

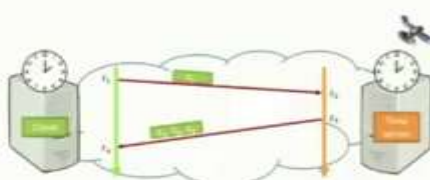
Foundation of Cloud Iot Edge ML
Professor Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna
Lecture 04
Time and Clock Synchronization in IoT

(Refer Slide Time: 0:26)

Preface

Content of this Lecture:

- In this lecture, we will discuss the fundamentals of clock synchronization in IoT and its different algorithms.
- To understand how clocks operate on IoT devices and how they can be synchronized in an accurate and efficient fashion.



Time and Clock Synchronization

Preface

Content of this Lecture:

- Internet of Things (IoT) devices that are wirelessly connected in mesh networks often need mutual clock time synchronization, to enable chronological ordering of sensor events, coordination of asynchronous processes across devices, or network-wide coordination of actuators. ✓
- We will also discuss the causality and a general framework of logical clocks and present two systems of logical time, namely, lamport and vector, timestamps to capture causality between distributed events of an Internet of things as a distributed system.

Time and Clock Synchronization

I am Dr. Rajeev Mishra IIT, Patna. So, today's lecture is on time and clock synchronization in the internet of things. So, the content of this lecture is that we will discuss the fundamentals of clock synchronization and the need in IoT and the different clock synchronization algorithms. To understand how the clocks operate in the IoT devices and how they are synchronized in an accurate and efficient manner is the main objective of this particular lecture.

So, the content of this lecture is the motivation in the following way. So, the internet of things devices are wirelessly connected forming a mesh networks often need for the mutual clock time synchronization to enable chronological ordering of sensor events, coordination of asynchronous processes across the devices or network wide coordination of actuators. So, this is very much essential, why because you know that the sensors are the devices which are deployed whether it is a smart city scenario or it is an industrial manufacturing unit.

So, these sensor networks or these sensors together will form a network and each particular sensor device is having the timestamp with the event when this particular data is now send. So, therefore there is a need of clock synchronization. In order to find out the chronological ordering of the sensor events that data which sensor send should have the timestamp to have the correct timestamp all the clocks need to be synchronized to find out the ordering of sensor events coordination of asynchronous processes across the devices and network wide coordination.

So, again I am summarizing the content or the motivation of this lecture is that internet of things devices which are often deployed and form a connected wireless network or you can also say it is a network of IoT devices. Now, these IoT devices require to send the data with a timestamp. Therefore, the clocks are running in these IoT nodes or the devices and often to have this correct ordering of the events of the sensor data they require to have the mutual clock synchronization at all points of time.

Now, this coordination of asynchronous processes also sometimes requires the clock to be synchronized in these IoT devices. And also the network wide coordination of actuators also sometimes requires the clock to be synchronized at the actuators. So, this particular need in IoT system which is a system of clocks running around IoT devices needs a clock synchronization in a distributed system of internet of things network.

So, here in this lecture we will discuss also other than the physical clock synchronization we will also consider that IoT devices may form a distributed system and therefore the logical clock synchronization principles also very much essential in the IoT network. So, we will consider the logical clock synchronization. And in that context we will talk about the lamport clocks, vector clocks and these particular clocks will give a timestamp to capture the causality between the distributed events which is captured by the internet of thing as the distributed systems.

(Refer Slide Time: 4:22)

Need of Synchronization

Synchronizing clocks on Internet of Things (IoT) devices is important for applications such as monitoring and real time control.

✓ You want to catch a bus at 9.05 am, but your watch is off by 15 minutes

What if your watch is Late by 15 minutes?

- You'll miss the bus! ✓

What if your watch is Fast by 15 minutes?

- You'll end up unfairly waiting for a longer time than you intended ✓

Time and Clock Synchronization

So, let us see the need for clock synchronization. So, synchronizing the clocks in the internet of things devices is important for the application such as monitoring and real time control. So, to understand the concept of the synchronization, you can see an example to make it understand the need of clock synchronization in the internet of things scenario as well from our life like examples.

So, you may want to catch a bus at let us say 9:05 am but your watch is off by 15 minutes. So what if your watch is late by 15 minutes, then you will miss the bus what if your watch is fast by 15 minutes then you will end up unfairly waiting for a longer time than you intended. Therefore, if your clock is not synchronized, one of these two events may happen either you are too late and you will miss the event that is to catch the bus or if let us say your clock is fast then it will be unfair that, that event is waiting for a longer time to get executed.

(Refer Slide Time: 5:36)

Time and Synchronization

- **Time and Synchronization**
("There's is never enough time...")
- **Distributed Time**
 - The notion of time is well defined (and measurable) at each single location
 - But the relationship between time at different locations is unclear
- **Time Synchronization is required for:**
 - Correctness ✓
 - Fairness

Time and Clock Synchronization

So, therefore time and synchronization are very much essential. As far as the distributed time is concerned the notion of time is well ordered at each single location, but the relationship between the time at different location is not clear. So, therefore, there is a need of time synchronization which we have seen in the previous slide the example that it requires for the correctness and it requires for the fairness.

(Refer Slide Time: 6:13)

Synchronization in an IoT

Example: Cloud based airline reservation system:

- Server X receives, a client request, to purchase the last ticket on a flight, say PQR 123.
- Server X timestamps the purchase using its local clock as **6h:25m:42.55s**. It then logs it. Replies ok to the client. ✓
- That was the very last seat, Server X sends a message to Server Y saying the "flight is full".
- Y enters, "Flight PQR 123 is full" + its own local clock value, (which happens to read **6h:20m:20.21s**).
- Server Z, queries X's and Y's logs. Is confused that a client purchased a ticket at X after the flight became full at Y. ✓
- **This may lead to full incorrect actions at Z**

Time and Clock Synchronization

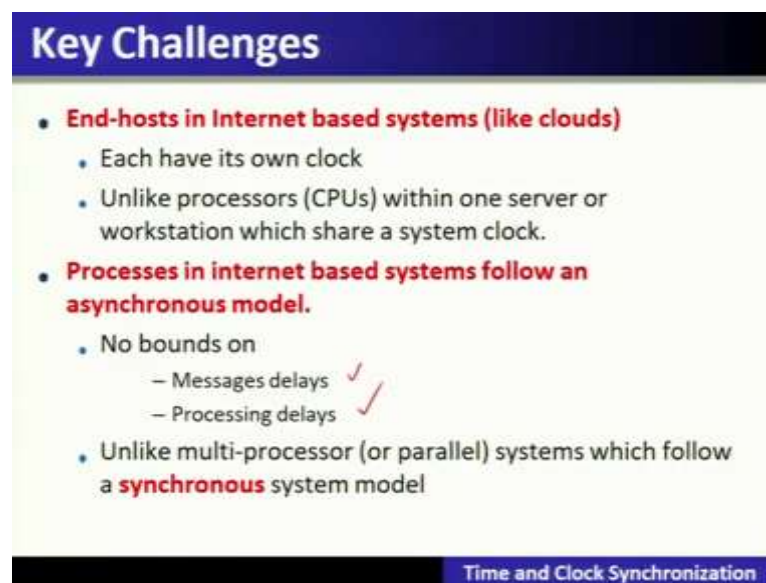
So, to understand the synchronization in an IoT system we have to now understand the basic principle of synchronization in a similar environment which is let us say a cloud based reservation system. So, when you say cloud you mean that a lot of servers are now running

and now there is a need for the synchronization in let us say airline reservation system. So, server X receives a client request to purchase the last ticket of a flight.

Now, server X timestamp the purchase during the using its local clock let us say 6 hours 25 minutes and locks it and replies okay to the customer that was the last seen the server X sends a message to the server Y saying that the flight is full. So, Y enters the flight as full and puts its local clock time as 6 hours 20 minutes because his clock is late. So, therefore 6:20 it will detect that server Y will detect that the flight is full whereas server X has it is clock is 6:25 when he has booked the ticket.

So, therefore server Z when queries X server X and server Y's log it is confused that the client has purchased the ticket at X 6:25 and whereas Y has seen that the flight is full at 6:20 time of the server Y so therefore this is a problem of clock synchronization the server Y's clock is slower than the server X's clock. So, it will not be able to reason why the seat is incorrectly full showing at Z.

