The communication channel abstraction makes no assumptions about the order of messages; a reallife

network might reorder messages. This fact has profound implications for a distributed application.

Consider for example a robot getting instructions to navigate from a monitoring facility with two

messages, “turn left” and “turn right,” being delivered out of order.

Message receiving and message delivery are two distinct operations; a delivery rule is an additional

assumption about the channel-process interface. This rule establishes when a message received is actually

delivered to the destination process. The receiving of a message m and its delivery are two distinct events in a causal relation with one another. A message can only be delivered after being received (see

Figure 2.6)

receive(m) → deliver(m). (2.25)

First In, First Out (FIFO) delivery implies that messages are delivered in the same order in which

they are sent. For each pair of source-destination processes (pi , p j ), FIFO delivery requires that the

following relation should be satisfied:

sendi (m) → sendi (m

) ⇒ deliver j (m) → deliver j (m

). (2.26)

Even if the communication channel does not guarantee FIFO delivery, FIFO delivery can be enforced

by attaching a sequence number to each message sent. The sequence numbers are also used to reassemble

messages out of individual packets.

Causal delivery is an extension of the FIFO delivery to the case when a process receives messages

from different sources. Assume a group of three processes, (pi , p j , pk ) and two messages m and m .

Causal delivery requires that

sendi (m) → send j (m

) ⇒ deliverk (m) → deliverk (m

). (2.27)

When more than two processes are involved in a message exchange, the message delivery may be

FIFO but not causal, as shown in Figure 2.7 where we see that

• deliver(m3) → deliver(m1), according to the local history of process p2.

• deliver(m2) → send(m3), according to the local history of process p1.

• send(m1) → send(m2), according to the local history of process p3. • send(m2) → deliver(m2).

• send(m3) → deliver(m3).

The preceding transitivity property and the causality relations imply that send(m1) → deliver(m3).

Call T S(m) the time stamp carried by message m. A message received by process pi is stable if no

future messages with a time stamp smaller than T S(m) can be received by process pi . When logical

clocks are used, a process pi can construct consistent observations of the system if it implements the

following delivery rule: Deliver all stable messages in increasing time-stamp order.

Let’s now examine the problem of consistent message delivery under several sets of assumptions.

First, assume that processes cooperating with each other in a distributed environment have access to a

global real-time clock, that the message delays are bounded by δ, and that there is no clock drift. Call

RC(e) the time of occurrence of event e. A process includes RC(e) in every message it sends, where e

is the send-message event. The delivery rule in this case is: At time t deliver all received messages with

time stamps up to (t −δ) in increasing time-stamp order. Indeed, this delivery rule guarantees that under

the bounded delay assumption the message delivery is consistent. All messages delivered at time t are

in order and no future message with a time stamp lower than any of the messages delivered may arrive.

For any two events, e and e , occurring in different processes, the so-called clock condition is satisfied

if

e → e  ⇒ RC(e) < RC(e

), ∀e, e

. (2.28)

Often, we are interested in determining the set of events that caused an event knowing the time

stamps associated with all events; in other words, we are interested in deducing the causal precedence

relation between events from their time stamps. To do so we need to define the so-called strong clock

condition. The strong clock condition requires an equivalence between the causal precedence and the

ordering of the time stamps

∀e, e

, e → e  ≡ T S(e) < T S(e

). (2.29)

Causal delivery is very important because it allows processes to reason about the entire system using

only local information. This is only true in a closed system where all communication channels are known; sometimes the system has hidden channels, and reasoning based on causal analysis may lead

to incorrect conclusions.