Reliable Communication Over Unreliable Channels

YEHUDA AFEK

Tel-Aviv University, Tel-Aviv, Israel

HAGIT ATTIYA

Technion, Haifa, Israel

ALAN FEKETE

University of Sydney, Sydney, Australia

MICHAEL FISCHER

Yale University, New Haven, Connecticut

NANCY LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

YISHAY MANSOUR

Tel-Aviv University, Tel-Aviv, Israel

DAI-WEI WANG

Academia Sinica, Nan Kang, Taipei, Taiwan

AND

LENORE ZUCK

Yale University, New Haven, Connecticut

This research was supported in part by Research Foundation for Information Technology, University of Sydney; Department of Computer Science, Tel-Aviv University; IBM Fellowship; Office of Naval Research (ONR) contracts N00014-85-K-0168, N00014-91-J-1046, N00014-85-K-0445, and N00014-82-K-0154; by National Science Foundation (NSF) grants CCR 86-11442, CCR 89-15206, IRI 90-15570, DCR 84-05478, and CCR 89-10289; and by DARPA contracts N00014-89-J-1988, N00014-87-K-0825, and N00014-92-J-4033.

Authors' addresses: Y. Afek, Computer Science Department, Tel-Aviv University, Tel-Aviv, Israel; H. Attiya, Department of Computer Science, Technion, Haifa 32000, Israel; A. Fekete, Department of Computer Science F09, University of Sydney 2006, Australia; M. Fischer, Department of Computer Science, Yale University, P.O. Box 208285, New Haven, CT 06520-8285; N. Lynch, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139; Y. Mansour, Department of Computer Science, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel; D.-W. Wang, Institute of Information Science, Academia Sinica, Nan Kang, Tapei, Taiwan; L. Zuck, Department of Computer Science, Yale University, P.O. Box 208285, New Haven, CT 06520-8285.

Abstract. Layered communication protocols frequently implement a FIFO message facility on top of an unreliable non-FIFO service such as that provided by a packet-switching network. This paper investigates the possibility of implementing a reliable message layer on top of an underlying layer that can lose packets and deliver them out of order, with the additional restriction that the implementation uses only a fixed finite number of different packets. A new formalism is presented to specify communication layers and their properties, the notion of their implementation by I/O automata, and the properties of such implementations. An I/O automaton that implements a reliable layer over an unreliable layer is presented. In this implementation, the number of packets needed to deliver each succeeding message increases permanently as additional packet-loss and reordering faults occur. A proof is given that no protocol can avoid such performance degradation.

Categories and Subject Descriptors: B.4.4 [**Input/Output and Data Communications**]: Performance Analysis and Design Aids—*formal models, verification, worst-case analysis*; C.2.0 [**Computer-Communication Networks**]: General—*data communications, Open System Interconnection reference model (OSI)*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*protocol architecture, protocol verification*; D.4.4 [**Operating Systems**]: Communications Management—*message sending, network communication*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism and concurrency*

General Terms: Algorithms, Design, Reliability, Theory, Verification

Additional Key Words and Phrases: Bounded packet header, datalink layer, fault recovery, FIFO layer, layer implementation, layered communication protocol, message reordering, packet-switching network, sequence transmission problem, transport protocol

1. *Introduction*

In order to overcome the great engineering complexity involved, designers typically organize a communication network as a series of layers. Each layer is viewed as a "black box" that can be used by the next higher layer. Typical higher layers provide communication services with "nicer" properties than the lower layers upon which they are implemented. The OSI reference architecture of the International Standards Organization is a well-known example of layered design. (See Bochmann and Gecsei [1977]; Tannenbaum [1989], and Zimmermann [1980] for more details.)

One of the most important functions of a higher-level interprocess communication layer is to mask faults exhibited by a less reliable lower layer. A higher reliable layer, which we call a *First In First Out* (FIFO) *layer*, must deliver messages correctly, exactly once, and in the intended order, whereas the lower layer upon which it is implemented might lack one or more of these desirable properties. Individual messages might be lost, duplicated, or corrupted, and sequences of messages might be delivered out of order.

This paper studies the general problem of implementing a higher reliable layer on a lower less-reliable layer. We call this the *reliable message transmission problem* (*RMTP*). Layers of the sort we consider arise, for example, at different places in the OSI architecture mentioned above. A reliable transport layer is often implemented using a connectionless network that permits message reordering and message loss. A reliable data link layer is usually implemented on top of a physical transmission medium that permits message loss and message corruption faults. To avoid confusion when discussing the implementation of one layer on another, we often use *packet* to denote messages of the lower layer, reserving the term "message" for the upper layer. We also sometimes refer to the lower layer as a "channel".

Solutions to RMTP for certain kinds of channels date back to the early work on communication protocols (cf. Aho et al. [1982], Bartlett et al. [1969], Stenning [1976]). Much of the early theoretical work was concerned with optimizing the number of states or number of packets under various assumptions about the channel. For example, Aho et al. [1982] consider RMTP using *synchronous* channels in which the loss of a packet can be detected by the recipient at the next time step.

Three kinds of faults are of interest when discussing RMTP in asynchronous systems: loss, reordering, and duplication of packets. Stenning's protocol [1976] solves RMTP and tolerates all three fault types. However, it requires packets of unbounded length, since each packet contains a sequence number as well as a message. This is not desirable in practice. For channels that use bounded length packets, whether or not solutions to RMTP exist depends on which combination of faults are to be tolerated. There are easy solutions for any one of the three fault types in isolation. There is also a solution, the Alternating Bit protocol, for the case of both loss and duplication faults [Bartlett et al. 1969]. By way of contrast, no solution is possible for the case of both reordering and duplication faults [Wang and Zuck 1989] and consequently also for the case of all three fault types.

The remaining case—channels that use only bounded-length packets and are subject to both reordering and loss faults—is considered in this paper. These channels are rather difficult to deal with. For example, if the transmitting station sends the sequence 1011210001 of one-digit packets, the receiving station might get 0011 or 1100 or 0000111112 or even nothing at all. It is not clear how the receiving station can derive any useful information from what it receives.

We use the term *nonduplication* for a channel where reordering and loss faults can occur arbitrarily. This is a natural abstraction of the service provided by a connectionless network layer. If reordering can occur only to a limited extent (so a packet cannot be overtaken by another which was sent more than a fixed time later), then a simple solution is provided by using a variant of Stenning's protocol with sequence numbers kept as remainders to a fixed modulus. This is done in existing communication networks, but it places undesirable interdependencies between constants used in the implementations of different layers, since the modulus used in the reliable layer depends on the extent to which packets can be reordered by the unreliable layer. If the reordering is arbitrary, as in a nonduplicating channel, then no modulus is large enough for this strategy to work. Indeed, it has often been conjectured informally that RMTP cannot be solved by a nonduplicating channel.

In this paper, we both prove and disprove this conjecture. We avoid the apparent contradiction in this statement by paying careful attention to the formal definitions. We present a solution in a natural model in which only the correctness of the layer implementation is required. We then show that there are no "efficient" solutions. Intuitively, a solution is efficient if it has the ability to recover form channel faults and resume transferring messages at a fixed rate, regardless of past channel behavior.

The above discussion makes apparent that a precise formal model is necessary to discuss RMTP and its possible solutions. We use the term, RMTP, in the remainder of this paper to refer to the problem of finding a protocol that implements a FIFO layer on a nonduplicating layer; hence, we need formal definitions of protocols and communication layers, and the notion of a protocol implementing one layer on another.

A "reactive system" (in the sense of Harel and Pnueli [1985] and Manna and Pnueli [1992]) is a system that interacts with its environment. A reactive system generates a "behavior" consisting of the visible activity of the system. Communication layers, protocols, and I/O automata [Lynch and Tuttle 1987; 1989], are all examples of reactive systems since they naturally generate behaviors.

The behavior of a communication layer is the visible activity that takes place at the two sites that form its interface with the environment. This activity takes the form of sends and receives of messages, which we call "actions". Messages to be transported by the layer are inserted into the layer at one site and removed from the layer at the other site.

A "program" is an activity in which all actions take place at a single site. We model programs by I/O automata. A "protocol" is a pair of programs that run at distinct sites. A system consisting of a protocol on two sites connected by a (lower) communication layer generates a behavior that is determined by the individual behaviors of the system's programs and communication layer and is thus an instance of general parallel composition of reactive systems. The system is said to "implement" a higher communication layer if its behavior satisfies the requirements for the higher layer.

The definitions for reactive systems at a single site and their realizations as I/O automata are presented in Section 2. Communication layers are defined in Section 3. The notion of a protocol implementing one layer on another is presented in Section 4; it gives the basis for a modular decomposition of layer implementations. This modularity is expressed by two general compositionality results. The first expresses how a stack of layer implementations, each using the service provided by the layer below, can be composed to give an implementation of the highest layer on the lowest one. The other allows two noninteracting layers, running in parallel, to be viewed as a single layer. These definitions and results give a formal framework in which to discuss communication protocols that extends beyond the particular problem treated here.