(Refer Slide Time: 8:25)



Key Challenges

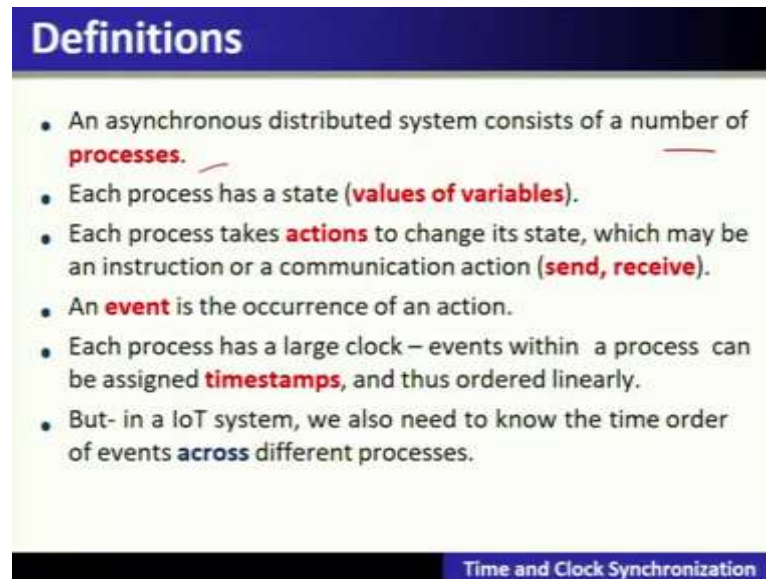
- **End-hosts in Internet based systems (like clouds)**
 - Each have its own clock
 - Unlike processors (CPUs) within one server or workstation which share a system clock.
- **Processes in internet based systems follow an asynchronous model.**
 - No bounds on
 - Messages delays ✓
 - Processing delays ✓
 - Unlike multi-processor (or parallel) systems which follow a **synchronous** system model

Time and Clock Synchronization

So, what are the key challenges? Key challenges is that the end host in the internet based systems like cloud and similar system is there internet of things each have it is own clock and unlike CPUs, unlike processor within one server or the workstation, which share a system clock, this becomes a quite challenging in the distributed environment, because all the clocks are running autonomously and they can only communicate through the messages.

So, the processes in the internet based systems follow an asynchronous model where there is no bounds on the message delays and also there is no bounds on the processing delays and unlike multiprocessor or parallel system which follows a synchronous model.

(Refer Slide Time: 9:17)



Definitions

- An asynchronous distributed system consists of a number of **processes**.
- Each process has a state (**values of variables**).
- Each process takes **actions** to change its state, which may be an instruction or a communication action (**send, receive**).
- An **event** is the occurrence of an action.
- Each process has a large clock – events within a process can be assigned **timestamps**, and thus ordered linearly.
- But- in a IoT system, we also need to know the time order of events **across** different processes.

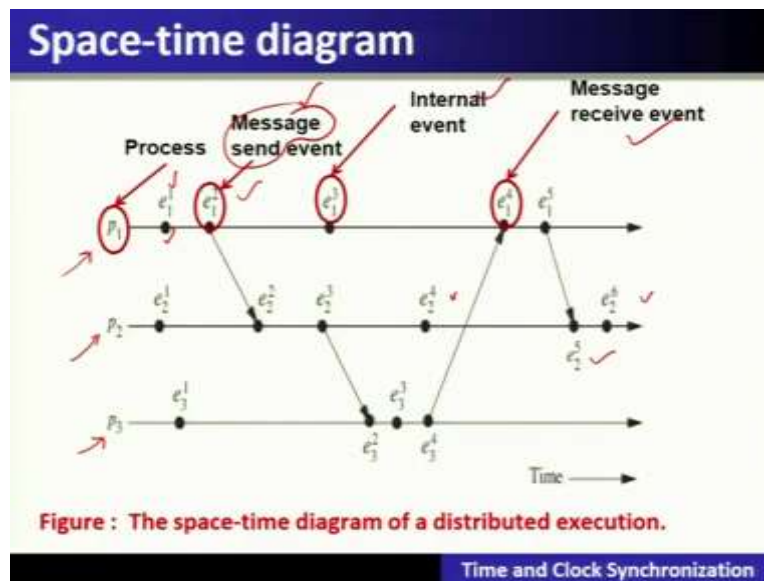
Time and Clock Synchronization

So, let us understand some definitions and then we will see the basic algorithms for the clock synchronization in the distributed system where we are considering the IoT also as an asynchronous distributed system, where each IoT device which is a sensor or an actuator running its own clock nearby in particular device in an embedded system. So, an asynchronous distributed system let us say it consists of number of processes.

So, whether it is the system or a process, we can interchangeably be using the same notion, which may mean the same thing. So, for the sake of simplicity, we will be using the term processes. So, each process has a state that is number of that is the values of the variable, each process takes the action to change it is state which may be an instruction or a command action that is the send or a receive action.

So, event is an occurrence of an action. So, each process has a large clock which is an event within the process and can be assigned a timestamp and then order linearly in IoT system we need to know that the time order of events across the process is also in the same manner.

(Refer Slide Time: 10:48)



So, let us understand using time and space diagram the same concept that this time and space diagram of a distributed execution. So, here let us say that there are three different processes you can think of three different IoT devices running these processes, where there is a event happening within that particular process or an IoT system that is e_1 , e_2 , e_3 and so on of a process 1.

Similarly, for a process 2 or a second internet device is also occurring, various internal events happening at that particular IoT device that is event 1 at process 2, event 2 at process 2, event 3 at process 2, event 4 at process 2, event 6 at process 2, event 5 at process 2.

Similarly, for process 3 is also shown in this particular manner. So, as we have mentioned in the previous slide that for a process 1 this event 1 is an internal event whereas, for process 1, event 2 is the send of a message is an action event, which will send a message to process 2. So, this event e_3 for a process 1 is called internal event and where the two other events which is shown what here in process 1 that is event 2 as a message send event and a message receive event.

So, either there are that means the events are the internal events happening within the process or within the IoT system or it is sending a message or it is receiving a message. So, there are three different types of events only we are considering to understand this particular time synchronization or a distributed execution for the sake of simplicity.

(Refer Slide Time: 13:06)

Clock Skew vs. Clock Drift

- Each process (running at some end host) has its own clock. ✓
- When comparing two clocks at two processes. ✓
 - **Clock Skew = Relative difference in clock values of two processes.** ✓
 - Like distance between two vehicles on road.
 - **Clock Drift = Relative difference in clock frequencies (rates) of two processes** ✓
 - Like difference in speeds of two vehicles on the road.
- A non-zero clock skew implies clocks are **not synchronized**. ✓
- A non-zero clock drift causes skew increases (eventually). ✓
 - If faster vehicle is ahead, it will drift away.
 - If faster vehicle is behind, it will catch up and then drift away.

Time and Clock Synchronization

Now, let us see the concept of a clock skew and clock drift. So, each process which is running at some end host has its own clock, let us assume that without loss of generality now when comparing 2 clocks at 2 processes, then the definition of a clock skew says that the relative difference in the clock values of the 2 processes is called a clock skew. For example, if one clock is saying 3 o'clock the other clock is saying 3:30. So, that difference of these 2 clock values that is 30 minutes is called a clock skew.

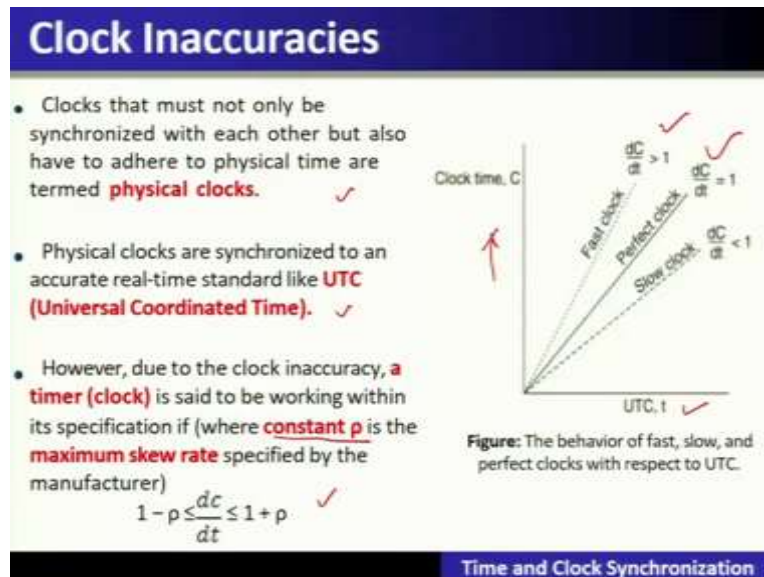
Similarly, you can also understand that the distance between two vehicles on the road is called a skew. Now, another concept is called a clock drift, the relative difference in the clock frequencies or the rates of the two processes is called the clock drift. Now, you know that there is a quartz crystal running the clock at different rates or the frequencies. So, that difference that relative difference of the clock frequencies is called a clock drift.

So, that example is you can also see from the real life example that the difference in the speed of 2 vehicles on the road is called a clock drift. So, the distance between two vehicles is called a clock skew and the difference in the speed of the 2 vehicles is called a clock drift. So, clock skew and clock drift there are two fundamental concepts which we will be using here in the clock synchronization in the distributed system.

So, non-zero clock skew means that the clocks are not synchronized. For example, if the clock is giving a time 3 o'clock the same instance the other clock is giving 3:30 that is there is a clock skew of a non-zero value therefore these clocks are not synchronized. Non-zero clock drift causes that skew increase eventually.

