Exercise Session 3 - Solutions
Causal and Total Order Broadcast

Exercise 1

Can we devise a broadcast algorithm that does not ensure the causal delivery *property but only its nonuniform variant: no correct process p_i delivers a message m_2 unless p_i has already delivered every message m_1 such that $m_1 \rightarrow m_2$?*

The answer is *no*. Assume by contradiction that some algorithm does not ensure the *causal delivery* property but ensures its nonuniform variant. This means that the algorithm has some execution in which some process p delivers some message m without delivering a message m' that causally precedes m. Given that we assume a model where processes do not self-destruct, p might as well be correct, in which case it violates even the nonuniform variant.

Exercise 2

Suggest an optimization of the garbage collection scheme of Algorithm 1′ (slide 24).

When removing a message m from the past, we can also remove all the messages that causally precede this message — and then recursively those that causally precede these. This means that a message stored in the past must be stored with its own distinct past.

Exercise 3

Why the condition on slide 30 is $VC[pk] \geq VC_x[pk]$ and not just $VC[pk] = VC_x[pk]$? Can you construct an execution where the local vector clock is greater than the received local clock for one place?

The greater or equal condition is important in order to handle old, non-causal messages. Just consider the case when in the beginning all processes send a message. Each message will carry a vector clock with all zeroes. When a process delivers one such message, the vector clock of the process gets incremented at a position matching the rank of the sender. During the next iteration, the process needs to deliver a message containing all zeroes in its vector clock, while the process has a non-zero value in at least one position.

Exercise 4

Can we devise a best-effort broadcast algorithm that satisfies the causal delivery *property without being a causal broadcast algorithm, i.e., without satisfying the* agreement *property of a reliable broadcast?*

The answer is no. Assume by contradiction that some broadcast algorithm ensures the *causal delivery* property and is not reliable but best-effort; define an instance *co* of the corresponding abstraction, where processes *co*-broadcast and *co*-deliver messages.

The only possibility for an algorithm to ensure the properties of best-effort broad- cast but not those of reliable broadcast is to violate the *agreement* property: there must be some execution of the algorithm where some correct process p *co*-delivers a message m that some other process q does not ever *co*-deliver.

Because the algorithm is best-effort, this can only happen if the process s that *co*-broadcasts the message is faulty.

Assume now that after *co*-delivering *m*, process *p* co-broadcasts a message m'. Given that *p* is correct and that the broadcast is best-effort, all correct processes, including *q*, will *co*-deliver m'. Given that *m* precedes m' in causal order, *q* must have co-delivered *m* as well, a contradiction. Hence, any best-effort broadcast that satisfies the *causal delivery* property satisfies *agreement* and is, thus, also a reliable broadcast.

Exercise 5

The Uniform Reliable Broadcast Algorithm requires a process to receive an acknowledgment from all nonfaulty processes before it can deliver a message. The acknowledgment is needed because when a process invokes the underlying best-effort broadcast and then crashes, all components of the process are affected and stop (including the best- effort broadcast module and any further underlying modules, such as the modules that may implement perfect links). The unit of failure is a process, not a module.

For this exercise only, consider an idealized and nonrealistic system model, where some component may invoke infallible lower-level components. In this model, the unit of failure is not a process but a module. Describe an implementation of uniform reliable broadcast that uses an infallible perfect point-to-point links abstraction in this idealized model. Do not use failure detectors of any kind.

Hint: *You may get some inspiration from the solution to last week's exercise 2.*

Suppose an "IdealPerfectPointToPointLinks" module is available in this idealized system model. In the following algorithm, the sender sends the broadcast message to itself over the ideal perfect links; upon delivering a message *m* over the ideal perfect links that has not been delivered yet, it resends *m* to all processes and *urb*-delivers it.

```
Implements:
  UniformReliableBroadcast, instance urb.

Uses:
  IdealPerfectPointToPointLinks, instance idealpl.

upon event <urb, Init> do
  delivered := ∅;

upon event <urb, Broadcast | m> do
  trigger <idealpl, Send | self, (DATA, self, m)>;

upon event <idealpl, Deliver | p, (DATA, s, m)> do
  if $m \notin$ delivered then
    delivered := delivered ∪ (m);
    forall $q \in \Pi$
      trigger <idealpl, Send | q, (DATA, s, m)>;
    trigger <urb, Deliver | s, m>;
```

The *uniform agreement* property holds because every process sends *m* with the infallible point-to-point links primitive before it *urb*-delivers *m*. The infallible underlying module does not crash in this idealized model. If a process crashes, only the broadcast module crashes. Any *urb*-delivered message will never be forgotten by the ideal link module and will consequently be *urb*-delivered by every correct process.

Exercise 6

Would it make sense to add the total-order *property to the best-effort broadcast?*

The resulting abstraction would not make much sense in a failure-prone environment, as it would not preclude the following scenario. Assume that a process *p* broadcasts several messages with *best-effort* properties and then crashes.

Some correct processes might end up delivering all those messages (in the same order) whereas other correct processes might end up not delivering any message.

Exercise 7

What happens in our consensus-based total order broadcast algorithm if the set of messages decided on is not sorted deterministically

a) *after the decision but is sorted prior to the proposal,*

If the deterministic sorting is done prior to proposing the set for consensus, instead of *a posteriori* upon deciding, the processes would not agree on a set but on a sequence of messages. But if they *to-deliver* the messages in decided order, the algorithm still ensures the total order property.

b) *neither a priori nor a posteriori?*

If the messages, on which the algorithm agrees in consensus, are never sorted deterministically within every batch (neither *a priori* nor *a posteriori*), then the total order property does not hold. Even if the processes decide on the same batch of messages, they might *to-deliver* the messages within this batch in a different order. In fact, the total order property would be ensured only with respect to batches of messages, but not with respect to individual messages. We thus get a coarser granularity in the total order.