Using these definitions and formalism, we exhibit (in Section 5) a modular solution to RMTP built from two parts. The first part uses the Alternating Bit protocol to implement a FIFO layer on an "order-preserving" layer, one that can lose and duplicate packets but not reorder them. The second part implements an order-preserving layer on a nonduplicating layer. These relatively simple parts are combined using the two compositionality results of Section 4 to yield an implementation of a FIFO layer on a nonduplicating layer. The modular structure allows for a simple proof of correctness.

Our solution to RMTP, however, is not "efficient" in a sense made precise in Section 6. In fact, we prove in Section 6 that no such efficient solution to RMTP exists. Thus, the originally conjectured impossibility of solving RMTP with nonduplicating channels turns out to be true after all when solutions are required to be efficient. The proof is quite short because it relies on general properties of layers and their implementations that are given in Sections 3 and 4.

Results related to ours appear in several other papers. A collection of general definitions and composition results about layered protocols in a model related to ours is given in Lam and Shankar [1990]. A preliminary version of Theorem 5.3.2 appears in Attiya et al. [1989]. A preliminary version of Theorem 6.2.4 appears in Lynch et al. [1988]. Subsequent papers consider other versions of RMTP and other definitions of efficiency. For example, Mansour and Schieber [1992] contains an impossibility result for efficient RMTP using a related but incomparable notion of efficiency, and it extends the result to channels where message loss is probabilistic rather than adversarial. A quantified version of Theorem 6.2.4 for a nonuniform model in which the transmitter knows the entire input sequence when the protocol begins, as well as a similar theorem for the case of channels that can reorder and duplicate packets, are shown in Wang and Zuck [1989]. The efficiency of RMTP is investigated in Tempero and Ladner [1990] relative to a new family of parameterized complexity measures that measure the speed of recovery from errors and the efficiency of message transmission in the absence of channel errors. Also, Fekete et al. [1993] contains an impossibility result for RMTP in the presence of crashes that lose information. Finally, Fekete and Lynch [1990] investigates the feasibility of solving RMTP with no headers at all.

2. *Formal Definitions*

2.1 SEQUENCES AND EVENTS. Let $\alpha = \alpha_0, \alpha_1, \ldots$ be an arbitrary finite or infinite sequence. An occurrence of an element in α is called an *event* of α. Formally, an *event* is a pair $\pi = (i, a)$, where $i \in \mathsf{N}$ and a is an arbitrary element. The event π is an *event of* α if α_i is defined and $a = \alpha_i$. Events are partially ordered by the "earlier-than" relation, where (i, a) is *earlier than* (i', a') if $i < i'$. The term, *a-event*, is used to denote an event whose second component is a. Let A be an arbitrary set. An event π is an *A-event* if it is an a-event for some $a \in A$.

A sequence α' is a *subsequence of* α if $\alpha' = \alpha_{k_0}, \alpha_{k_1}, \ldots$ for some $k_0 < k_1 < \cdots$. A sequence α' is a *finite prefix* of α, written $\alpha' \preccurlyeq \alpha$, if $\alpha' = \alpha_0, \alpha_1, \ldots, \alpha_l$ for some $l < |\alpha|$. A sequence α' is the *restriction* of α to A, written $\alpha | A$, if α' is the subsequence of α obtained by deleting all non-A-events in α. We extend restrictions to sets of sequences in the usual way.

The restriction $\alpha' = \alpha | A = \alpha_{k_0}, \alpha_{k_1}, \ldots$ induces a natural correspondence σ from events of α' into events of α defined by $\sigma(i, \alpha'_i) = (k_i, \alpha_{k_i})$ and called the *embedding of* α' *in* α. Obviously, σ maps a-events to a-events.

2.2. MULTISETS. A *multiset* (or *bag*) is a collection of elements with multiplicities. Formally, a multiset Q is a pair $(dom[Q], copies[Q])$, where $dom[Q]$ is a set and $copies[Q]$ is a function from $dom[Q]$ to $\mathsf{N} - \{0\}$. For every element $u \in dom[Q]$, $copies[Q](u)$ denotes the number of occurrences of u in Q. We define the size of Q to be $\sum_{u \in dom[Q]} copies[Q](u)$. Where convenient, we extend $copies[Q]$ to larger domains $U \supseteq dom[Q]$ by defining $copies[Q](u) = 0$ for $u \in U - dom[Q]$.

Familiar set operations can be extended to multisets. For two multisets Q and Q', we say that Q is a *submultiset* of Q', written $Q \sqsubseteq Q'$, if $dom[Q] \subseteq dom[Q']$ and $copies[Q](u) \leq copies[Q'](u)$ for every $u \in dom[Q]$. We also define $Q \sqsubset Q'$ to mean $Q \sqsubseteq Q'$ and $Q \neq Q'$. This implies that $copies[Q](u) \neq copies[Q'](u)$ for some $u \in dom[Q']$. If $Q \sqsubseteq Q'$, then we can define the *multiset difference*, $R = Q' - Q$, where $dom[R] = dom[Q']$ and $copies[R](u) = copies[Q'](u) - copies[Q](u)$.[1]

[1]Strictly speaking, $dom[R]$ should be reduced to include only those elements u for which $copies[R](u) > 0$

We also have need in Section 6 for a more complicated partial ordering among multisets. Let k be a positive integer. For a multiset Q, let Q^k be the *k-bounded* multiset defined by $dom[Q^k] = dom[Q]$, and $copies[Q^k](u) = \min(k, copies[Q](u))$ for every $u \in dom[Q]$. Thus, Q^k has at most k copies of any element. For multisets Q_1 and Q_2, define $Q_1 <_k Q_2$ if $Q_1^k \sqsubset Q_2^k$. Note that $<_k$ is a strict partial order, that is, it is irreflexive, antisymmetric and transitive.

The ordering $<_k$ has an important finite chain property.

LEMMA 2.2.1. *Let $\mathscr{C} = Q_1 <_k Q_2 <_k \cdots$ be a possibly infinite increasing chain of multisets, and let $U = \bigcup_i dom[Q_i]$. If U is finite, then $\mathscr{C}$ has at most $k|U| + 1$ elements.*

PROOF. Define a measure $f(Q_i) = \Sigma_{u \in U} \min(k, copies[Q_i](u))$. It is easily shown that $f(Q_{i+1}) \geq f(Q_i) + 1$ and $f(Q_i) \leq k|U|$ for each i. Since also $f(Q_1) \geq 0$, the result follows. □

2.3. REACTIVE SYSTEMS AND BEHAVIORS. We use the term, *reactive system*, to describe computational entities which exhibit an ongoing activity, interacting with their environment and possibly not terminating (cf. Harel and Pnueli [1985]). The communication layers and protocols that we discuss in this paper are examples of reactive systems. Intuitively, a reactive system is a black box that, from time to time, performs externally visible atomic activities called "actions". An observer may record the history of a run by writing down the sequence of visible actions as they occur. Obviously, after the system performs a finite number of steps, the observed sequence is finite. We call it a "partial trace". A "trace" is the sequence observed when the system is allowed to run forever. Traces can be finite or infinite; every finite trace is a partial trace, but partial traces are not necessarily (finite) traces.

For many purposes, how the traces are developed is of no interest; all that matters is the set of possible traces. We call the description of a system's possible traces the "behavior" of the system, and we often identify a system with its behavior. Thus, our behaviors are based on trace semantics (cf. Hoare [1985], Kahn [1974], and Milner [1980]).

Formally, a *behavior* S is a pair $(acts(S), traces(S))$, where $acts(S)$ is a set of actions and $traces(S)$, the *traces of* S, is a set of finite and infinite sequences over $acts(S)$. Each element of $traces(S)$ is called an *S-trace*. We call α a *partial S-trace* if α is a finite prefix of some S-trace. In general, if the possible traces of a reactive system R are described by a behavior S, then we write $S = beh(R)$.

A reactive system R with behavior S is often a component of a larger system. A trace α of the larger system will in general contain symbols over a superset of $acts(S)$. Symbols in $acts(S)$ describe activity involving the R-component, and symbols not in $acts(S)$ describe the activity of other parts of the system. By restricting α to the symbols in $acts(S)$, we obtain a sequence describing the activity of R within the context of the larger system. We say that a sequence α is *S-consistent* if $\alpha | acts(S)$ is an S-trace. Thus, if α is S-consistent, then the S-activity it describes is allowable according to the definition of S. We say that α is *partial S-consistent* if α is finite and $\alpha | acts(S)$ is a partial S-trace. Equivalently, α is partial S-consistent iff it is a finite prefix of some S-consistent sequence. Note that the above definition of S-consistency extends naturally to arbitrary sequences since the assumption that α describes the activity of a "larger system" plays no formal role.

We often write S as shorthand for $acts(S)$; thus, $\alpha|S$ denotes $\alpha|acts(S)$. Similarly, we say that S and S' are *disjoint* if $acts(S) \cap acts(S') = \emptyset$.

Let S_A and S_B be behaviors. We say that S_B *refines* S_A and write $S_B \lhd S_A$ if $acts(S_B) \supseteq acts(S_A)$, and every S_B-trace is S_A-consistent. Intuitively, S_B is more refined than S_A in the sense that it requires all of S_A's actions and possibly more, and the restriction of any trace it permits to S_A's actions must also be permitted by S_A.

The following is immediate from the definitions.