So, for example if a faster vehicle is ahead it will drift away or if a faster vehicle is behind it will catch up and then drift away. So, therefore, non-zero clock drift causes the skew increases eventually. So, these two non-zero clock skew, non-zero clock drift requires adjustment and that is called a clock synchronization.

(Refer Slide Time: 15:44)



So, let us see what you mean by the clock inaccuracies. So, clock must not only be the synchronized with each other but also have to adhere to the physical time which is termed as the physical clocks. So, physical clocks are synchronized to an accurate real time standard called universal coordinated time. So, universal coordinated time is considered to be the correct time and all the other clocks required to be synchronized with that time.

So, how are due to the clock inaccuracies a timer or a clock is set to be working within the specification if there is a constant row is the maximum skew rate specified by the manufacturer. So, with this particular notion and given the universal coordinated time and also given the specification that is a constant row and maximum skew rate which is specified by the manufacturer.

So, you can also understand the behavior of a fast, slow and the perfect clock in this particular example shown in this particular figure. So, on the X axis you have a universal coordinated time UTC shown on the X axis and the clock time you can see that is capital C on the Y axis and for the perfect clock this particular rate that is dc by dt should be equal to 1 for a slow clock this dc by dt should be if it is less than 1 and for a first clock dc by dt is greater than 1. So, these are all aspects are the clock inaccuracies.

(Refer Slide Time: 17:41)

How often to Synchronize

- **Maximum Drift rate (MDR) of a clock** ✓
- Absolute MDR is defined to relative coordinated universal Time (UTC). UTC is the correct time at any point of time.
 - MDR of any process depends on the environment.
- Maximum drift rate between two clocks with similar MDR is **$2 * \text{MDR}$** .
- Given a maximum acceptable skew M between any pair of clocks, need to synchronize at least once every: $M / (2 * \text{MDR})$ time units. ✓
 - Since time = Distance / Speed. ✓

Time and Clock Synchronization

Now, how to synchronize how often to synchronize the clock. So, the maximum drift rate MDR of a clock that is MDR is defined to be the relative coordinated universal time UTC is the correct time at any point of time and maximum drift rate MDR of any process depends upon the environment. So, the maximum drift rate between 2 clocks with the similar MDR is 2 times MDR.

So, given a maximum acceptable skew M between any 2 pair of the clocks they need to synchronize at least once every M divided by 2 times MDR time unit since the time is equal to the distance by speed.

(Refer Slide Time: 18:26)

External vs Internal Synchronization

- Consider a group of processes ✓
- **External synchronization** ✓
 - Each process $C(i)$'s clock is within a bounded D of a well-known clock S external to the group ✓
 - $|C(i) - S| < D$ at all times. ✓
 - External clock may be connected to UTC (Universal Coordinated Time) or an atomic clock.
 - **Example: Cristian's algorithm, NTP**
- **Internal Synchronization**
 - Every pair of processes in group have clocks within bound D ✓
 - $|C(i) - C(j)| < D$ at all times and for all processes i, j . ✓
 - **Example: Berkeley Algorithm, DTP** ✓

Time and Clock Synchronization

So, having understood these concepts, let us see the external versus internal synchronization methods of clock synchronization let us consider a group of processes in this scenario. So, external clock synchronization where each process C_i 's clock is within a bounded D of the well-known clock synchronization let us say S external to the group. So, the process says C_i is clock C_i minus S that is the external clock if we see the skew that is the difference and if it is less than D at all points of time which is a bound which is acceptable otherwise this external clock may be connected to the universal coordinated time or atomic clock.

So, this particular type of synchronization with the external clock or universal coordinated time is called external synchronization. Two algorithms, well known algorithms is based on the external synchronization they are called Cristian's algorithm and network time protocol whereas the internal synchronization is the synchronization among the pair of processes in the group of processes that is called internal synchronization.

So, every pair of processes in the group have the clocks within the bound D that is given two clocks within that particular system C_i minus C_j is bounded by D at all points of time for all processes i and j . So, this is called the internal synchronization where in the above example, we have seen for the external synchronization that every clock if you find out the skew with the external clock has to be bounded by D at all points of time that is called external synchronization. So, internal synchronization algorithms are called Berkeley algorithm and data center time protocol.

(Refer Slide Time: 20:35)

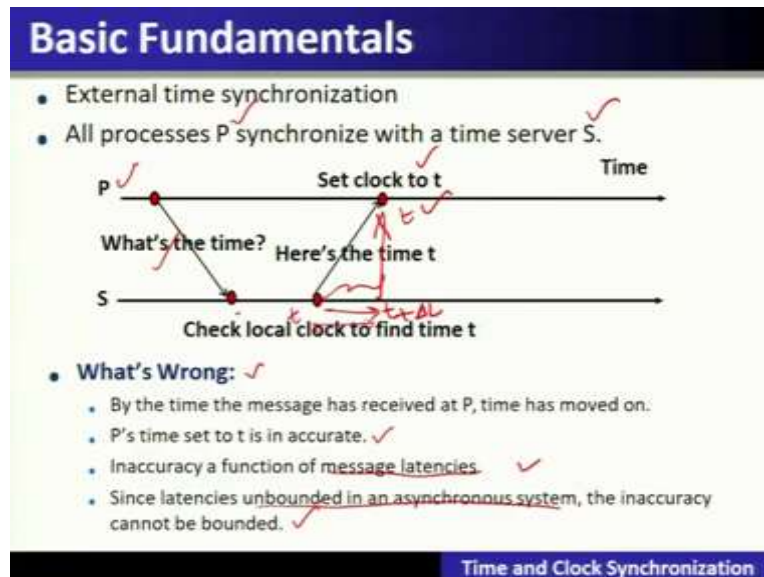
External vs Internal Synchronization

- **External synchronization with $D \Rightarrow$ Internal synchronization with $2 \cdot D$.**
- **Internal synchronization does not imply External Synchronization.**
 - In fact, the entire system may drift away from the external clock S !

Time and Clock Synchronization

So, external synchronization if you have achieved with a bound D this means that you have already achieved the internal synchronization with 2 times D . So, internal synchronization if you have done that does not imply the external synchronization is met. So, in fact the entire system may drift away from the external clock S that means that.

(Refer Slide Time: 20:59)



Let us have understood the basic fundamentals of external time synchronization let us say that all the processes P will synchronize their clock with a time server S which is a universal time. So, how that is, how that is all done we are going to understand using this particular diagram. So, a process P want to synchronize it is clock with a times server S what it will do, it will ask with a server S through a message what is the time at your clock server.

So, the server after receiving this message check the local clock to find the time t and it will send its time t back by a message and when the message is received the process P will set its time to t and this is way this way the process P has synchronize it is clock with the external clock but what is the wrong in this entire process.

So, you can see that by the time the message has received at P from S the time of S has moved on. So, this particular time when the clock, when P is synchronizing its clock that time of S has already moved from t to let us say some little Δt time so it is not same. So, this is inaccurate. So, inaccuracy here is a function of message delays. So, this particular message delay that is the time when it takes to reach the message that is called the message delay or a message latency is the error here in this particular external synchronization.

Now, these latencies are unbounded in asynchronous system means that when the message will be delivered that is not known in an asynchronous system the inaccuracy is also unbounded in the asynchronous situation.

(Refer Slide Time: 23:07)

(i) Christians Algorithm

- P measures the round-trip-time RTT of message exchange
- Suppose we know the minimum $P \rightarrow S$ latency \min_1
- And the minimum $S \rightarrow P$ latency \min_2
 - > \min_1 and \min_2 depends on the OS overhead to buffer messages, TCP time to queue messages, etc.
- The actual time at P when it receives response is between $[t + \min_2, t + \text{RTT} - \min_1]$

Time and Clock Synchronization

(i) Christians Algorithm

- P measures the round-trip-time RTT of message exchange
- Suppose we know the minimum $P \rightarrow S$ latency \min_1
- And the minimum $S \rightarrow P$ latency \min_2
 - > \min_1 and \min_2 depends on the OS overhead to buffer messages, TCP time to queue messages, etc.
- The actual time at P when it receives response is between $[t + \min_2, t + \text{RTT} - \min_1]$

Time and Clock Synchronization

Christians Algorithm

- The actual time at P when it receives response is between $[t + \text{min2}, t + \text{RTT} - \text{min1}]$
- P sets its time to halfway through this interval
 - To: $t + (\text{RTT} + \text{min2} - \text{min1}) / 2$
- Error is at most $(\text{RTT} - \text{min2} - \text{min1}) / 2$
 - Bounded ✓

Time and Clock Synchronization

Let us see the Christian's algorithm how it does this external synchronization algorithm. So, here the P now measures the round trip time of the message exchange, round trip time means that when the P has send a message it will note down it is time and let us say at time t of the clock S it received the message and then it will send the message let us say at time t 2 back and it has received the message let us say tp1 tp2.

