Exercise Session 3
Total Order Broadcast

October 17, 2010

Exercise 1

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 2

What happens in our consensus-based total order broadcast algorithm if the set of messages decided on are not sorted deterministically after the decision but prior to the proposal? What happens in our consensus-based total order broadcast algorithm if the set of messages decided on is not sorted deterministically, neither a priori nor a posteriori?

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.

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.

We could avoid using the deterministic sort function at the cost of proposing a single message at a time in the consensus abstraction. This means that we would need exactly as many consensus instances as there are messages exchanged between the processes. If messages are generated very slowly by processes, the algorithm ends up using one consensus instance per message anyway. If the messages are generated rapidly, then it is beneficial to use several messages per instance: within one instance of consensus, several messages would be gathered, i.e., every message of the consensus algorithm would concern several messages to *to-deliver*. Agreeing on large batches with many messages at once is important for performance in practice, because it considerably reduces the number of times that the consensus algorithm is invoked.