LEMMA 2.3.1. *Refinement of behaviors is a transitive relation.*

The *parallel composition* of behaviors S_1 and S_2 is the behavior $S = S_1 \| S_2$ such that $acts(S) = acts(S_1) \cup acts(S_2)$ and $traces(S)$ consists of all sequences over $acts(S)$ that are both S_1- and S_2-consistent. Intuitively, the behavior S describes the result of running S_1 and S_2 in parallel, where S_1 and S_2 interact through coordinating on mutual actions. For a behavior S and disjoint behaviors S_1 and S_2, such that $S = S_1 \| S_2$, we say that S is *decomposable into* S_1 *and* S_2.

The following lemma, which is immediate from the definitions, shows that parallel composition can be extended naturally to sequences that include elements outside of the composed behavior.

LEMMA 2.3.2. *Let S_1 and S_2 be behaviors and α be an arbitrary sequence. Then*

$$\alpha \textit{ is } (S_1 \| S_2)\textit{-consistent iff } \alpha \textit{ is both } S_1\textit{- and } S_2\textit{-consistent.}$$

An analog to Lemma 2.3.2 holds for partial behavior-consistent sequences providing the two behaviors are disjoint.

LEMMA 2.3.3. *Let S_1 and S_2 be disjoint behaviors and α an arbitrary finite sequence. Then*

$$\alpha \textit{ is partial } (S_1 \| S_2)\textit{-consistent iff}$$
$$\alpha \textit{ is both partial } S_1\textit{-consistent and partial } S_2\textit{-consistent.}$$

PROOF. In one direction, assume that $\alpha \preccurlyeq \gamma$ for some $(S_1 \| S_2)$-consistent sequence γ. By Lemma 2.3.2, γ is S_1- and S_2-consistent and the implication follows:

In the other direction, assume that $\alpha \preccurlyeq \gamma_1$ and $\alpha \preccurlyeq \gamma_2$, where γ_1 is S_1-consistent and γ_2 is S_2-consistent. Let $\gamma_1 = \alpha\beta_1$, and $\gamma_2 = \alpha\beta_2$. Since S_1 and S_2 are disjoint, there exists a sequence β' such that $\beta'|S_1 = \beta_1|S_1$ and $\beta'|S_2 = \beta_2|S_2$. Let $\gamma = \alpha\beta'$. Clearly, $\gamma|S_1 = \gamma_1|S_1$ and $\gamma|S_2 = \gamma_2|S_2$; hence, γ is both S_1- and S_2-consistent. By Lemma 2.3.2, γ is $(S_1 \| S_2)$-consistent. The implication now follows since $\alpha \preccurlyeq \gamma$. □

The following lemma is immediate from the definitions. It shows that parallel composition of behaviors is associative and commutative.

LEMMA 2.3.4. *For every behavior S_1, S_2, and S_3, $S_1 \| (S_2 \| S_3) = (S_1 \| S_2) \| S_3$ and $S_1 \| S_2 = S_2 \| S_1$.*

The following lemma captures the interaction between composition and refinement.

LEMMA 2.3.5. *Let S, S_1, and S_2 be behaviors. If $S_1 \lhd S_2$, then $(S \| S_1) \lhd (S \| S_2)$.*

PROOF. Since $S_1 \lhd S_2$, we have $acts(S_1) \supseteq acts(S_2)$. Consequently, $acts(S \| S_1) \supseteq acts(S \| S_2)$. It remains to show that every $(S \| S_1)$-trace is $(S \| S_2)$-consistent. Let β be a $(S \| S_1)$-trace. Then $\beta | S \in traces(S)$ and $\beta | S_1 \in traces(S_1)$. Since $S_1 \lhd S_2$, it follows that $\beta | S_2 = (\beta | S_1) | S_2 \in traces(S_2)$. It follows from Lemma 2.3.2 that β is $(S \| S_2)$-consistent. □

2.4. I/O AUTOMATA. Although the behavior of a system describes what the system should do, it does not describe how it does it. We use a variant of the I/O automaton model [Lynch and Tuttle 1987; 1989] as a state-machine model of a reactive system.

An I/O automaton is a state machine with state transitions labeled by actions, classified as *input actions*, *output actions*, and *internal actions*. Intuitively, input and output actions are externally visible, and internal actions are hidden. Input actions are assumed to originate in the environment and always cause the automaton to take a step. Output and internal actions result from autonomous steps of the automaton. The output actions are presented to the environment, where they have the potential to affect other components.

Formally, an *I/O automaton A*, or simply an *automaton*, is described by:

(1) Three pairwise disjoint sets, $in(A)$, $out(A)$, and $internal(A)$ which denote the sets of input, output, and internal actions, respectively. Their union, $acts(A)$, is the set of *actions of A*. The subset $ext(A) = in(A) \cup out(A)$ is the set of *externally visible actions of A*. The *local actions of A* are the actions that are within A's control, namely, its internal and output actions.
(2) A set $states(A)$ of A's *states* and a set $start(A) \subseteq states(A)$ of A's *start states*.
(3) A set $steps(A) \subseteq states(A) \times acts(A) \times states(A)$ of *allowed steps*. We say that an action a is *enabled* from a state s if for some s', $(s, a, s') \in steps(A)$. We require that A be *input enabled*, that is, every input action is enabled from every state.
(4) A *fairness partition*, $fair(A)$, on A's local actions that has countably many equivalence classes. As explained below, A's fair executions are those that are weakly fair with respect to each class in $fair(A)$ [cf. Francez 1986]). Fairness is an attempt to restrict a system's behavior to be "realistic". Each class of $fair(A)$ typically consists of the actions controlled by a single component, so fairness means giving each component repeated opportunities to take a step.

An *execution* is a (possibly infinite) sequence $\alpha = s_0, a_1, s_1, a_2, \ldots$ of alternating states and actions such that each (s_i, a_{i+1}, s_{i+1}) is an allowed step of A, s_0 is a start state, and when α is finite, the last element is a state.

A finite execution is *fair* if no local action is enabled from its last state. An infinite execution is *fair* if for every class $F \in fair(A)$, either actions from F are taken infinitely many times or infinitely many times no F action is enabled. In other words, an infinite execution $s_0, a_1, s_1, a_2, \ldots$ is fair if for every class $F \in fair(A)$, either $a_i \in F$ for infinitely many i's, or no action of F is enabled from s_i for infinitely many i's.

A sequence α over $ext(A)$ is an *A-trace* if $\alpha = \eta | ext(A)$ for some fair execution η of A. A finite sequence over $ext(A)$ is a *partial A-trace* if it is a finite prefix of some A-trace. Similarly, any sequence whose restriction to $ext(A)$ is an A-trace is called *A-consistent*, and any finite prefix of an A-consistent sequence is *partial A-consistent*. We let $traces(A)$ be the set of all A-traces. Thus, $traces(A)$ are exactly the sequences of the externally visible actions in A's fair executions, and we define $beh(A) = (ext(A), traces(A))$ to be the *behavior of A*.

The following theorem establishes that finite executions of an automaton A are partial A-consistent. The proof of the theorem appears in Lynch and Stark [1989] and Reingold et al. [1992], where a state-by-state construction of a fair execution starting with a finite execution is described. The proof depends on the Axiom of Choice.

THEOREM 2.4.1. *Let A be an automaton and let α be a finite execution of A. Then α is partial A-consistent.*

We sometimes write A as a shorthand for $ext(A)$; thus, $\alpha | A$ denotes the restriction of α to A's externally visible actions. As with behaviors, we say that A and A' are *disjoint* if $acts(A) \cap acts(A') = \emptyset$.

2.5. COMPOSITION OF AUTOMATA. Two I/O automata running in parallel and interacting through coordinated mutual actions can be described by another I/O automaton, called the "composition". We restrict composition to "compatible" automata in order to maintain the idea that each action of the composition is controlled by at most one component. We show in Lemma 2.5.1 that composition is an associative and commutative operation on mutually compatible automata, and we show in Lemma 2.5.2 that the composition is an explicit representation of the behavior generated by the parallel execution of the component automata.

We say that two automata A and B are *compatible* if every action common to both is either an input of one and output of the other, or is an input of both. The *composition* of compatible automata A and B is an automaton $C = A \circ B$ such that:

(1) $in(C) = in(A) \cup in(B) - (out(A) \cup out(B))$.
(2) $out(C) = out(A) \cup out(B)$.
(3) $internal(C) = internal(A) \cup internal(B)$.
(4) $states(C) = states(A) \times states(B)$ and $start(C) = start(A) \times start(B)$.
(5) $((s_A, s_B), a, (s'_A, s'_B)) \in steps(C)$ if one of the following holds:

—$a \in acts(A) - acts(B)$, $(s_A, a, s'_A) \in steps(A)$, and $s_B = s'_B$;
—$a \in acts(B) - acts(A)$, $(s_B, a, s'_B) \in steps(B)$, and $s_A = s'_A$;
—$a \in acts(A) \cap acts(B)$, $(s_A, a, s'_A) \in steps(A)$, and $(s_B, a, s'_B) \in steps(B)$.

(6) $fair(C) = fair(A) \cup fair(B)$. Note that, since A and B do not have any common local actions, $fair(C)$ is indeed a partition of C's local actions.