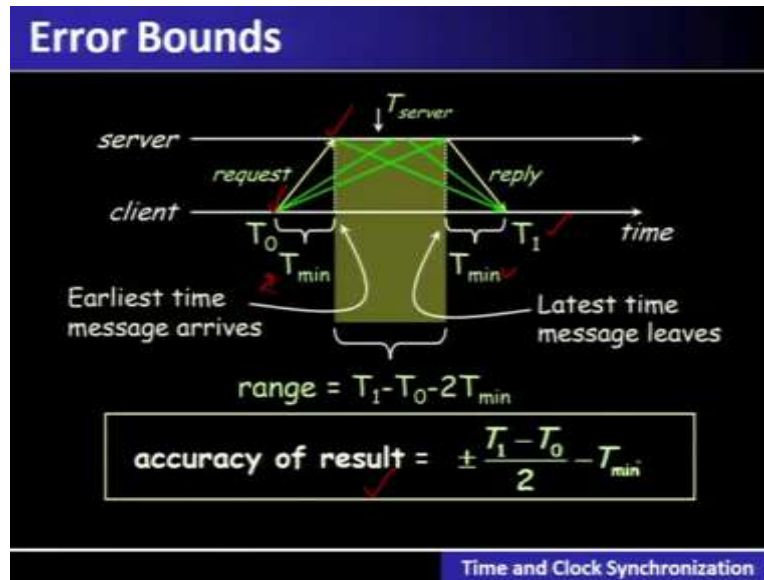
So, tp2 minus tp1 becomes the round trip time the time when it takes to send to S and S gives back the reply this is called round trip time. So, let us see that P measures the round trip time of the message exchange. So, measuring the round trip time is by sending several message and noting down RTT and finding the average of it this is how the Christian algorithm does suppose we know that minimum of P to S latency is let us say minimum 1.

So, this is S to P latency, P to S latency similarly and the minimum from S to P this latency let us say it is minimum 2. So, minimum 1 and minimum 2 latencies they depends upon the operating system overhead also to buffer the messages and TCP also time to give the message and so on. So, the actual time at P when it receives the response is between T plus T is the time here. So, let us understand this. So, the actual time at P when it receives the message that is the response is nothing but t plus minimum 2.

So, this is minimum 2 range from t plus minimum 2 to t plus this is time t, t plus RTT minus minimum 1. So, that you can see from this particular figure so the Christian's algorithm what it does is that it will set the P it is time to the halfway through this particular interval that is divided by 2. So, the error here is at most RTT minus min 1 min 2 minus min 1 divided by 2 that is half of this particular RTT minus of minimum 1 plus minimum 2. So, this is min 1 and

this is min 2 so RTT is this one so RTT minus of min 1 plus min 2 this becomes the error and this is bounded here in this case in the Cristian's algorithm.

(Refer Slide Time: 26:45)



So, here you can also understand through this particular example the Cristian's algorithm error bound so the client will send the message let us say time T_0 and let us say that the message is delivered to the server with a delay of T_{min} of the T_{min} time then the server will give a time at that is called server time T and this particular message by which it is sending the reply we will take T_{min} minimum delay to reach at time T_1 .

So, therefore this particular time which is elapsed at the server is to be noted as T_1 minus T_0 minus 2 of T_{min} considering that T_{min} and so one-way latency is let us say T_{min} in this case. So, that particular in accuracy of a result is T_1 minus T_0 divided by 2 minus T_{min} and that is all bounded.

(Refer Slide Time: 27:53)

Error Bounds

- **Allowed to increase clock value but should never decrease clock value**
 - May violate ordering of events within the same process.
- **Allowed to increase or decrease speed of clock**
- **If error is too high, take multiple readings and average them**

Time and Clock Synchronization

So, error bounds here you can see that they are allowed to increase the clock values. For example, if you know the clock synchronization how to synchronize the clocks. So, you are only allowed to increase the clock value and you are not allowed to decrease the clock value because the time never goes back. So, may violate the ordering of events within the same process if let us say you want to decrease you are decreasing the clock value so you are allowed only to increase but as far as speed is concerned you are allowed to increase or decrease the speed. Here in this case if the error is too high, then you take multiple readings and average them.

(Refer Slide Time: 28:36)

Christians Algorithm: Example

- Send request at 5:08:15.100 (T_0)
- Receive response at 5:08:15.900 (T_1)
 - Response contains 5:09:25.300 (T_{server})
- Elapsed time is $T_1 - T_0$
 - 5:08:15.900 - 5:08:15.100 = 800 msec
- Best guess: timestamp was generated
 - 400 msec ago
- Set time to $T_{server} +$ elapsed time
 - 5:09:25.300 + 400 = 5:09:25.700

If best-case message time = 200 msec
 $T_0 = 5:08:15.100$
 $T_1 = 5:08:15.900$
 $T_{server} = 5:09:25.300$
 $T_{min} = 200\text{msec}$

$$Error = \pm \frac{900 - 100}{2} - 200 = \pm \frac{800}{2} - 200 = \pm 200$$

Time and Clock Synchronization

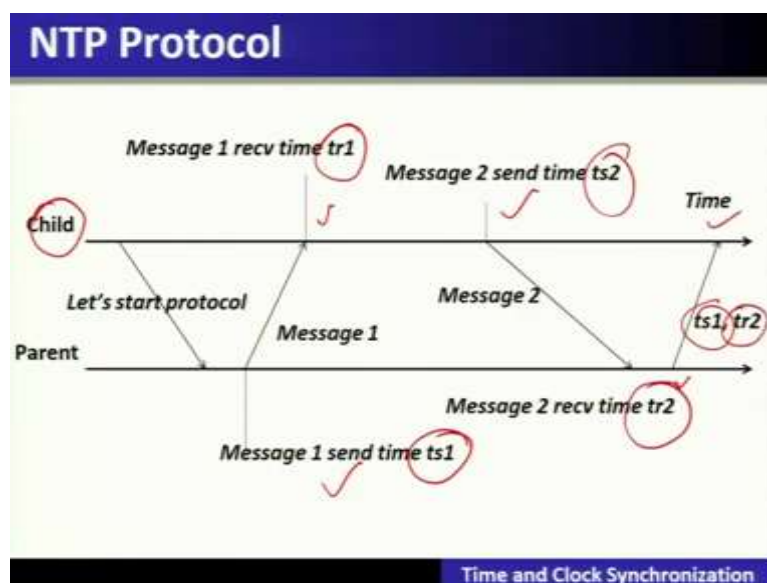
(ii) NTP: Network time protocol ✓

- (1991, 1992) Internet Standard, version 3: RFC 1305
- **NTP servers organized in a tree.** ✓
- Each client = a leaf of a tree.
- Each node synchronizes with its tree parent ✓

Time and Clock Synchronization

So, another example of a Christian's algorithm is shown over here that we have already explained. So, let us see the next protocol for external synchronization that is called network time protocol NTP. So, NTP is used in the internet for time synchronization among their servers or network devices. So NTP servers is organized in the form of a tree and each child is a leaf of a tree that is shown and each node synchronizes with its parent in a tree. So, the first server is called the primary server. Second level servers are called secondary server and tertiary server and finally you have the client. So, this NTP servers are organized in a form of a tree.

(Refer Slide Time: 29:23)



Now, let us see the NTP protocol. Here the child will say let us start a protocol send a message and the parent will receive the message and parent will reply the message at time t_s

1 which message will be received at time t_{r1} by the child and then child will send another message to at time t_{s2} . So this message will be replied back by parent at time t_{r2} and which will be received here by the child 1. So, now you have several time stamps t_{s1} , t_{r1} then t_{r2} , t_{s2} and also t_{s1} and t_{r2} is also being send.

(Refer Slide Time: 30:10)

Why $o = (t_{r1} - t_{r2} + t_{s2} - t_{s1})/2$?

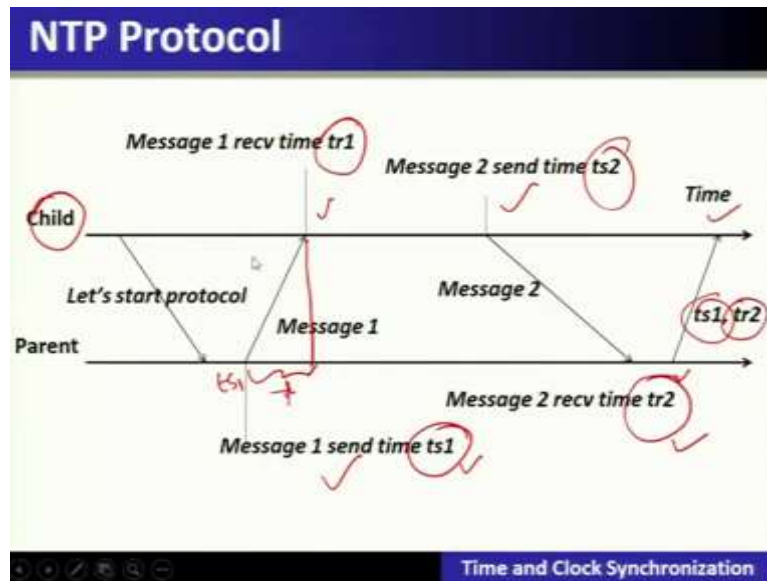
- Offset $o = (t_{r1} - t_{r2} + t_{s2} - t_{s1})/2$
- Let's calculate the error.
- Suppose real offset is **oreal**
 - Child is ahead of parent by oreal.
 - Parent is ahead of child by $-oreal$.
- Suppose one way latency of Message 1 is $L1$.
($L2$ for Message 2)
- No one knows $L1$ or $L2$!
- **Then**
 - $t_{r1} = t_{s1} + L1 + oreal$
 - $t_{r2} = t_{s2} + L2 - oreal$

Time and Clock Synchronization

Why $o = (t_{r1} - t_{r2} + t_{s2} - t_{s1})/2$?

