Brief Announcement: Concurrent Maintenance of Rings*

Xiaozhou Li Jayadev Misra C. Greg Plaxton

Department of Computer Science
University of Texas at Austin
Austin, Texas 78712
{xli,misra,plaxton}@cs.utexas.edu

Categories and Subject Descriptors: E.1 [Data Structures]: Distributed Data Structures; C.2.2 [Computer-Communication Networks]: Network Protocols—*Protocol Verification*

General Terms: Algorithms, Design, Theory, Verification

Keywords: Ring, concurrency, network protocols

A central problem for structured peer-to-peer networks is topology maintenance, that is, how to properly update neighbor variables when membership changes (i.e., nodes join or leave the network, possibly concurrently). Depending on whether neighbor variables are immediately updated once membership changes occur, there are two general approaches to topology maintenance: the *passive* approach and the *active* approach. Existing work on the active approach has several shortcomings: the protocols cannot handle both joins and leaves actively; the protocols are complicated; the correctness proofs are operational and sketched at a high level. It is well known, however, that concurrent programs often contain subtle errors and operational reasoning is unreliable for proving their correctness.

In this work, we address the maintenance of the ring topology, the basis of several peer-to-peer networks, in the fault-free environment. We design, and prove the correctness of, protocols that actively maintain a bidirectional ring under both joins and leaves. Using an assertional proof method, we prove the correctness of a protocol by developing a global invariant and showing that every action of the protocol preserves the invariant. We show that, although the ring topology may be tentatively disrupted during membership changes, the protocols restore the ring topology once membership changes subside or all the messages associated with membership changes are delivered. The protocols are based on an asynchronous communication model where only reliable delivery is assumed.

To illustrate our approach, we show in Figure 1 a join protocol for a unidirectional ring. The protocol is written as a collection of actions. We assume without loss of generality that the actions are atomic, and we reason about the system state between actions. In the protocol, the $contact()$ function returns an arbitrary non-out process if there is one, and returns the calling process otherwise. Our global invariant identifies a secondary ring structure that is preserved by every action.

We then design a protocol that maintains a bidirectional ring under both joins and leaves. Our approach is to first design a join protocol and a leave protocol and then combine them. To facilitate the combining of the two protocols, the join protocol and the leave protocol are designed to be symmetric. In both protocols, completing a join or a leave operation takes only four messages.

```
process p
 var   s : {in, out, jng}; r : V'; a : V'
 init  s = out ∧ r = nil
begin
  □ s = out →  a := contact();
       if a = p → r, s := p, in
       □ a ≠ p → s := jng; send join() to a fi
  □ rcv join() from q →
       if s = in → send grant(r) to q; r := q
       □ s ≠ in → send retry() to q fi
  □ rcv grant(a) from q →  r, s := a, in
  □ rcv retry() from q →  s := out
end
```

Figure 1: The join protocol for a unidirectional ring.

The combined protocol, however, is not a straightforward combination of the join protocol and the leave protocol; some subtleties arise when both joins and leaves are considered. We again use an assertional method to prove the correctness of these protocols; the invariants for these protocols are more involved. In these protocols, a process enters a $busy$ state when it is processing a join or a leave request; a $busy$ process declines any other join or leave requests. However, if we assume reliable and ordered delivery of messages, then there is a join protocol that does not require a $busy$ state.

A desirable property of a topology maintenance protocol is that out processes do not receive messages. We show that, assuming reliable and ordered delivery, the leave protocol provides this property, and with some simple extensions, the combined protocol also provides this property.

In certain extreme scenarios (e.g., if all of the processes in the network try to leave at once), the leave protocol and the combined protocol may suffer from livelock. Due to the similarity between this problem and the classical dining philosophers problem, it may be difficult to obtain a livelock-free deterministic symmetric protocol. In practice, a system can use other techniques to avoid livelock. For example, as in the Ethernet protocol, a process may delay a random amount of time before sending out another leave request.

It would be interesting to extend the techniques and results of this paper to establish stronger progress properties, to maintain more sophisticated peer-to-peer topologies, and to allow for faults. In addition, it would be worthwhile to develop machine-checked proofs for the protocols using a theorem prover such as ACL2.

*Full paper available as TR-04-03, Department of Computer Science, University of Texas at Austin, February 2004. This research was supported by NSF grants CCR–0310970 (Li and Plaxton) and CCR–0204323 (Misra).

PODC'04, July 25–28, 2004, St. Johns, Newfoundland, Canada.
ACM 1-58113-802-4/04/0007.