The following lemma is proved in Lynch and Tuttle [1987]. It establishes that composition of automata is associative and commutative, modulo renaming of states of the resulting automata.

LEMMA 2.5.1. *Let A_1, A_2, and A_3 be pairwise compatible automata. Then $A_1 \circ (A_2 \circ A_3) = (A_1 \circ A_2) \circ A_3$ and $A_1 \circ A_2 = A_2 \circ A_1$ modulo renaming of states.*

The composition of automata induces a composition of behaviors. The following lemma is proved in Lynch and Tuttle [1987], it shows that the behavior of the composition of two automata is just the parallel composition of the behaviors of the two automata.

LEMMA 2.5.2. *Let A and B be compatible automata. Then $beh(A \circ B) = beh(A) \| beh(B)$.*

We are often interested in the behavior of systems comprising both I/O automata and "black box" reactive systems. This is accomplished by composing the behaviors of the components of the system. Formally, for a behavior S and an automaton A, we define $S \| A = A \| S$ to be the behavior $S \| beh(A)$. It follows immediately from Lemma 2.3.2 that every sequence α is $(S \| A)$-consistent if and only if it is both S- and $beh(A)$-consistent. Hence, if $ext(A) \supseteq acts(S)$, then $traces(S \| A)$ consists exactly of the S-consistent sequences in $traces(A)$.

3. *Layered Communication Systems*

In this paper, we consider three specific kinds of communication layers: FIFO layers, nonduplicating layers and order-preserving layers. In FIFO layers, successive messages from the same site are received, exactly once, in the order sent. In order-preserving layers, messages can be lost or duplicated, but not reordered. In nonduplicating layers, each message sent is received at most once, but messages can be received in any order. Since these three kinds of layers are similar in many ways, it is economical for formalize them all as special cases of a general notion of *communication layer*.

3.1. LAYER ABSTRACTIONS.

Informally, a communication layer moves messages back and forth between two sites. A transmission from a sending site to a receiving site takes place in three steps. First, the sending site takes an action that inserts a message into the communication layer. Next, the message flows through the communication layer, possibly being corrupted, duplicated, delayed, or lost along the way. Finally, the receiving site takes an action that removes a copy of the message from the communication layer. Many different transmissions can be taking place concurrently in the communication layer, since once the sending site has finished inserting a message into the communication layer, it is free to continue its computation, possibly inserting additional messages, before the first message is received.

Our formal definition of communication layer is more abstract, ignoring what goes on inside the layer and instead specifying only the behavior that is visible at the sending and receiving sites, that is, the actions of inserting and removing messages from the communication layer. Thus, in place of talking about a message "flowing" through the layer, we must talk about a pair of related actions which in general take place at different times and locations: the "send" action that enters the message into the communication layer and the "receive" action that removes the message from the communication layer. Any additional properties we might want to impose, such as the fact that the receive action is "caused" by a corresponding send action (which must have occurred earlier in time) must be specified explicitly in the definition of the particular layer.

We can abstract further by ignoring the sites at which messages are sent and received. Thus, we consider a "directionless layer" to be a particular kind of behavior, as defined in Section 2.3, whose actions consist of sends and receives of messages. It is defined by the set of messages that it can transmit and the set of the allowable traces of any communication subsystem that correctly implements the layer. A communication layer then is a directionless layer with additional structure that reflects the notion of independent two-way communication between two sites.

3.1.1. *Directionless Layers.* Let m be a *message*. We associate with m a *send* action $\mathsf{send}(m)$ and a *receive* action $\mathsf{recv}(m)$. For a set M of messages, let $\mathsf{send}(M) = \{\mathsf{send}(m)\colon m \in M\}$, $\mathsf{recv}(M) = \{\mathsf{recv}(m)\colon m \in M\}$, and $io(M) = \mathsf{send}(M) \cup \mathsf{recv}(M)$.

A *directionless layer* L consists of a set M_L of messages and a behavior $beh(L)$, where $acts(beh(L)) = io(M_L)$. We say that L is *nondegenerate* if $M_L \neq \varnothing$. We sometimes write L to refer to $beh(L)$, so for example, $acts(L)$ is the set of actions in $beh(L)$. We refer to any $m \in M_L$ as an *L-message*, and we sometimes write $\mathsf{send}_L(m)$ and $\mathsf{recv}_L(m)$ with the layer name L as a subscript to emphasize that m is an L-message. We also write send_L and recv_L without arguments to denote the sets $\mathsf{send}(M_L)$ and $\mathsf{recv}(M_L)$, respectively.

We extend relations and operations defined for behaviors to directionless layers in the obvious way. Let L_1 and L_2 be directionless layers. Then L_1 and L_2 are *disjoint* if $beh(L_1)$ and $beh(L_2)$ are disjoint behaviors, and L_1 *refines* L_2, written $L_1 \lhd L_2$, if $beh(L_1) \lhd beh(L_2)$. Similarly, the *parallel composition* of disjoint directionless layers L_1 and L_2 is the directionless layer $L = L_1 \Diamond L_2$, where $beh(L) = beh(L_1) \| beh(L_2)$ and $M_L = M_{L_1} \cup M_{L_2}$. For a layer L and disjoint layers L_1 and L_2, we say that L is *decomposable into* L_1 *and* L_2 if $beh(L)$ is decomposable into $beh(L_1)$ and $beh(L_2)$, or equivalently, if $L = L_1 \Diamond L_2$.

Let L be a directionless layer. For a set $M \subseteq M_L$, we define a directionless layer $L' = L|M$ called the *restriction of* L *to* M. The layer L' is defined by $M_{L'} = M$ and $beh(L') = beh(L)|io(M)$. We note that, if L is decomposable into L_1 and L_2,then $L|M$ is decomposable into $L_1|M$ and $L_2|M$.

3.1.2. *Communication Layers.* A *communication layer* L (or *layer* for short) between a *site* t and a *site* r is a directionless layer together with a pair (M_L^{tr}, M_L^{rt}) such that $\{M_L^{tr}, M_L^{rt}\}$ is a partition of the message set M_L and L is decomposable into (directionless layers) $L|M_L^{tr}$ and $L|M_L^{rt}$.

The elements of M_L^{tr} are the messages that can travel from site t to r, and the elements of M_L^{rt} are the messages that can travel from site r to t. We partition the actions of L according to where they occur. For messages $m \in M_L^{tr}$, $\mathsf{send}(m)$ actions take place at site t and $\mathsf{recv}(m)$ actions take place at site r. For messages $m \in M_L^{rt}$,the opposite is true. Thus, the set of actions that take place at site t is $acts^t(L) = \mathsf{send}(M_L^{tr}) \cup \mathsf{recv}(M_L^{rt})$, and the set of actions that take place at site r is $acts^r(L) = \mathsf{send}(M_L^{rt}) \cup \mathsf{recv}(M_L^{tr})$.

A layer is diagrammed in Figure 1. The two boxes represent the sites t and r. The arrows represent actions. The wiggly line represents the network connection between the two sites.

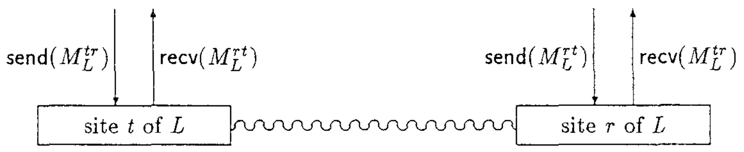

FIG. 1. A communication layer.

The definition of restriction of a directionless layer can be naturally extended to a communication layer: For a communication layer L between t and r and a subset $M \subseteq M_L$, the communication layer $L|M$ between t and r consists of the directionless layer $L|M$ together with the pair $(M_L^{tr} \cap M, M_L^{rt} \cap M)$. Obviously, $L|M$ is decomposable into $(L|M)|(M_L^{tr} \cap M)$ and $(L|M)|(M_L^{rt} \cap M)$ since $(L|M)|(M_L^{tr} \cap M) = (L|M_L^{tr})|M$ and $(L|M)|(M_L^{rt} \cap M) = (L|M_L^{rt})|M$.

The layer L is *one-way from t to r* if $M_L^{rt} = \emptyset$. Hence, in a one-way layer from t to r, all send actions take place at site t and all recv actions take place at site r. A one-way layer from r to t is similarly defined. Obviously, for any communication layer L, the layer $L^{tr} = L|M_L^{tr}$ is a one-way layer from t to r; we call it the *restriction of L to the t-to-r direction*. Similarly, $L^{rt} = L|M_L^{rt}$ is also a one-way layer; we call it the *restriction of L to the r-to-t direction*.

3.2. AXIOMS FOR COMMUNICATION LAYERS. Each send(m)-event in the trace of a layer represents the sending of a copy of m, and each recv(m)-event in the trace represents the receipt of a copy of m. In the layers we consider, each recv-event is "caused" by an earlier send-event, so messages are not spontaneously generated.

Real layers are not necessarily perfect. In a trace, messages may be corrupted, duplicated, lost, or reordered. A message m is *corrupted* if a send(m)-event causes a recv(m')-event for $m' \neq m$, it is *duplicated* if a send(m)-event causes more than one recv-event, and it is *lost* if some send(m)-event causes no recv-event. Two messages m and m' are *reordered* if some send(m)-event precedes a send(m')-event but their caused recv-events are not similarly ordered.