- **Then**
 - $t_{r1} = t_{s1} + L1 + oreal$.
 - $t_{r2} = t_{s2} + L2 - oreal$.
- **Subtracting second equation from first**
 - $oreal = (t_{r1} - t_{r2} + t_{s2} - t_{s1})/2 - (L2 - L1)/2$
 - $\Rightarrow oreal = o + (L2 - L1)/2$
 - $\Rightarrow |oreal - o| < |(L2 - L1)/2| < |(L2 + L1)/2|$
 - Thus the error is bounded by the round trip time (RTT)

Time and Clock Synchronization



You have four different timestamps and using offset you can calculate the difference offset o t_r1 minus t_r2 plus t_s2 minus t_s1 and you can calculate the error let us say that real offset is $oreal$. So, child is ahead of a parent by $oreal$, parent is ahead by $minus\ oréal$, suppose one-way latency of a message one is L_1 for L_2 is 1 message 2 no one knows for L_2 and L_1 and L_2 . So, there are two equations. Now, using these two equations, you can solve $oreal$.

So, $oreal$ value comes out to be this particular equation which is nothing but the offset o which we have simplified so $oreal$ minus o will become this particular value which is bounded by the round trip time. So, if you want to understand again so you can see here that this is the timestamp when server 1 has send, this is the timestamp of server 1 t_s1 and t_r2 , t_r2 and child has send received this message as t_r1 and t_s2 .

So, these are the message let us see from this equation that what is the offset. So, what you will form is t_r1 is equal to t_s1 plus L_1 . So, t_r1 is equal to so, the t_r1 is equal to you can see that the time when t_s1 has send plus this delay. So, that is what is being added over here the same concept which we have seen L_1 plus $oreal$ similarly, t_r2 is also given. So, these equations you will solve here in this case and this particular equation is now can see that this error is bounded in network time protocol.

(Refer Slide Time: 32:26)

(iii) Berkeley's Algorithm

- **Gusella & Zatti, 1989**
- Master polls each machine periodically
 - Ask each machine for time
 - Can use Christian's algorithm to compensate the network's latency.
- When results are in compute,
 - Including master's time.
- **Hope: average cancels out individual clock's tendency to run fast or slow**
- Send offset by which each clock needs adjustment to each slave
 - Avoids problems with network delays if we send a time-stamp.

Time and Clock Synchronization

Now, you have internal time synchronization protocol called Berkeley's algorithm here the master polls each machine periodically asks each machine for the time now can use the Christian's algorithm to compensate the networks latency. So, Berkeley's algorithm is given by gusella and zatti in 1989. So, it starts with the master polls each machine periodically asks each machine for the time for that it use it can use the Christian's algorithm to compensate the network latency.

Now, when the results are in compute which includes the masters time and here the average cancels out the individual clock's tendency to run fast or slow send the offset by which each clock needs adjustment to each slave.

(Refer Slide Time: 33:21)

Berkley's Algorithm : Example

1. Request timestamps from all slaves

2. Compute fault-tolerant average:

$$\frac{3:25 + 2:50 + 3:00}{3} = 3:05$$

3. Send offset to each client

Internal clock speed

Time and Clock Synchronization

You can understand the Berkeley's algorithm working through this running example. So, as we have pointed out that Berkeley's is internal synchronization, algorithm, internal clock synchronization here in this example, we are having 1 2 3 4 different clocks running out of which one is the master which polls with all others clock values that is called slaves.

So, let us say that clock 2, clock 3, clock 4 they will send their timestamp values or clock values to the master which is having the clock value 3. So, what it does is it will find out all other clock values except the clock value coming from 4 which is detected as the anomaly it will not be considered an averaging. So, excluding the clock value of clock 4 they will compute the fault tolerant average of all other clock values.

There are 3 different clock values whose average is being calculated as 3 or 5. So, what it will do it will adjust its clock while you accordingly and now it will calculate the difference for this is a 20 minutes difference for this is plus 15 and this value also will be send as the difference and it will no offset as the difference. So, it will send the offset to each client in this case.

(Refer Slide Time: 35:16)

(iv) DTP: Datacenter Time Protocol

Globally Synchronized Time via Datacenter Networks

Ki Suh Lee, Han Wang, Vishal Shrivastav, Hakim Weatherspoon
Computer Science Department
Cornell University
kslee,hwang,vishal,hweathers@cs.cornell.edu **ACM SIGCOMM 2016**

Application
Transport
Network
Data Link
Physical

- DTP uses the physical layer of network devices to implement a decentralized clock synchronization protocol.
- **Highly Scalable with bounded precision!**
 - ~25ns (4 clock ticks) between peers
 - ~150ns for a datacenter with six hops
 - No Network Traffic
 - Internal Clock Synchronization
- End-to-End: ~200ns precision!

Time and Clock Synchronization

DTP: Phases



- Runs in two phases between two peers
 - Init Phase: Measuring OWD (one-way delay)
 - Beacon Phase: Re-Synchronization



Time and Clock Synchronization

DTP: (i) Init Phase

- **INIT phase:** The purpose of the INIT phase is to measure the one-way delay between two peers. The phase begins when two ports are physically connected and start communicating, i.e. when the link between them is established.
- Each peer measures the one-way delay by measuring the time between sending an INIT message and receiving an associated INIT-ACK message, i.e. measure RTT, then divide the measured RTT by two.



- $delay = (t_4 - t_1 - \alpha) / 2$
 - $\alpha=3$: Ensure *delay* is always less than actual delay
- Introduce 2 clock tick errors
 - Due to oscillator skew, timing and Sync FIFO



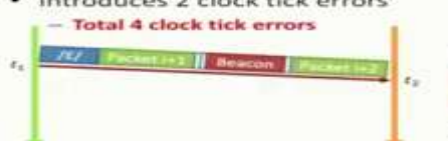
Time and Clock Synchronization

DTP: (ii) Beacon Phase

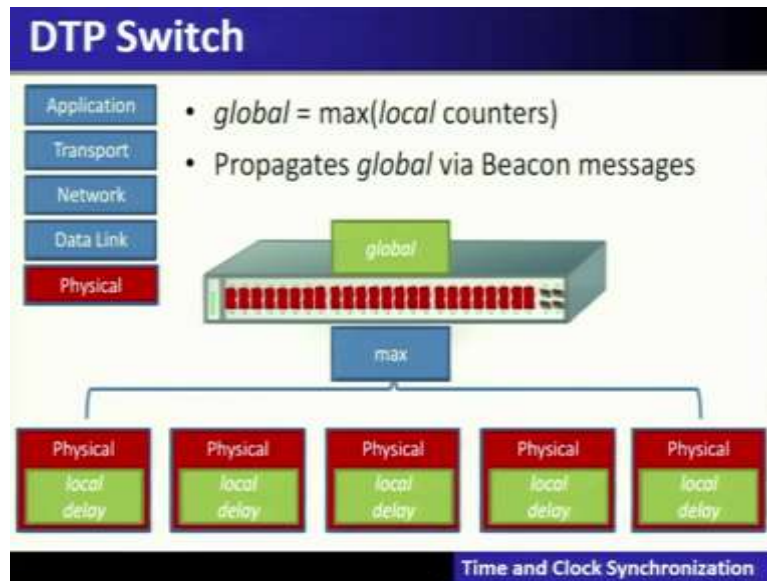
- **BEACON phase:** During the BEACON phase, two ports periodically exchange their local counters for resynchronization. Due to oscillator skew, the offset between two local counters will increase over time. A port adjusts its local counter by selecting the maximum of the local and remote counters upon receiving a BEACON message from its peer. Since BEACON messages are exchanged frequently, hundreds of thousands of times a second (every few microseconds), the offset can be kept to a minimum.



- $local = \max(local, remote + delay)$
- Frequent messages
 - Every 1.2 us (200 clock ticks) with MTU packets
 - Every 7.2 us (1200 clock ticks) with Jumbo packets
- Introduces 2 clock tick errors
 - Total 4 clock tick errors



Time and Clock Synchronization



But Yet...

