A major concern in any parallel and distributed system is communication in the presence of channel

failures. There are multiple modes for a channel to fail, and some lead to messages being lost. In the

general case, it is impossible to guarantee that two processes will reach an agreement in case of channel

failures (see Figure 2.4.)

Given two processes p1 and p2 connected by a communication channel that can lose a message with

probability  > 0, no protocol capable of guaranteeing that two processes will reach agreement exists,

regardless of how small the probability   is.

The proof of this statement is by contradiction. Assume that such a protocol exists and it consists of

n messages; recall that a protocol is a finite sequence of messages. Since anymessage might be lost with

probability  , the protocol should be able to function when only n −1 messages reach their destination,

the last one being lost. Induction on the number of messages proves that indeed no such protocol exists;

indeed, the same reasoning leads us to conclude that the protocol should function correctly with (n−2)

messages, and so on.

In practice, error detection and error correction allow processes to communicate reliably though

noisy digital channels. The redundancy of a message is increased by more bits and packaging a message

as a code word; the recipient of the message is then able to decide if the sequence of bits received is a

valid code word and, if the code satisfies some distance properties, then the recipient of the message is

able to extract the original message from a bit string in error.

Communication protocols implement not only error control mechanisms, but also flow control and

congestion control. Flow control provides feedback from the receiver; it forces the sender to transmit

only the amount of data the receiver is able to buffer and then process. Congestion control ensures that

the offered load of the network does not exceed the network capacity. In store-and-forward networks,

individual routersmay drop packetswhen the network is congested and the sender is forced to retransmit.

Based on the estimation of the round-trip-time (RTT), the sender can detect congestion and reduce the

transmission rate.

The implementation of thesemechanisms requires the measurement of time intervals, the time elapsed

between two events; we also need a global concept of time shared by all entities that cooperate with one

another. For example, a computer chip has an internal clock, and a predefined set of actions occurs at

each clock tick. Each chip has an interval timer that helps enhance the system’s fault tolerance; when

the effects of an action are not sensed after a predefined interval, the action is repeated.

When the entities communicating with each other are networked computers, the precision of the

clock synchronization is critical [205]. The event rates are very high and each system goes through

state changes at a very fast pace; modern processors run at a 2–4 GHz clock rate. That explains why we

need to measure time very accurately; indeed, we have atomic clocks with an accuracy of about 10−6

seconds per year.

An isolated system can be characterized by its history, expressed as a sequence of events, each event

corresponding to a change of the state of the system. Local timers provide relative timemeasurements.A

more accurate description adds to the system’s history the time of occurrence of each event as measured

by the local timer.

Messages sent by processes may be lost or distorted during transmission.Without additional restrictions

regarding message delays and errors, there are no means to ensure a perfect synchronization of

local clocks and there are no obvious methods to ensure a global ordering of events occurring in different

processes. Determining the global state of a large-scale distributed system is a very challenging

problem.

Themechanisms described here are insufficient onceweapproach the problem of cooperating entities.

To coordinate their actions, two entities need a common perception of time. Timers are not enough.

Clocks provide the only way to measure distributed duration, that is, actions that start in one process

and terminate in another. Global agreement on time is necessary to trigger actions that should occur

concurrently (e.g., in a real-time control system of a power plant, several circuits must be switched

on at the same time). Agreement on the time when events occur is necessary for distributed recording

of events – for example, to determine a precedence relation through a temporal ordering of events.

To ensure that a system functions correctly, we need to determine that the event causing a change of

state occurred before the state change – for instance, the sensor triggering an alarm has to change its

value before the emergency procedure to handle the event is activated. Another example of the need for

agreement on the time of occurrence of events is in replicated actions. In this case several replicas of a

process must log the time of an event in a consistent manner.

Time stamps are often used for event ordering using a global time base constructed on local virtual

clocks [235]. The  -protocols [94] achieve total temporal order using a global time base. Assume that

local virtual clock readings do not differ by more than π, called precision of the global time base. Call

g the granularity of physical clocks. First, observe that the granularity should not be smaller than the

precision; given two events a and b occurring in different processes, if tb − ta   π + g we cannot tell which one occurred first [361]. Based on these observations, it follows that the order discrimination of

clock-driven protocols cannot be better than twice the clock granularity.

System specification, design, and analysis require a clear understanding of cause-effect relationships.

During the system specification phase we view the system as a state machine and define the actions that

cause transitions from one state to another. During the system analysis phase we need to determine the

cause that brought the system to a certain state.

The activity of any process ismodeled as a sequence of events; hence, the binary relation cause-effect

relationship should be expressed in terms of events and should express our intuition that the cause must

precede the effects. Again, we need to distinguish between local events and communication events.

The latter events affect more than one process and are essential for constructing a global history of an

ensemble of processes. Let hi denote the local history of process pi and let ek

i denote the k-th event in

this history.

The binary cause-effect relationship between two events has the following properties:

1. Causality of local events can be derived from the process history:

if ek

i , el

i

∈ hi and k < l then ek

i

→ el

i . (2.20)

2. Causality of communication events:

if ek

i

= send(m) and el

j

= receive(m) then ek

i

→ el

j . (2.21)

3. Transitivity of the causal relationship:

if ek

i

→ el

j and el

j

→ enm

then ek

i

→ enm

. (2.22)

Two events in the global history may be unrelated. If so, neither one is the cause of the other; such

events are said to be concurrent events.