We are also concerned with two types of fairness. A trace satisfies *progress* if for every message m, if there are infinitely many send(m)-events, then infinitely many of them cause recv-events. A trace satisfies *weak progress* if either it contains infinitely many send-events that cause recv-events, or it contains only finitely many send-events.

We define the layer families of interest according to which of the following axioms they are required to satisfy. Formally, given a message set M and a sequence α, a *valid cause function for M and α* is a total function from the recv(M)-events to the send(M)-events of α that maps each recv(M)-event to an earlier send(M)-event. Given a valid cause function *cause* for M and α, we define the following axioms for (M, α, *cause*).

LC1 [*No corruption*] For every $m \in M$, for every recv(m)-event π in α, *cause*(π) is a send(m)-event.

LC2 [*No duplication*] The *cause* function is one-to-one.

LC3 [*No loss*] The *cause* function is onto.

LC4 [*No reordering*] For all $\mathsf{recv}(M)$-events π_1 and π_2 in α, if $cause(\pi_1)$ is earlier than $cause(\pi_2)$, then π_1 is earlier than π_2.

LC5 [*Progress*] For every $m \in M$, if α contains infinitely many $\mathsf{send}(m)$-events, then *cause* has infinitely many $\mathsf{send}(m)$-events in its range.

LC6 [*Weak Progress*] If α contains infinitely many $\mathsf{send}(M)$-events, then *cause* has an infinite range.

Let M be a message alphabet and let $\mathscr{X}$ be a subset of axioms (LC1), ..., (LC6). A sequence α is *$\mathscr{X}$-consistent with respect to M* if there exists a valid cause function *cause* for M and α such that $(M, \alpha, cause)$ satisfies each of the axions in $\mathscr{X}$. The following lemma shows that the property of a sequence being $\mathscr{X}$-consistent with respect to M is determined solely by its subsequence of actions in $io(M)$.

LEMMA 3.2.1. *Let $\mathscr{X}$ be a subset of axioms $(LC1), \ldots, (LC6)$, let α be a sequence, and let M be a message set. Then α is $\mathscr{X}$-consistent with respect to M iff $\alpha | io(M)$ is $\mathscr{X}$-consistent with respect to M.*

PROOF. Let $\alpha' = \alpha | io(M)$, and let σ be the embedding of α' in α.

Assume that α is $\mathscr{X}$-consistent with respect to M. Then there is a valid cause function *cause* for M and α such that $(M, \alpha, cause)$ satisfies each of the axioms in $\mathscr{X}$. For each $\mathsf{recv}(M)$-event π in α', we define $cause'(\pi) = \sigma^{-1}(cause(\sigma(\pi)))$.It is easily verified that *cause'* is a valid cause function for M and α', and that $(M, \alpha', cause')$ satisfies the axioms in $\mathscr{X}$. Consequently, α' is $\mathscr{X}$-consistent with respect to M.

Conversely, suppose that α' is $\mathscr{X}$-consistent with respect to M. Then there is a valid cause function *cause'* for M and α' such that $(M, \alpha', cause')$ satisfies each of the axioms in $\mathscr{X}$. Let τ be a $\mathsf{send}(M)$-event in α and define $cause(\tau) = \sigma(cause'(\sigma^{-1}(\tau)))$. It is easily verified that *cause* is a valid cause function for M and α, and that $(M, \alpha, cause)$ satisfies the axioms in $\mathscr{X}$. Consequently, α is $\mathscr{X}$-consistent with respect to M. □

A one-way layer L is said to be a *one-way $\mathscr{X}$-layer* if $traces(L)$ is the set of all sequences over $acts(L)$ that are $\mathscr{X}$-consistent with respect to M_L. A layer L is said to be an *$\mathscr{X}$-layer* if both L^{tr} and L^{rt} are one-way $\mathscr{X}$-layers.

Our use of the term "$\mathscr{X}$-consistent" is justified by the following lemma:

LEMMA 3.2.2. *Let $\mathscr{X}$ be a subset of axioms $(LC1), \ldots, (LC6)$, let L be a one-way $\mathscr{X}$-layer, and let α be a sequence. Then α is L-consistent iff α is $\mathscr{X}$-consistent with respect to M_L.*

PROOF. From the definition of L-consistency, the sequence α is L-consistent iff $\alpha | io(M_L)$ is an L-trace. Since L is an $\mathscr{X}$-layer, $\alpha | io(M_L)$ is an L-trace iff $\alpha | io(M_L)$ is $\mathscr{X}$-consistent with respect to M_L. By Lemma 3.2.1, $\alpha | io(M_L)$ is $\mathscr{X}$-consistent with respect to M_L iff α is $\mathscr{X}$-consistent with respect to M_L. □

Let $\mathscr{X}$ be a subset of the axioms that includes (LC1) and does not include (LC6). The following lemma shows that the family of one-way $\mathscr{X}$-layers is closed under restriction to a subset of the message alphabet.

LEMMA 3.2.3. *Let $\mathscr{X}$ be a subset of axioms $(LC1)$–$(LC5)$ that includes $(LC1)$, and let L be a one-way $\mathscr{X}$-layer. Then for every $M \subseteq M_L$, $L|M$ is a one-way $\mathscr{X}$-layer.*

PROOF. Let $L' = L|M$. It suffices to show that *traces*(L') is the set of all sequences over *acts*$(L') = io(M)$ that are $\mathscr{X}$-consistent with respect to $M_{L'} = M$.

Let α' be an L'-trace. Then there exists some L-trace α such that $\alpha' = \alpha|io(M)$. By Lemma 3.2.2, α is $\mathscr{X}$-consistent with respect to M_L. Hence, there is a valid cause function *cause* for M_L and α such that $(M_L, \alpha, cause)$ satisfies each of the axioms in $\mathscr{X}$. Let σ be the embedding of α' in α, and let π be a `recv`-event in α'. Since $\mathscr{X}$ contains axiom (LC1), $cause(\sigma(\pi))$ is an $io(M)$-event α and hence is in the range of σ. Thus, we may define $cause'(\pi) = \sigma^{-1}(cause(\sigma(\pi)))$. It is easily verified that *cause'* is a valid cause function for M and α', and that $(M, \alpha', cause')$ satisfies the axioms in $\mathscr{X}$. Consequently, α' is $\mathscr{X}$-consistent with respect to M.

Conversely, suppose α' is a sequence over $io(M)$ that is $\mathscr{X}$-consistent with respect to M. Since $M \subseteq M_L$, α' is a sequence over *acts*(L) that is $\mathscr{X}$-consistent with respect to M_L. Hence, α' is an L-trace. Since $L' = L|M$ and α' is a sequence over $io(M)$, it follows that α' is an L'-trace. □

Let $\mathscr{X}$ be a subset of the axioms that excludes (LC4) and (LC6). The following lemma shows that the family of one-way $\mathscr{X}$-layers is closed under layer composition.

LEMMA 3.2.4. *Let $\mathscr{X}$ be a subset of axioms (LC1)–(LC3) and (LC5). Let L_1 and L_2 be disjoint one-way $\mathscr{X}$-layers from t to r. Then $L_1 \diamond L_2$ is a one-way $\mathscr{X}$-layer from t to r.*

PROOF. Let $L = L_1 \diamond L_2$. It suffices to show that for every sequence α, α is L-consistent iff α is $\mathscr{X}$-consistent with respect to $M_L = M_{L_1} \cup M_{L_2}$.

Let α be $\mathscr{X}$-consistent with respect to M_L. From Lemma 3.2.2, it follows that α is $\mathscr{X}$-consistent with respect to M_{L_1} and with respect to M_{L_2}. It therefore follows that α is both L_1- and L_2-consistent.

In the other direction, let α be L-consistent. For $i = 1, 2$, since L_i is an $\mathscr{X}$-layer, there exists a valid cause function $cause_i$ for M_i and α such that $(M_i, \alpha, cause_i)$ satisfies the axioms in $\mathscr{X}$. Define a new function *cause* by taking the union of $cause_1$ and $cause_2$. We leave it to the reader to verify that *cause* is a valid cause function for M_L and α and that $(M_L, \alpha, cause)$ satisfies the axioms in $\mathscr{X}$. It therefore follows that α is $\mathscr{X}$-consistent with respect to M_L. □

Let $\mathscr{X}$ be a subset of the axioms that includes (LC2) and (LC3), and let L be an $\mathscr{X}$-layer. The following lemma shows that L-consistent sequences are closed under the operation of removing an L-consistent prefix:

LEMMA 3.2.5. *Let $\mathscr{X}$ be a subset of the axioms that includes (LC2) and (LC3), and let L be an $\mathscr{X}$-layer. Let α and $\alpha\beta$ be L-consistent sequences. Then β is also L-consistent.*

PROOF. By Lemma 3.2.2, α and $\alpha\beta$ are $\mathscr{X}$-consistent with respect to M_L. Thus, there exists a valid cause function *cause* for M_L and $\alpha\beta$ such that $(M_L, \alpha\beta, cause)$ satisfies the axioms in $\mathscr{X}$. Because $\mathscr{X}$ includes axioms (LC2) and (LC3), *cause* is a bijection. This implies that α has the same number of send_L and recv_L events. Since *cause* maps each recv_L-event in α to an earlier send_L-event in α, every send_L-event in α is the image under *cause* of some recv_L-event in α. Hence, *cause* maps β's recv_L-events to send_L-event in β.