- We still have a non-zero error! ✓
- We just can't seem to get rid of error
 - Can't as long as messages latencies are non-zero.
- Can we avoid synchronizing clocks altogether, and still be able to order events?

Physical clock
 ↑
 Extend Interval
 UTC DTP
 NSP

Time and Clock Synchronization

Let us see another critical clock synchronization that is called internal synchronization which is happening inside the data center that clock synchronization is called data center time protocol. So, you know that in a data center or in a cloud data center hundreds and thousands of the nodes they are sending the message with each other through the network that is called cloud network.

So, these devices are running let us say the protocol and let us see that these particular devices also need to be clock synchronized. And therefore, data center time protocol is an essential requirement. So, that all those nodes or the server inside the data center are synchronized. We are not going in detail but you will see that it uses the internal clock synchronization and it runs in two phases between two peers that is they are running this stack.

We are interested only in the physical layer because that is the hardware and that particular message passed its time exchange its time and now then it will calculate this particular bound of a precision and accuracy. Now, what we have so far seem is the physical clock synchronization and there are two types of physical clock synchrony one is the external synchronization with the universal time clock and this is the internal synchronization algorithms which we have seen in the terms of data center time protocol.

And here we have seen network time protocol as external synchronization algorithms. In every method we have seen a nonzero error. So, therefore, you know that you just cannot seem get rid of these errors because of the message latency which are nonzero. So, how can you avoid synchronizing the clocks all together yet you are able to use some other method for ordering the events in the distributed system.

(Refer Slide Time: 37:35)

Ordering events in a distributed system

- To order events across processes, trying to synchronize clocks is an approach.
- What if we instead assigned timestamps to events that were not absolute time?
- As long as those timestamps obey **causality**, that would work
 - If an event A causally happens before another event B, then $\text{timestamp}(A) < \text{timestamp}(B)$
 - Example: Humans use causality all the time
 - I enter the house only if I unlock it
 - You receive a letter only after I send it

Time and Clock Synchronization

So, ordering the events in a distributed system without physical clock synchronization is a method which is widely used in the in various applications. So, let us understand this and it can be applicable as well in the internet that is a network scenario, internet that is IoT network scenario. So, in order to order the events across the processes so we consider the IoT devices running these processes trying to synchronize the clock in an approach.

So, what if we have assigned the timestamp to the events that are not that were not absolute time, absolute means not according to the physical clock time, as long as these timestamps obey causality that would work. For example, if an event A causally happened before another

event B then any timestamp of A should be less than the timestamp of B. So, humans are used the causality at all points of time for example, I enter the house only if I unlock it.

So, without knowing the clock values you know that which event has happened before which other event. Similarly, if you have received a letter only after you send it so sending of a letter happened before you receive the letter so without even having the physical clock ordering the events is possible using the happen before relation and that is called causality relation.

(Refer Slide Time: 39:13)

Logical (or Lamport) ordering

- **Proposed by Leslie Lamport in the 1970s.**
- Used in almost all distributed systems since then
- Almost all cloud computing systems use some form of logical ordering of events.

Leslie B. Lamport (born February 7, 1941) is an American computer scientist. Lamport is best known for his seminal work in distributed systems and as the initial developer of the document preparation system LaTeX. Leslie Lamport was the winner of the **2013 Turing Award** for imposing clear, well-defined coherence on the seemingly chaotic behavior of distributed computing systems, in which several autonomous computers communicate with each other by passing messages.

Time and Clock Synchronization

So, using this concept of causal relation of ordering the events without physical clock without physical time is the concept called logical ordering or this kind of ordering is called a Lamport's clock and given by a famous scientist, which is called Leslie Lamport. So, Leslie Lamport has proposed this particular notion which is used almost in all distributed system and a cloud computing system and even in the IoT systems.

So, about the Leslie Lamport he is an American scientist. And this work of time and ordering in a distributed system was a seminal work and is being recognized as 2013 Turing award this is the highest award in computer science any scientists can get.

(Refer Slide Time: 40:11)

Lamport's research contributions

- Lamport's research contributions have laid the foundations of the theory of distributed systems. Among his most notable papers are
 - "Time, Clocks, and the Ordering of Events in a Distributed System", which received the PODC Influential Paper Award in 2000.
 - "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", which defined the notion of Sequential consistency,
 - "The Byzantine Generals' Problem",
 - "Distributed Snapshots: Determining Global States of a Distributed System" and
 - "The Part-Time Parliament".
- These papers relate to such concepts as logical clocks (and the *happened-before* relationship) and Byzantine failures. They are among the most cited papers in the field of computer science and describe algorithms to solve many fundamental problems in distributed systems, including:
 - the Paxos algorithm for consensus,
 - the bakery algorithm for mutual exclusion of multiple threads in a computer system that require the same resources at the same time,
 - the Chandy-Lamport algorithm for the determination of consistent global states (snapshot), and
 - the Lamport signature, one of the prototypes of the digital signature.

Time and Clock Synchronization

Logical (or Lamport) Ordering(2)

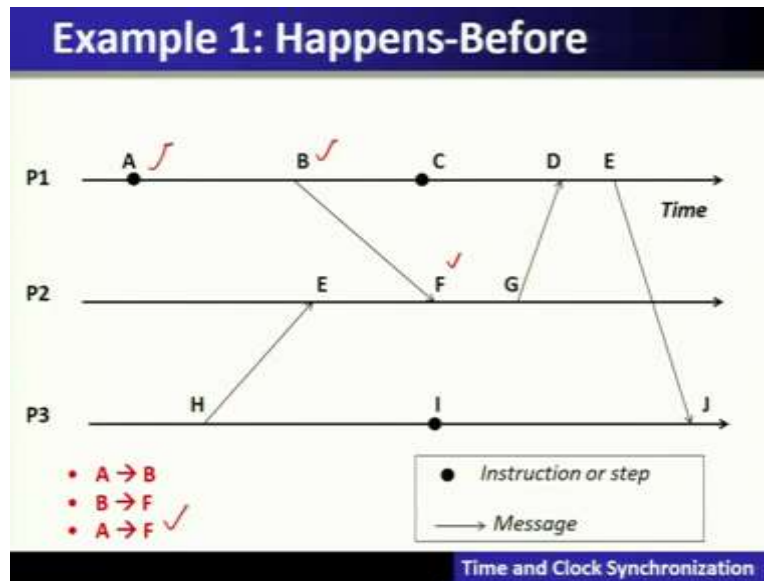
- Define a logical relation *Happens-Before* among pairs of events
- Happens-Before* denoted as \rightarrow
- Three rules:
 - On the same process: $a \rightarrow b$, if $time(a) < time(b)$ (using the local clock)
 - If p1 sends m to p2: $send(m) \rightarrow receive(m)$
 - (Transitivity) If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- Creates a *partial order among* events
 - Not all events related to each other via \rightarrow

Time and Clock Synchronization

Example 1:

● Instruction or step
→ Message

Time and Clock Synchronization



So, these are some of this is the paper which has received this particular award the title of a paper is time clock and ordering of events in a distributed system. Now, let us understand this work in more detail to appreciate this work and how that is all done without having the physical clocks synchronized yet you can order the events using the mechanism called Lamport's clock.

So, define the logical relation that is called happened before relation among the pair of events and which is noted down by the that relation happened before relation. So, this happened before relation follows three different rules, the first says that on any on the same process if there are two events a and b. So, if you see the local time of a and the local time of b so the time of a is always less than b this means that a has happened before b.

So, a happened before b if and only if the timestamp of a is less than timestamp b and this time stamp is taken from the local clock. Similarly, when p 1 sends the message m to p 2. So, in that case send a message has happened before the receive of a message. So, even without having clocks synchronized. So, this event can be send and receive can order the event these are external event and these two events you can also form a transitive relation that if a has happened before b, b has happened before c, then c, a is happened before c.

Now, these three relations they create a partial order relation among all the events in the distributed system and this particular relation is applicable in all the events happening in the distributed system. Let us take this example in this example you have three different IoT devices let us say p 1, p 2, p 3, at different locations. So, p 1 is generating these events A B C D E so A is the internal event, B is send off a message.

Similarly, D is the received of a message, E is the sender of a message similarly, P 2 also has the events happening. Now, you can see that A has happened before B why because it is internal events. So, that is similarly B has happened before F and A has happened before F why because A has happened before B is internal event B has send a message so received of a message. So, sending of a message happened before the receive of a message using Lampert's clock rule two. So, if you include this A, B and F so A happened before using transitive relation. So, likewise you can order the event in this particular manner. So, this is called a Lampert's timestamp.

