As noted at the start of Chapter 16, researchers have carried the idea of group communication to something of an extreme. This chapter tackles some of the more esoteric topics that arise when building a group-communication platform that permits the creation of large numbers of groups without requiring (as in the Spread system) that they all be subgroups of a single encompassing group. The questions raised by such a model are quite interesting, but aren’t necessarily very important, since existing group communication platforms such as Ensemble and Spread focus on supporting a small number of groups, although permitting the creation of subgroups within them. Yet if group communication begins to be applied in Web Services settings, the very nature of those platforms may force them to support much larger numbers of groups and preclude the assumption that larger enclosing groups will generally be present. Thus the material that follows is of limited importance in existing systems but may become more significant down the road.

Multi-group communication protocols have a way of becoming both complex and costly, and this should send up warning flags for developers seeking elegant, high-performance solutions to reliability in large systems. Certainly, if a system will support large numbers of groups, any special properties it offers to the user who works with multiple groups should be “selectable”: capable of being enabled or disabled quite easily under program control. Moreover, the defaults should generally be set to achieve the lowest possible costs, much as we have opted for non-uniform failure behavior rather than safe, dynamically uniform behavior, despite the relative conceptual simplicity of the latter: at the end of the day, if a property is very costly, it shouldn’t be the default.
17.1 Causal Communication Outside of a Process Group

Although there are sophisticated protocols in guaranteeing that causality will be respected for arbitrary communication patterns, the most practical solutions generally confine concurrency and associated causality issues to the interior of a process group—for example, at the end of Section 16.3, we briefly cited the replication protocol of Ladin and Liskov (see Ladin et al. [1992], Liskov et al.). This protocol transmits a timestamp to the client, and the client later includes the most recent of the timestamps it has received in any requests it issues to the group. The group members can detect causal ordering violations and delay such a request until causally prior multicasts have reached their destinations, as seen in Figure 17.1.

An alternative is to simply delay messages sent out of a group until any causally prior multicasts sent within the group have become stable—in other words, have reached their destinations. Since there is no remaining causal ordering obligation in this case, the message need not carry causality information. Moreover, such an approach may not be as costly as it sounds, for the same reason that the flush protocol introduced earlier turns out not to be terribly costly in practice: Most asynchronous cbcast or fbcast messages become stable shortly after they are issued—long before any reply is sent to the client. Thus, any latency is associated with the very last multicasts to have been initiated within the group, and will normally be small. We will see a similar phenomenon (in more detail) in Section 19.5, which discusses a replication protocol for stream protocols.