Let $cause_\beta$ be the restriction of *cause* to β's events. We leave it to the reader to show that $cause_\beta$ is a valid cause function for M_L and β and that $(M_L, \beta, cause_\beta)$ satisfies the axioms of $\mathscr{X}$. It follows that β is $\mathscr{X}$-consistent with respect to M_L. By Lemma 3.2.2, β is L-consistent. □

3.3. THREE LAYER FAMILIES. We now define FIFO, nonduplicating, and order-preserving layers.

Let $\mathscr{X}_{FI}$ be the set consisting of axioms (LC1)–(LC5). Any $\mathscr{X}_{FI}$-layer is called a *FIFO layer*. Thus, the traces that are considered appropriate for each direction of a FIFO layer are those in which every message sent is eventually received, exactly once. Messages in each direction are received in the same order as they are sent, and no message is received before it is sent.

Let $\mathscr{X}_{OP}$ be the set consisting of axioms (LC1), (LC4), and (LC6). Any $\mathscr{X}_{OP}$-layer is called an *order-preserving layer*. Thus, the traces that are considered appropriate for each direction of an order-preserving layer are those in which, for every message sent, zero or more copies are received. In addition, for each message sent, any copy that is received arrives after the message was sent and before any later message travelling in the same direction is received. In other words, messages can be lost or duplicated but not reordered. Finally, if infinitely many messages are sent, then infinitely many of those messages are not lost.

Let $\mathscr{X}_{ND}$ be the set consisting of axioms (LC1), (LC2), and (LC5). Any $\mathscr{X}_{ND}$-layer is called a *nonduplicating layer*. Thus, the traces that are considered appropriate for a nonduplicating layer are those in which, for every message sent, at most one copy is received, and no message is received before it is sent. If infinitely many copies of any message are sent, then infinitely many copies of that message are also received.

3.4. PROPERTIES OF NONDUPLICATING LAYERS. The following lemma establishes closure properties of nonduplicating layers. The first two parts of the lemma are the counterparts of Lemmas 3.2.3 and 3.2.4 for nonduplicating layers but with the one-way restriction lifted. The third part establishes that nonduplicating layers are decomposable according to partitions of their message alphabet.

LEMMA 3.4.1. *Let ND be a $\mathscr{X}_{ND}$-layer. Then*

(1) *For every $M' \subseteq M_{ND}$, $ND|M'$ is a $\mathscr{X}_{ND}$-layer.*
(2) *For every $\mathscr{X}_{ND}$-layer ND' disjoint from ND, $ND' \diamond ND$ is a $\mathscr{X}_{ND}$-layer.*
(3) *Let M_1 and M_2 partition M_{ND}. Then ND decomposes into $ND|M_1$ and $ND|M_2$.*

PROOF. The rather tedious proof of the first two parts follows directly from Lemmas 3.2.3 and 3.2.4, using the observation that for every $M', M'' \subseteq M_{ND}$, $ND|M'|M'' = ND|M''|M'$. The third part is an immediate corollary of the second part. □

The following lemma shows that in a nonduplicating layer *ND*, finite prefixes of *ND*-consistent sequences are *ND*-consistent. Thus, any partial *ND*-consistent sequence is itself *ND*-consistent—no further actions need take place to achieve *ND*-consistency. This is in contrast, for example, to a FIFO layer *FI*, where *FI*-consistency is not achieved until every message sent has been delivered.

LEMMA 3.4.2. *Let ND be a $\mathscr{X}_{ND}$-layer. Every partial ND-consistent sequence is also ND-consistent.*

PROOF. This follows from the fact that *cause* is not required to be onto in nonduplicating layers. □

Let *ND* be a nonduplicating layer and let α be a finite *ND*-consistent sequence. We write $rcvd(\alpha, ND)$ to denote the multiset of *ND*-messages received in α. Formally, $dom[rcvd(\alpha, ND)] = \{m \in M_{ND}\colon \mathsf{recv}_{ND}(m) \text{ occurs in } \alpha\}$ and $copies[rcvd(\alpha, ND)](m)$ is the number of times $\mathsf{recv}_{ND}(m)$ occurs in α. Similarly, we write $sent(\alpha, ND)$ to denote the multiset of *ND*-messages sent in α. Finally, we define $pend(\alpha, ND) = sent(\alpha, ND) - rcvd(\alpha, ND)$ to be the multiset of *ND*-messages *pending at* α. These are the messages that are "in transit"—they have been sent but not yet received. Note that from (LC2) it follows that $pend(\alpha, ND)$ is always defined. The next lemma says that any submultiset of pending *ND*-messages after an *ND*-trace can be delivered at any time.

LEMMA 3.4.3. *Let ND be a $\mathscr{X}_{ND}$-layer and let α be a finite ND-trace. Let β be a finite sequence of recv_{ND}-events such that $rcvd(\beta, ND) \sqsubseteq pend(\alpha, ND)$. Then $\alpha\beta$ is an ND-trace.*

PROOF. Since α is $\mathscr{X}_{ND}$-consistent with respect to M_{ND}, there exists a valid cause function *cause* for M_{ND} and α such that $(M_{ND}, \alpha, cause)$ satisfies the axioms of $\mathscr{X}_{ND}$. Since $\mathscr{X}_{ND}$ includes axiom (LC2), *cause* is one-to-one. Moreover, since $rcvd(\beta, ND) \sqsubseteq pend(\alpha, ND)$, *cause* can be extended to a valid cause function *cause*$'$ for M_{ND} and $\alpha\beta$ that is also one-to-one and maps every $\mathsf{recv}(m)$-event in β to a $\mathsf{send}(m)$-event that is not in the range of *cause*. We leave it to the reader to verify that $(M_{ND}, \alpha\beta, cause')$ satisfies the axioms of $\mathscr{X}_{ND}$. □

The next lemma says that after any finite period of activity, a nonduplicating layer may act just like a nonduplicating layer starting from the start state. This is because a nonduplicating layer may lose messages, so the pending messages need never be delivered.

LEMMA 3.4.4. *Let ND be a $\mathscr{X}_{ND}$-layer, let β be a finite ND-consistent sequence, and let β' be any ND-consistent sequence. Then $\beta\beta'$ is ND-consistent. Moreover, $pend(\beta, ND) \sqsubseteq pend(\beta\gamma, ND)$ for every finite $\gamma \preccurlyeq \beta'$.*

PROOF. The proof relies on the fact that a nonduplicating layer can lose finitely many messages. Details are left to the reader. □

The special properties of I/O automata allow us to prove an analog to Lemma 2.3.3 for the composition of an automaton with a nondisjoint layer.

LEMMA 3.4.5. *Let A be an automaton and ND a $\mathscr{X}_{ND}$-layer. Let α be a finite sequence over any superset of $acts(A\|ND)$. Then α is partial$(A\|ND)$-consistent iff α is partial A-consistent and α is ND-consistent.*

PROOF. In one direction, the claim is trivial. In the other direction, it suffices to show the existence of an execution η of A which is both fair and *ND*-consistent, such that $\alpha|(A\|ND) \preccurlyeq \eta|(A\|ND)$. Such an execution η is constructed along the same lines as the proof of Theorem 2.4.1. *ND*-consistency of η is guaranteed by occasionally adding $\mathsf{recv}(m)$ actions to the execution when such an action does not violate ND-consistency. Being input actions of A, they can always be added. This will guarantee that axiom (LC5) is satisfied. □

4. *Implementation of Layers*

The idea of a layered architecture (cf. Tannenbaum [1989], Bochmann and Gecsei [1977] and Zimmermann [1980]) is to find protocols to implement a given communication layer L_1 on top of another given communication layer L_2. A protocol consists of two independent processes A^t and A^r for sites t and r, respectively, that have the proper interface to L_1 and L_2. When the protocol is expressed as a pair of I/O automata, the two processes running together in parallel can be viewed as a single I/O automaton $A = A^t \circ A^r$. Having a proper interface to L_1 means that A has the sets of actions required by L_1. For example, A's output actions should include $\mathsf{recv}(M_{L_1})$. A proper interface to L_2 means that A interacts with L_2 in the correct manner. For example, A's output actions should include $\mathsf{send}(M_{L_2})$. Such an implementation only makes sense when L_1 and L_2 are disjoint. The implementation is correct if the behavior of A when restricted to L_2-consistent sequences yields L_1-consistent sequences.

Because most of the properties of interest in an implementation depend only on the composition automaton A, we first define the general notion of an automaton A implementing L_1 on L_2. We then define a protocol implementing L_1 on L_2 as an "independent" pair (A^t, A^r) of automata whose composition $A^t \circ A^r$ implements L_1 on L_2.

4.1. Layer Implementations. Formally, let L_1 and L_2 be directionless layers. We say that an automaton A is *consistent with* L_1 *on* L_2 if L_1 and L_2 are disjoint, and the following conditions are satisfied:

(1) $in(A) \supseteq \mathsf{send}(M_{L_1}) \cup \mathsf{recv}(M_{L_2})$.
(2) $out(A) \supseteq \mathsf{recv}(M_{L_1}) \cup \mathsf{send}(M_{L_2})$.