(Refer Slide Time: 43:11)

Lamport timestamps

- Goal: Assign logical (Lamport) timestamp to each event ✓
- Timestamps obey causality ✓
- Rules
 - Each process uses a local counter (clock) which is an integer
 - initial value of counter is zero ✓
 - A process increments its counter when a **send** or an **instruction** happens at it. The counter is assigned to the event as its timestamp. ✓
 - A **send (message)** event carries its timestamp ✓
 - For a **receive (message)** event the counter is updated by
 - ✓ ✓ ✓

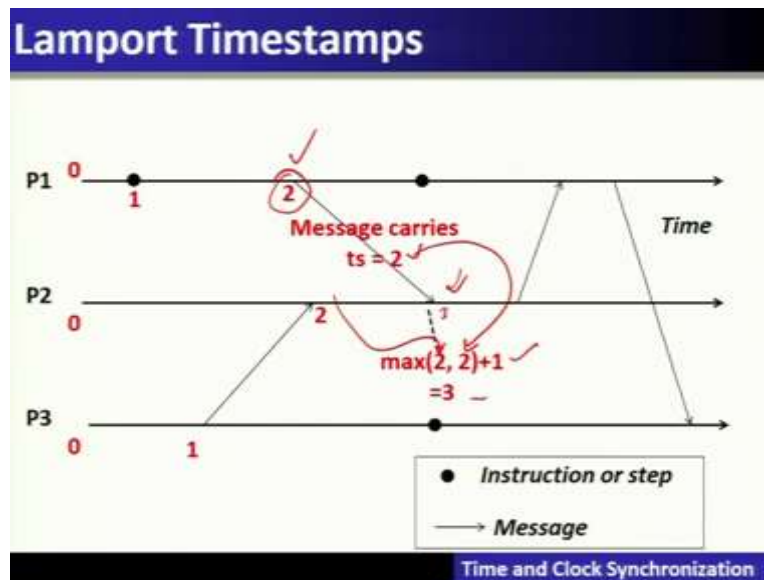
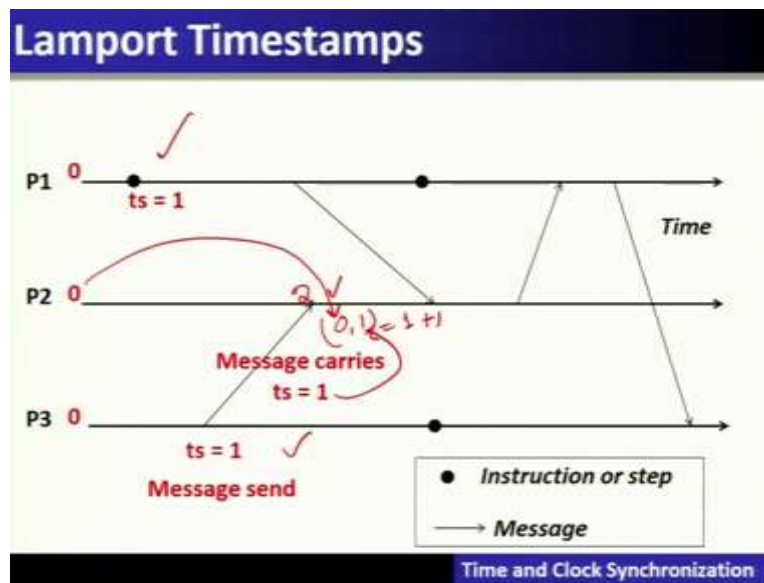
$\max(\text{local clock}, \text{message timestamp}) + 1$

Time and Clock Synchronization

So, the goal of Lampert's timestamp is to assign the logical or a Lamport or timestamp to each events and these timestamps, you know, that obey causality relations using these rules. So, each process uses a local counter, which is nothing but an integer value initial value of the counter is 0. So, a process increments it to counter when the send or an instruction happens at it, the counter is assigned to the event as the timestamp.

So, this is called internal event rule, which we have seen in the previous happened before relation, which is also defined here in the Lampert's timestamp. Similarly, the send of a message event carries its timestamp and receive event carries the timestamp and when the message is received then the clock can be synchronized using the following rule. So that is nothing but the maximum of local clock and the message time stamp plus 1 is done.

(Refer Slide Time: 44:16)

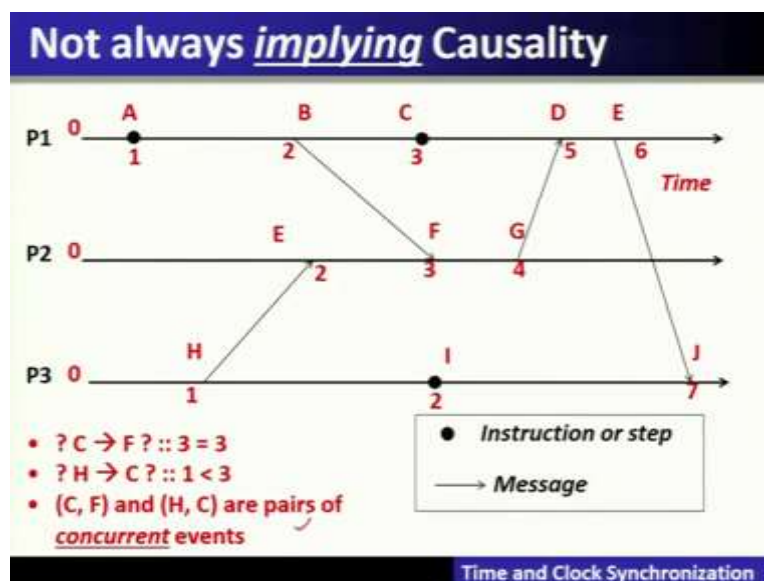
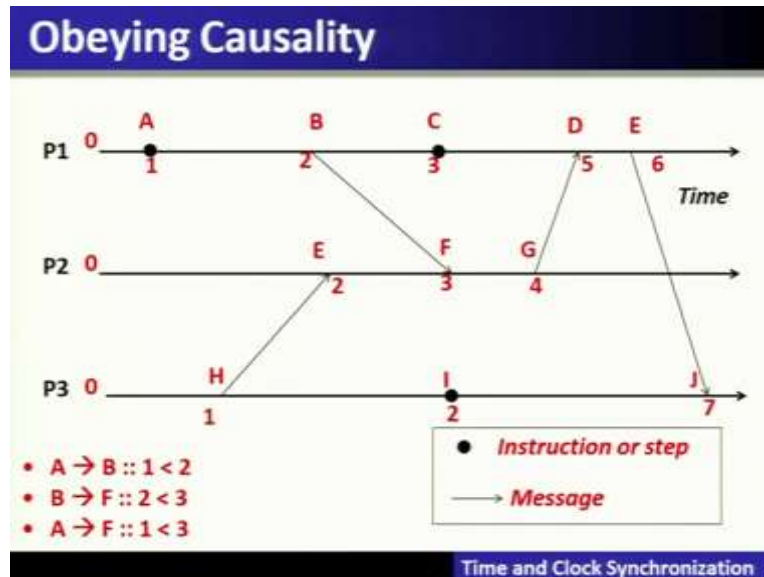


Take this example, that initially all three values are 0 clock values. So the timestamp of first internal event is $ts = 1$ and here $ts = 1$ at p_3 send off an event. So, when this particular message is received at p_2 so it will carry the timestamp of p_3 as 1. So using that particular rule the maximum of 0 and 1, 1 is coming from this particular message and 0 is the local clock time so maximum of 0 and 1 is 1 so it will be the value as 2.

So, you can see that from 0, the clock value will be jumped to 2 directly. So, that will happen here. So, that is what is shown here in this particular example, you can see another example, that the clock value which is being carried by the send message that is $ts = 2$, which is the clock value of p_1 and p_1 when it received over here the clock value of p_1 is 2 and the local

clock is also 2. So, the maximum of 2 and 2 will become 2 plus 1 that is called 3. So, this clock value will become 3.

(Refer Slide Time: 45:38)



So, all of the clock values are calculated here in this case so you can see that this particular logical clock obey the causality relation and you can order the event here in this case but this causality is not always applicable on all the events. So, those events which do not obey these relations then they are called concurrent events here in this case.

(Refer Slide Time: 46:11)

Concurrent Events

- A pair of concurrent events doesn't have a causal path from one event to another (either way, in the pair)
- Lamport timestamps not guaranteed to be ordered or unequal for concurrent events
- Ok, since concurrent events are not causality related!
- **Remember:**

$E1 \rightarrow E2 \Rightarrow \text{timestamp}(E1) < \text{timestamp}(E2)$, **BUT**
 $\text{timestamp}(E1) < \text{timestamp}(E2) \Rightarrow$
 $\{E1 \rightarrow E2\} \text{ OR } \{E1 \text{ and } E2 \text{ concurrent}\}$

Time and Clock Synchronization

Vector Timestamps

- Used in key-value stores like Riak ✓
- Each process uses a vector of integer clocks
- Suppose there are N processes in the group 1...N
- Each vector has N elements
- Process i maintains vector $V_i[1...N]$ ✓
- j th element of vector clock at process i , $V_i[j]$, is i 's knowledge of latest events at process j ✓

Time and Clock Synchronization

So, pair of concurrent event does not have the causal path from one event to another event. So, Lamport's timestamp does not guarantee to be ordered or unequal. Hence, the concurrent event are not causally related events. So, therefore, with this drawback another time stamp is being proposed in the distributed system which is called a vector timestamp and it is used in the cloud databases which we have discussed as the no equal database in the lecture 1.

