Specification and analysis of a data transfer protocol using systems of communicating machines

Gilbert M. Lundy, Jr and Raymond E. Miller

1 Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943, USA
2 Department of Computer Science, University of Maryland, College Park, MD 20742, USA
3 NASA Center of Excellence in Space Data and Information Sciences, Goddard Space Flight Center, Greenbelt, MD 20771, USA

Received December 1988 / Accepted July 1991

Summary. A model for communication protocols called systems of communicating machines is used to specify a data transfer protocol with variable window size (e.g., HDLC), which is an arbitrary nonnegative integer, and to analyze it for freedom from deadlocks. The model uses a combination of finite state machines and variables. This allows the size of the specification (i.e., number of states and variables) to be linear in the window size, a considerable reduction from the pure finite state machine model. A new type of analysis is demonstrated which we call system state analysis. This is similar to the reachability analysis used in the pure finite state model, but it provides substantial simplification by reducing the number of states generated. For example, with the protocol in this paper, if $w$ is the window size, then the global analysis produces $O(w^2)$ states, while the system state analysis produces $O(w^3)$ states. The system state analysis is then combined with an inductive proof, extending the analysis to all nonnegative integers $w$.

Key words: Communication protocol - Formal model - Specification - Analysis - Formal description technique

1 Introduction

The problem of the specification and analysis of communication protocols has been the subject of much research in recent years. The inherent complexity and variety in
These protocols make this problem a difficult one. Yet protocol correctness is critical to reliable network operation, and standardization and interoperability of protocols is important for ease of network usage. For these reasons the formal modeling and analysis of protocols is an important area of study. In [37], the importance of formal modeling of protocols is discussed. Reference [41] has a comparison of the various techniques for specifying protocols, and [19] discusses the evaluation of formal description techniques. Most of the methods used in modeling protocols can be put into one of the following general classifications: communicating finite state machines, Petri nets, programming languages, and hybrids. A considerable amount of work has been done in each of these areas, and each seems to have advantages and disadvantages both generally and with respect to particular protocols.

### 1.1 Communicating finite state machines

In the communicating finite state machine (CFSM) model, each process is modeled as a finite state machine, and implicit queues between the machines are used for communication. A global state of the network is a tuple containing the state of each machine and the contents of each queue in the system. The most common method of analysis used with this model is called reachability analysis. In this method, all possible global states are generated from the initial global state by taking all possible transitions out of each machine. It is well known that if the implicit queues are allowed to have unbounded length, then it is undecidable whether the analysis will terminate, but if an upper bound is placed on the queue length, the method will eventually terminate (see [42]). This model has the advantage of simplicity and a method of analysis that can be easily automated. The obvious disadvantage is that the analysis might not terminate if the queue length is unbounded. A second disadvantage, related to but distinct from the first, is that the number of global states in the reachability analysis is often, for nontrivial protocols, so large as to make the analysis impractical, even if the queue length is bounded, and even if the analysis is automated. This problem is discussed in [22], which contains the automated analysis of a small protocol using this model.

One might argue that the first disadvantage, though of theoretical interest, is not really a problem for practical protocols, because real queues are always of finite length, so we can analyze the protocol of interest using the maximum allowable queue length. This argument has some merit, though the physical bound on queue length is often arbitrary. However, it is our view that the second disadvantage listed is the real problem for this model. The number of global states in the complex protocols used in computer networks grows so quickly that, while a computer analysis might eventually terminate, “eventually” might mean after days or even months of CPU time – clearly impractical.

A third disadvantage is that, with no memory other than the use of states, the specification of a practical protocol can be so complex, containing hundreds of states and transitions, that one can never really be sure it is the intended specification, or grasp an intuitive feeling for what the protocol is intended to do. This problem also manifested itself in the work leading up to [22]. The author was forced to break a 2 machine system into a 6 machine system, because of the difficulty in specifying the protocol with this model.

Much of the work which has been done using this model is an attempt to lessen the effect of these disadvantages, or to get around them. Choi and Miller [8] have worked on simplifying protocols through decomposition. Gouda has done a considerable amount of work in this area. For example, in [11], with Yu, the second problem listed above is addressed. A bounded queue length is assumed, and a method of reachability analysis is used which generates a smaller number of global states than the usual method. Vuong and Cowan also have obtained some results with this model, which address the first problem (decidability). In [42], a class of protocols is defined for which some of the properties are decidable.

### 1.2 Other models for protocols and parallel programs

Programming languages have the advantage of being more powerful than pure finite state machine models, at the expense of added complexity. The complexity generally makes analysis more difficult. Several papers have appeared in the literature using programming language models, and combinations of these with FSMs, for protocol description. Some of the languages used are CSP, LOTOS, and Ada. CSP (Communicating Sequential Processes) [12, 34] is a high level language with facilities for describing concurrent processes. Several papers on LOTOS have appeared; [4] is a tutorial. Ada [6] contains tasking for parallel or concurrently executing processes, with the rendezvous mechanism for communication between processes. In [40], a data transfer protocol is specified using the language Pascal. That protocol is a host-host, or transport layer protocol. It is verified by proving assertions stating certain desirable properties of the protocol.

There have also been several models which combined programming languages with the finite state model. Estelle is one such approach [5, 9, 20]. In [3], Bochmann and Gecsei describe an alternating bit protocol using a model combining finite state machines and variables. In [39], Shankar verifies a data transfer protocol with variable flow control, using a set of state variables to specify the processes. The protocol allows the channels to make various errors, and history variables are defined which are used in the verification. There are several differences between these and our approach. For instance, in Estelle, the machines are specified using an extended form of Pascal.

In [1, 2] a model is presented which had goals very similar to our goals in this work. Protocols were described using a model called “selection/resolution”, and a software package was built which serves as a tool in helping to evaluate the protocol. That model was also