**Q**. Name and explain at least three sources of delay that can be introduced between WWV broadcasting the time and the processors in a distributed system setting their internal clocks.

**Ans:**

**1. Propagation Delay**

Propagation delay is the time it takes for a signal to travel from the WWV transmitter to the receiver in the distributed system.

This delay depends on factors such as the distance between the transmitter and the receiver and the speed at which the radio waves propagate through the atmosphere.

**Factors Affecting Propagation Delay**:

**Distance**: Longer distances between the transmitter and receiver result in longer propagation times.

**Atmospheric Conditions**: Variations in atmospheric conditions can affect the speed of radio waves, causing slight fluctuations in propagation delay.

**2. Reception and Processing Delay:**

Once the **time signal** is received by a processing unit, there are additional delays introduced by the **hardware** and **software** involved in **decoding** and **processing** the signal.

**Factors Affecting Reception and Processing Delay**:

**Receiver Hardware**: The quality and design of the receiver hardware can impact how quickly it processes the incoming time signal.

**Software Processing**: The software that handles the reception and decoding of the time signal may introduce delays due to its processing algorithms and efficiency.

1. **Network Delay**:

In distributed systems, network delay refers to the time it takes for synchronization messages or requests to travel between nodes in the system.

This delay can affect the accuracy of time synchronization if the system relies on network communication to disseminate the time signal or synchronize clocks.

**Factors Affecting Network Delay**:

**Network Traffic**: High traffic can increase delays and affect the timeliness of synchronization messages.

**Network Topology**: The physical and logical layout of the network can impact how long it takes for messages to traverse from one node to another.

**Congestion and Latency**: Variations in network congestion and latency can introduce discrepancies in the timing information received by different nodes.

These delays collectively impact the precision of time synchronization across a distributed system. To mitigate these issues, distributed systems often use protocols like **NTP** (Network Time Protocol) or **PTP** (Precision Time Protocol) that account for and adjust these delays to maintain accurate time synchronization.

Q: Consider the behaviour of two machines in a distributed system. Both have clocks that are supposed to tick 1000 times per millisecond. One of them actually does, but the other ticks only 990 times per millisecond. If UTC updates come in once a minute, what is the maximum clock skew that will occur?

**Ans**:

In a distributed system where two machines have clocks ticking at different rates, the maximum clock skew can be calculated based on the difference in their ticking rates over a given period. Here, one machine ticks 1000 times per millisecond (let's call it Machine A), and the other ticks 990 times per millisecond (Machine B). UTC updates occur once per minute (60 seconds).

**Determine the rate of ticking for each machine**:

· Machine A: 1000 ticks per millisecond

Machine B: 990 ticks per millisecond

· Calculate how many milliseconds each machine would tick in one minute:

· 1 minute = 60 seconds = 60,000 milliseconds

**For Machine A**:

Ticks by Machine A in 1 minute=1000 ticks/ms×60,000 ms=60,000,000 ticks

**For Machine B**:

Ticks by Machine B in 1 minute=990 ticks/ms×60,000 ms=59,400,000 ticks

Calculate the difference in ticks between the two machines in one minute:

Difference in ticks=60,000,000 ticks−59,400,000 ticks=600,000 ticks

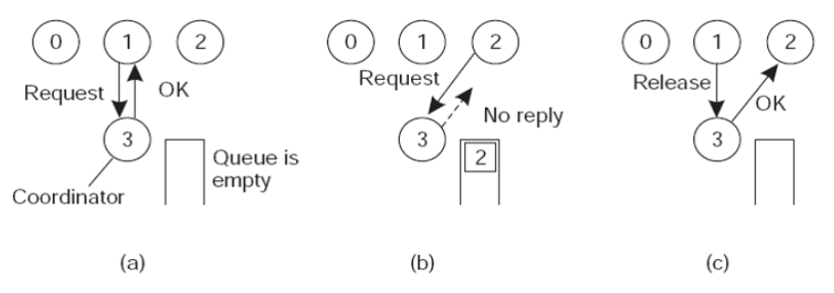
**Determine the maximum clock skew in milliseconds**:

To find the maximum skew, convert the difference in ticks to milliseconds using the rate of Machine B (as Machine B ticks slower and represents the slower rate that would lead to the maximum difference over time):

Each millisecond for Machine B corresponds to 990 ticks. Thus:

Milliseconds corresponding to the difference=990 ticks/ms600,000 ticks≈606.06 m

**Q:** Consider the following figure of a centralized algorithm to implement mutual exclusion in distributed systems:



Suppose that the coordinator crashes. Does this always brings the system down? If not, under what circumstances does this happen? Is there any way to avoid the problem and make the system able to tolerate coordinator crashes?

**Ans**:

In a centralized algorithm for mutual exclusion in distributed systems, the coordinator is a single point of authority that controls access to the critical section. If the coordinator crashes, the ability of the system to maintain mutual exclusion and continue functioning depends on several factors:

**Does the Crash Always Bring the System Down?**

**Single Coordinator Crash Impact:**

**Yes, if**: The system's design does not include mechanisms to handle the coordinator's failure, then the system might be unable to grant access to the critical section, leading to a halt in the processes that need mutual exclusion.

**No, if**: The system is designed with fault tolerance in mind. There are strategies to handle such failures, making the system resilient to the crash of the coordinator.

**Circumstances Affecting the Impact**:

**Coordinator Crash without Recovery Mechanism**:

If the coordinator crashes and there is no recovery mechanism or backup coordinator, the system cannot proceed with mutual exclusion since no process can take over the coordinator's responsibilities.

**Coordinator Crash with Recovery Mechanism**:

If the system includes mechanisms for handling the crash of the coordinator, such as an **election algorithm** to select a new coordinator or having multiple coordinators, then the impact of the coordinator's crash can be mitigated.

**Avoiding the Problem and Tolerating Coordinator Crashes:**

**Backup Coordinators**:

Implement backup coordinators that can take over if the primary coordinator crashes. This approach requires designing a fail-over mechanism where backup coordinators are in sync with the state of the system.

**Election Algorithm**:

Use a distributed election algorithm (e.g., Bully Algorithm, Ring Algorithm) to select a new coordinator when the current one fails. This approach requires all processes to participate in the election to choose a new coordinator.

**Replication and Redundancy**:

Replicate the coordinator's state across multiple nodes. When the primary coordinator fails, one of the replicas can take over with minimal disruption. This approach is more complex but provides resilience against coordinator failure.

**Heartbeat and Timeout Mechanisms**:

Implement a heartbeat mechanism where the coordinator periodically sends signals to the processes to indicate it's still alive. If a process does not receive a heartbeat within a certain timeout period, it can initiate a recovery procedure to handle the crash.

**Logging and State Recovery**:

Keep logs of the coordinator's state so that a new coordinator can recover the state from the logs and continue operations smoothly.