPC:
  - Performs most of the ISR activity at a lower priority value.
- Support for nested interrupts:
  - Not only preemptive kernel routines,
  - Preemption during interrupt servicing as well.

3/1/2010 450

So, unless we have nested interrupt support. We need to perform bulk of the activities in a deferred procedure call. The interrupt service, the idea is that if the interrupt service routine takes long and nested interrupts are not possible. Then we need to minimize the duration for which an ISR runs, and we will see that all operating systems, including the present Windows, Linux etcetera. They use a deferred procedure call. The idea is that; only do the very basic processing required as a ISR and the rest of the task is queued as a deferred procedure call. A deferred procedure call is substantial part of the ISR activity, which is executed just like a normal task, only the very crucial part of the ISR is carried out, and then the DPC’s are queued a tasks. And not only we need preemptive kernel routines, but preemption during interrupt servicing as well.

(Refer Slide Time: 52:03)



Requirements on Memory Management

- Traditional operating systems support:
  - Virtual memory and memory protection features.
  - Not supported by embedded RTOS:
    - Increases worst-case memory access time drastically.

3/1/2010 459

And, we will also need some special support with respect to memory management. The traditional operating systems we know that they support virtual memory and the memory protection features. Many of the embedded real time operating systems, they do not really support virtual memory. The reason is that - in presence of virtual memory, the worst case memory access time increases drastically; why should it increase drastically? When your virtual memory support, why should the memory worst case memory access time increase drastically compare to the average at the best case memory access time?

Some pages are in the…

Yes, page faults. So, the memory access time will increase, because they might be hard disk like he says page fault. So, that may be unacceptable for some embedded real-time operating - embedded real-time tasks. Therefore, many of these as we will see commercial real-time operating systems, they do not support virtual memory. So, we will just stop here, and we will continue from this point in the next class.

Thank you.