Abstract: A View Synchronous Communication (VSC) service provides applications in a multicast group with 1) reliable message delivery and 2) information about changes in the multicast group membership, in the form of membership views. To elaborate, a VSC service guarantees that the applications that receive the same membership view change receive the same set of group messages before receiving that view change.

In this paper, we show how to use a generic static total and causal ordering multicast protocol to design a VSC service that guarantees totally and causally ordered message delivery without the need for blocking message delivery during view change operations. A special feature of the protocol is the ability to merge multiple groups of processes simultaneously.

INTRODUCTION

In this paper, we present a view synchronous totally ordered message delivery protocol for a dynamic asynchronous process group in an asynchronous communication environment. The protocol can handle asynchronous process or link failures and also the simultaneous joining of multiple groups of processes. Therefore it is capable of handling network partitions and remerges. The protocol is efficient for use in replicated services that do not require global
consistency but instead must provide high availability in the face of network partitions. Our protocol is closely related to the membership protocol of Transis (Amir et al., 1992) since it uses the causal order to establish agreement among connected processes.

Many distributed systems involve a set of applications that have to coordinate their actions only by passing messages to each other. For many of these systems, it is necessary to ensure that all the applications agree upon a common order for the set of messages transacted between them. This is because the order in which the applications accept messages may affect their future actions. For example, in a distributed database application, an application may make updates to a local database, based on the messages delivered to it. If all the databases in the system are to store information consistently, then all the applications must agree upon a common order for the messages. Furthermore, in many situations (like a replicated database), it is also important that if a message is delivered to any single application in the system, it is also eventually delivered to every other application in the system. This is the atomic (all or none) delivery requirement.

**Total ordering** protocols in a group of processes deliver the same set of messages at all processes in the group in the same order.

**Causal ordering** message delivery protocols stand roughly for those protocols that ensure that a response to a message is never seen before the message itself. Causal ordering is formally defined in Section 2.

In this paper, we present a protocol that ensures both total and causal ordering in dynamic asynchronous process groups. The protocol can be implemented by running a set of identical **protocol processes** on **suitable** hosts in the communication network. By suitable hosts, we mean that each application using the protocol should have a quick access to a protocol process. For example, if the system consists of several high speed reliable local area networks (LANs) each having multiple applications and connected by an unreliable wide area network (WAN), it would be suitable to have a dedicated server running a protocol process on a host in each LAN. The user applications could then log on to the local protocol server as clients and register as users for one or more multicast groups when needed. For each of the multicast groups to which belongs, a protocol server would deliver over the LAN all the group messages to each of the group's users, according to the total order.

We consider the total ordering problem as among the protocol **processes** (also simply called **processes**, in short) and not among the applications which are considered as users making use of the protocol service. Fig. 2.1 shows a model of such a distributed system. For simplicity, the clients/users of a single protocol process are lumped together into a single application site.

Total ordering is closely related to the problem of consensus. Informally, in the consensus problem, each process holds some initial input value and the goal is for the nonfaulty processes to agree on what value from the set of initial values to choose as the output. It is easy to prove that consensus can be achieved among a fixed group of processes using total ordered message delivery. This