We say that A *implements* L_1 *on* L_2 if A is consistent with L_1 on L_2 and $A \| L_2 \lhd beh(L_1)$.

The following theorem shows that A implements L_1 on L_2 if every L_2-consistent A-trace is also L_1-consistent:

Theorem 4.1.1. *Let L_1 and L_2 be directionless layers, and let A be automaton that is consistent with L_1 on L_2. Then A implements L_1 on L_2 iff every A-trace that is L_2-consistent is also L_1-consistent.*

Proof. In one direction the claim is trivial. In the other direction, assume that every A-trace that is L_2-consistent is also L_1-consistent. We must show that A implements L_1 on L_2, that is, that $A \| L_2 \lhd beh(L_1)$.

Let $S = A \| L_2$. By definition, $acts(S) = acts(A) \cup acts(L_2) \supseteq acts(A)$. The compatibility requirements imply that $acts(A) \supseteq acts(L_1)$. Hence, $acts(S) \supseteq acts(L_1)$. It remains to show that every sequence in $traces(S)$ is L_1-consistent. Let $\beta \in traces(S)$. Then $\beta | A \in traces(A)$ and $\beta | L_2 \in traces(L_2)$. Since $acts(A) \supseteq acts(L_2)$, we have $(\beta | A) | L_2 \in traces(L_2)$; hence $\beta | A$ is an A-trace that is L_2-consistent. By assumption, $\beta | A$ is L_1-consistent. But this means that $(\beta | A) | L_1 \in traces(L_1)$. Since $acts(A) \supseteq acts(L_1)$, then $(\beta | A) | L_1 = \beta | L_1$; thus $\beta | L_1 \in traces(L_1)$; that is, β is L_1-consistent.

We have shown that $A \| L_2 \lhd beh(L_1)$. That is, A implements L_1 on L_2. □

The following theorem establishes a transitivity property of layer implementations.

THEOREM 4.1.2. *Let L_1, L_2, and L_3 be pairwise disjoint directionless layers. Let A and B be automata such that A implements L_1 on L_2, and B implements L_2 on L_3.*[2] *Assume further that $acts(A) \cap acts(B) = acts(L_2)$. Then $A \circ B$ implements L_1 on L_3.*

PROOF. From the assumptions of the theorem, it follows that A and B are compatible automata and that $A \circ B$ is consistent with L_1 on L_3. It remains to show that $beh(A \circ B) \| beh(L_3) \lhd beh(L_1)$.

From the given implementations, we have

$$beh(A) \| beh(L_2) \lhd beh(L_1) \tag{1}$$

and

$$beh(B) \| beh(L_3) \lhd beh(L_2). \tag{2}$$

Lemma 2.3.5 applied to (2) yields

$$beh(A) \| (beh(B) \| beh(L_3)) \lhd beh(A) \| beh(L_2). \tag{3}$$

By Lemma 2.3.1, (1) and (3) yield

$$beh(A) \| (beh(B) \| beh(L_3)) \lhd beh(L_1). \tag{4}$$

By Lemmas 2.3.4 and 2.5.2

$$beh(A \circ B) \| beh(L_3) = beh(A) \| (beh(B) \| beh(L_3)). \tag{5}$$

Consequently, (4) and (5) yield

$$beh(A \circ B) \| beh(L_3) \lhd beh(L_1). \tag{6}$$

Hence, $(A \circ B)$ implements L_1 on L_3. □

The following theorem describes a parallel composition of layer implementations.

THEOREM 4.1.3. *Let L_1, L_2, K_1, and K_2 be pairwise disjoint directionless layers. Suppose A_1 and A_2 are disjoint automata such that A_1 implements L_1 on K_1 and A_2 implements L_2 on K_2. Then $A_1 \circ A_2$ implements $L_1 \diamond L_2$ on $K_1 \diamond K_2$.*

PROOF. The proof is trivial because of the disjointness assumptions. Details are left to the reader. □

4.2. PROTOCOLS. A protocol is a pair of automata that communicate over an underlying layer. We assume two physical sites t and r. One automaton, A^t, is located at site t and the other, A^r is at site r. The underlying layer is naturally assumed to be a communication layer between t and r. Formally, the pair $P = (A^t, A^r)$ is a *protocol* if A^t and A^r are disjoint automata. It is said to be *with respect to* a communication layer L if $acts(A^t) \supseteq acts^t(L)$, and $acts(A^r) \supseteq acts^r(L)$. The composition $A = A^t \circ A^r$ is the *automaton* of P.

[2] Here and in the remainder of the paper, we assume without explicit mention that the internal action set of any automaton is disjoint from all other sets of actions under consideration.

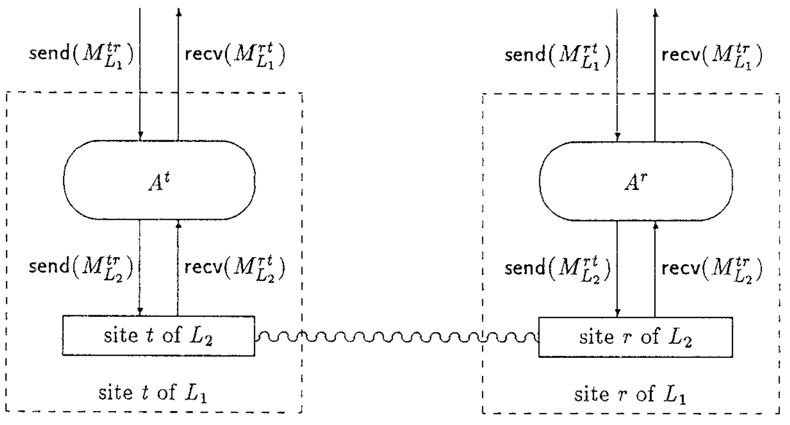

FIG. 2. Implementation of layer L_1 on layer L_2.

Protocols can be composed in the obvious way. The protocols $P_1 = (A^t, A^r)$ and $P_2 = (B^t, B^r)$ are *compatible* if A^t and B^t are compatible automata, A^r and B^r are compatible automata, and $P_1 \circ P_2 = (A^t \circ B^t, A^r \circ B^r)$ is a protocol. The protocol $P_1 \circ P_2$ is called the *composition* of P_1 and P_2. Note that if L_1 and L_2 are disjoint layers between t and r, P_1 is with respect to L_1 and P_2 is with respect to L_2, and P_1 and P_2 are compatible, then $P_1 \circ P_2$ is with respect to $L_1 \diamond L_2$.

Let L_1 and L_2 be layers between t and r, and let P be a protocol. Then P *implements* L_1 *on* L_2 if P is with respect to both L_1 and L_2, and the automaton of P (directionlessly) implements L_1 on L_2. Figure 2 illustrates such a protocol. Furthermore, P is a *clean implementation* of L_1 on L_2 if $acts(A) = acts(L_1) \cup acts(L_2)$, where A is the automaton of P.

The following theorem establishes a transitivity property of protocols implementing layers. It follows immediately from the definitions and Theorem 4.1.2.

THEOREM 4.2.1. *Let L_1, L_2, and L_3 be pairwise disjoint layers. Let P_1 be a protocol that implements L_1 on L_2, and let P_2 be a protocol that implements L_2 on L_3. Assume further that P_1 and P_2 are compatible. Then $P_1 \circ P_2$ is a protocol that implements L_1 on L_3.*

The following theorem describes a parallel composition of protocols implementing layers. It follows immediately from the definitions and Theorem 4.1.3.

THEOREM 4.2.2. *Let L_1, L_2, K_1, and K_2 be pairwise disjoint layers. Let P_1 be a protocol that implements L_1 on K_1, and let P_2 be a protocol that implements L_2 on K_2. Assume further that P_1 and P_2 are compatible. Then $P_1 \circ P_2$ is a protocol that implements $L_1 \diamond L_2$ on $K_1 \diamond K_2$. Moreover, if P_1 and P_2 are both clean implementations of their respective layers, then $P_1 \circ P_2$ is a clean implementation of $L_1 \diamond L_2$ on $K_1 \diamond K_2$.*

4.3. THE RELIABLE MESSAGE TRANSMISSION PROBLEM.

The reliable message transmission problem is to show that for every FIFO layer, there is a protocol that implements it on a suitable nonduplicating layer.