So, this particular database uses the vector time is time. So, each process uses the vector of integer clock suppose there are N processes in a group so N elements will be there. So, process i maintains a vector v_i 1 to N so j th element of a vector clock at process i will be of V_i of J is i th knowledge of the latest event happened at process j .

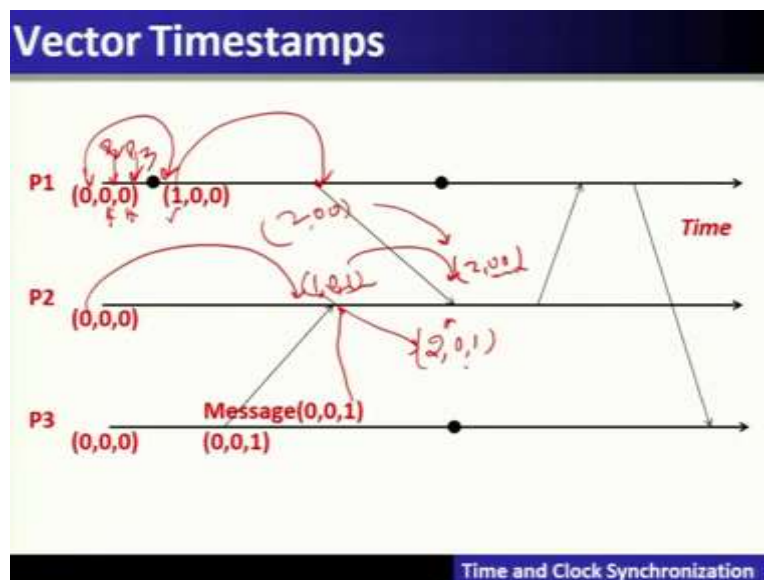
(Refer Slide Time: 47:08)

Assigning Vector Timestamps

Incrementing vector clocks

1. On an instruction or send event at process i , it increments only its i th element of its vector clock ✓
2. Each message carries the send-event's vector timestamp $V_{\text{message}}[1...N]$ ✓
3. On receiving a message at process i :
 $V_i[i] = V_i[i] + 1$ ✓
 $V_i[j] = \max(V_{\text{message}}[j], V_i[j])$ for $j \neq i$ ✓

Time and Clock Synchronization



So, implementing the vector clock so here you can see that on an instruction, on an instruction or a send event of a process i it increments only its i th element of a vector clock and message when it receives a message. So, each message carries the send event's vector timestamp and both these events are used together to synchronize the clocks for example on receiving the message at a process i .

So, it is internal event or it is internal clock that is called v_i of i will be incremented by 1 whereas, all other clock values in the vector will be now synchronized using this particular equation that is V_i 's knowledge of all other clock values is a maximum of the clock values which is brought into by the process by the message and its own knowledge whichever is the

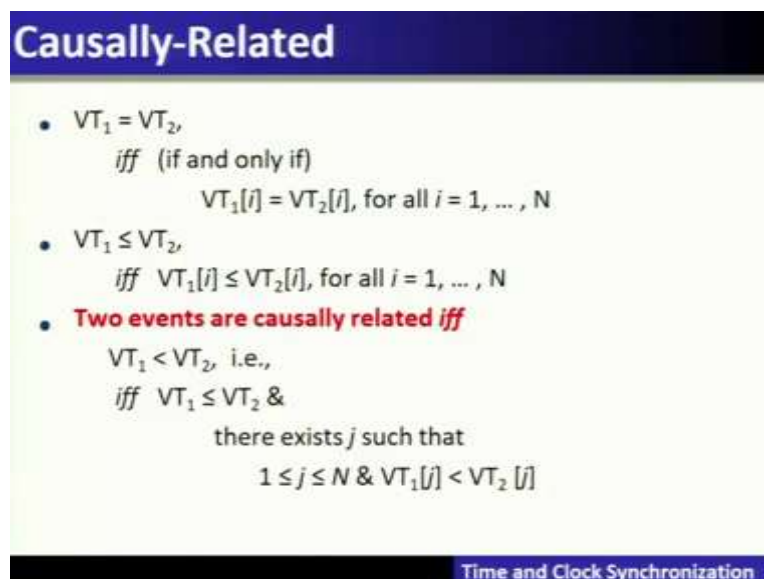
maximum will be taken up except for its own message. So, this is called vector clock and this we will understand using this particular example.

For example, that p 1 initially all the clocks are 0 0 0. Why because this is the clock value of p 1, this is p 2, this is p 3, so p 1's knowledge of p 2 is 0 having it is timestamp 0 and here for p 3 is 0. Now, if p 1 will do its internal event, then this clock value will be incremented by 1 only. Similarly, when p 1 will send a message so send of an event will also increment the clock value. So, it will become 2 0 0 and this value will reach over here.

Now, when the value will reach over here, what p 2 will do p 2 has these values 0 0 0 initially when it receives a message and the message is 0 0 1. So, it will synchronize its clock as this particular internal event will become 1 0 and this is being seen as the most recent values and when the message is received over here then this particular value will be updated.

So, how it will be updated you can see that this is the receipt of a message. So, internal event will increase. So, it will become 2 and what about 0 this particular message is carrying 2 0 0. So, this is incremented to 2 whereas between 0 and 1 and 0 0. So, this 0 and 1 will be more updated value over here.

(Refer Slide Time: 50:06)



Causally-Related

- $VT_1 = VT_2$,
iff (if and only if)
 $VT_1[i] = VT_2[i]$, for all $i = 1, \dots, N$
- $VT_1 \leq VT_2$,
iff $VT_1[i] \leq VT_2[i]$, for all $i = 1, \dots, N$
- **Two events are causally related iff**
 $VT_1 < VT_2$, i.e.,
iff $VT_1 \leq VT_2$ &
there exists j such that
 $1 \leq j \leq N$ & $VT_1[j] < VT_2[j]$

Time and Clock Synchronization

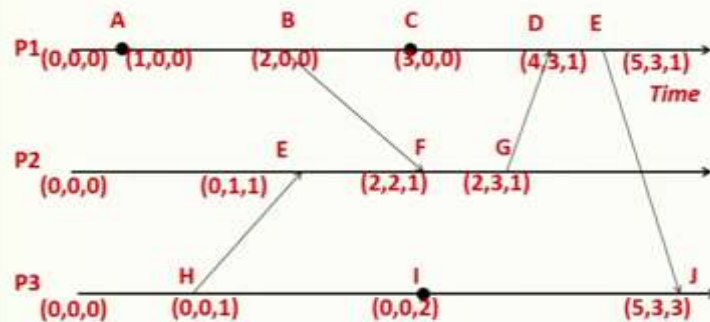
... or Not Causally-Related

- Two events VT_1 and VT_2 are **concurrent**
iff
 $\text{NOT } (VT_1 \leq VT_2) \text{ AND NOT } (VT_2 \leq VT_1)$

We'll denote this as $VT_2 ||| VT_1$

Time and Clock Synchronization

Identifying Concurrent Events



Time and Clock Synchronization

Conclusion

Internet of Things (IoT) devices that are wirelessly connected in mesh networks often need mutual clock time synchronization, to enable chronological ordering of sensor events, coordination of asynchronous processes across devices, or network-wide coordination of actuators. ✓

Time synchronization: ✓

Christian's algorithm

Berkeley algorithm

NTP

DTP

But error a function of RTT

Can avoid time synchronization altogether by instead assigning logical timestamps to events

Time and Clock Synchronization

So, that is what you know that vector clock. So, vector clocks follow the causally related. So, if the 2 events are causally related, if and only if this particular vector condition holds for two events, they are concurrent, then these less than relation does not hold so they obey the causality and if two events are causally related, then you can think of it is happening the other one.

So, therefore let us conclude with this discussion that internet of things devices that are wirelessly connected in the mesh often needs the mutual clock time synchronization to enable the chronological ordering of sensor events coordination of asynchronous processes across the devices or the network wide coordination of actuators. So, this is very much required, because when the sensor sends the data it also attaches its timestamp.

So, all the sensor locations all the sensor devices running at different location requires their clock to be synchronized. So, therefore, different time synchronization algorithms we have seen which is applicable in the internet of things as well. And we have seen the two algorithms Cristian's algorithm, Berkeley's algorithm network time protocol, data center time protocol and we have also seen that there is an error condition and therefore with these error possibilities and synchronization very frequently and it is taking time in the internet of things scenarios. So, we have also seen a method of logical clock synchronization in the distributed system. Thank you.