More formally, let *FI* be a FIFO layer, let *ND* be a nonduplicating layer, and let P be a protocol with respect to *ND*. We say that the pair (P, ND)

```
Transmitter $A^t$

Variables:
    $queue$, a finite queue over $M_{FI}$,
        initially empty
    $flag$, a Boolean, initially true

send$_{FI}(m)$, $m \in M_{FI}$:
    effect:
        add $m$ to $queue$

send$_{OP}(m, b)$ $m \in M_{FI}$, $b$ a Boolean:
    precondition:
        $m$ is first on $queue$
        $b = flag$

recv$_{OP}(b)$, $b$ a Boolean:
    effect:
        if $b = flag$ then
            remove first element from $queue$
            $flag := \neg flag$
```

```
Receiver $A^r$

Variables:
    $queue$, a finite queue over $M_{FI}$,
        initially empty
    $flag$, a Boolean, initially false

recv$_{FI}(m)$, $m \in M_{FI}$:
    precondition:
        $m$ is first on $queue$
    effect:
        remove first element from $queue$

send$_{OP}(b)$, $b$ a Boolean:
    precondition:
        $b = flag$

recv$_{OP}(m, b)$, $m \in M_{FI}$, $b$ a Boolean:
    effect:
        if $b \neq flag$ then
            add $m$ to $queue$
            $flag := \neg flag$
```

FIG. 3. A distributed implementation of *FI* on *OP*.

solves RMTP for FI if *P* implements *FI* on *ND*. The *reliable message transmission problem* (*RMTP*) is to show that for every FIFO layer *FI*, there exists a pair (*P*, *ND*) that solves RMTP for *FI*.

5. *A Solution to RMTP*

We construct a solution to RMTP for an arbitrary FIFO layer *FI* with a finite message alphabet. Following Afek and Gafni [1988], we obtain the solution from two basic constructions. The first implements an arbitrary *one-way FIFO layer* with a finite message alphabet on a suitable *two-way* order-preserving layer and is given in Section 5.1. The second implements an arbitrary *one-way* order-preserving layer with a finite message alphabet on a suitable *two-way* nonduplicating layer and is given in Section 5.2. These constructions are combined in Section 5.3.

5.1. IMPLEMENTATION OF A FIFO LAYER ON AN ORDER-PRESERVING LAYER. Let *FI* be a one-way FIFO layer from a "transmitter" *t* to a "receiver" *r* and let *OP* be a disjoint order-preserving layer with $M_{OP}^{tr} = M_{FI} \times \{0, 1\}$ and $M_{OP}^{rt} = \{0, 1\}$. We construct a protocol $P_A = (A^t, A^r)$ that implements *FI* on *OP*. The protocol P_A is the I/O automaton version of the Alternating Bit Protocol [Bartlett et al. 1969].

The automata A^T and A^r are given in Figure 3, in a form that is standard for I/O automata. (See, for example, Lynch and Saias [1992].) The fairness partition for A^t has one class containing all of the send$_{OP}$ actions. The fairness partition for A^r has two classes: one for all of the send_{OP} actions, and one for all of the recv_{FI} actions.

In the Alternating Bit Protocol, the transmitter conveys to the receiver a sequence of values. The values correspond to the *FI*-messages given to the transmitter. Since I/O automata are input-enabled, incoming *FI*-messages may arrive at the transmitter faster than it can process them. A^t uses a variable *queue* to buffer those messages. Likewise, the receiver uses a variable *queue* to buffer *FI*-messages until they can be output to the environment. This is also necessary because of input-enabledness.

To convey an *FI*-message to the receiver, the transmitter sends it repeatedly, tagged with a bit corresponding to the parity of the index of that *FI*-message in the sequence. A^t uses a Boolean variable *flag* for the tag and sends *OP*-messages of the form (m, b), where m is the *FI*-message to be conveyed, and b is the current value of *flag*. Axiom (LC6) insures that at least one copy is eventually received. The transmitter stops sending the current *FI*-message and starts sending the next *FI*-message in the sequence when it receives an acknowledgment for the current *FI*-message. The acknowledgment is a Boolean value equal to the current tag. When A^t receives an *OP*-message b, where $b = flag$, it removes the first element from the queue and complements its *flag*.

The receiver learns a new *FI*-message when it receives an *OP*-message with a new tag. A^r uses a Boolean variable *flag*, which, at any given time, is equal to the parity of the index of the last *FI*-message which it has learned. When it receives an *OP*-message of the form (m, b) where $b \neq flag$, it adds m to its queue and complements *flag*. After the receiver has learned the new *FI*-message, it acknowledges it by repeatedly sending the parity of the index of the *FI*-message just received. A^r accomplishes that by repeatedly sending *flag*. Axiom (LC6) insures that at least one copy of the acknowledgment is eventually received.

Standard arguments about the Alternating Bit Protocol (see, e.g., Halpern and Zuck [1992]) can be used to show the following correctness theorem.

LEMMA 5.1.1. *The protocol P_A implements FI on OP.*

Obviously, if *OP* above is replaced by a different order-preserving layer *OP′*, which has the same size message alphabet in each direction, and which is disjoint from *FI*, then P_A can be easily modified to implement *FI* on *OP′*. This argument and Lemma 5.1.1 imply the following theorem:

THEOREM 5.1.2. *Let FI be a one-way FIFO layer from t to r with a finite message alphabet. Let OP be an order-preserving layer, disjoint from FI, such that $|M_{OP}^{tr}| = 2 \cdot |M_{FI}|$ and $|M_{OP}^{rt}| = 2$. Then it is possible to construct a clean implementation of FI on OP.*

5.2. IMPLEMENTATION OF AN ORDER-PRESERVING LAYER ON A NONDUPLICATING LAYER. Let *OP* be a one-way order-preserving layer from a "transmitter" t to a "receiver" r with finite message alphabet M_{OP}, and let *ND* be a disjoint nonduplicating layer with $M_{ND}^{tr} = M_{OP} \times \{0\}$ and $M_{ND}^{rt} = \{query\}$. For every $m \in M_{OP}$, we abbreviate the pair $(m, 0) \in M_{ND}^{tr}$ by $\hat{m}$. We construct a protocol $P_B = (B^t, B^r)$ that implements *OP* on *ND*. The protocol P_B implements the idea of a "probe" as introduced in Afek and Gafni [1988].

Transmitter B^t

```
Variables:
    latest, an element of $M_{OP} \cup \{nil\}$,
        initially nil
    unanswered, a nonnegative integer,
        initially 0

$send_{OP}(m)$, $m \in M_{OP}$:
    effect:
        $latest := m$

$recv_{ND}(query)$:
    effect:
        $unanswered := unanswered + 1$

$send_{ND}(\hat{m})$, $m \in M_{OP}$:
    precondition:
        $unanswered > 0$
        $m = latest \neq nil$
    effect:
        $unanswered := unanswered - 1$
```

Receiver B^r

```
Variables:
    pending, a nonnegative integer,
        initially 0
    old, a nonnegative integer, initially 0
    for each $m \in M_{OP}$, $count[m]$,
        a nonnegative integer, initially 0

$recv_{OP}(m)$, $m \in M_{OP}$:
    precondition:
        $count[m] > old$
    effect:
        $count[w] := 0$ for all $w \in M_{OP}$
        $old := pending$

$send_{ND}(query)$:
    effect:
        $pending := pending + 1$

$recv_{ND}(\hat{m})$, $m \in M_{OP}$:
    effect:
        $pending := pending - 1$
        $count[m] := count[m] + 1$
```

FIG. 4. A distributed implementation of *OP* on *ND*.

The automata B^t and B^r are given in Figure 4. The fairness partition for B^t has one class containing all of the send_{ND} actions. The fairness partition for B^r has two classes: one for all of the send_{ND} actions, and one for all of the recv_{OP} actions.

The transmitter sends an *OP*-message to the receiver only in response to a *query* message from the receiver. The *OP*-message it sends is always the most recent *OP*-message m that was given to it, saved in *latest*. To ensure that it answers each *query* message exactly once, the transmitter keeps a variable *unanswered*, which is incremented whenever a new *query* message is received, and decremented whenever an *OP*-message is sent.

The receiver continuously sends *query messages* to the transmitter, keeping track, in *pending*, of the number of unanswered *query* messages. The receiver counts, in *count*$[m]$, the number of copies of each *OP*-message m received since the last time the receiver output an *OP*-message (or from the beginning of the run if no *OP*-message has yet been output). At the beginning, and whenever a new *OP*-message is output, the receiver sets *old* to *pending*. When *count*$[m] >$ *old*, the receiver knows that m was the *OP*-message of *latest* at some time after the receiver performed its last recv_{OP}-event. It can therefore safely output m by performing a $\mathsf{recv}_{OP}(m)$-action.

The following lemmas are used to establish the correctness of P_B.

LEMMA 5.2.1. *Let η be an ND-consistent execution of P_B. There exists a function f that maps each $\mathsf{recv}_{OP}(m)$-event π of η to an earlier $\mathsf{send}_{ND}(\hat{m})$-event $f(\pi)$ of η such that there are no recv_{OP}-events between $f(\pi)$ and π in η.*

PROOF. Let η be an *ND*-consistent execution of protocol P_B. Let $\eta' \preccurlyeq \eta$ be any execution that is a prefix of η, let $a \in acts(B^t \circ B^r)$, and let v be a program variable. We write $\#(a, \eta')$ to denote the number of a-events in η', and we write $val(v, \eta')$ to denote the value of v at the last state of η'.

Let $cause_{ND}$ be a valid cause function for M_{ND} and η such that $(M_{ND}, \eta, cause_{ND})$ satisfies the axioms in $\mathscr{X}_{ND}$. Let π be a $\mathsf{recv}_{OP}(m)$-event in η, and let $\eta_1 \preccurlyeq \eta$ end with the state immediately preceding π. From the precondition on recv_{OP}-actions, it follows that $val(count[m], \eta_1) > 0$. Since $count[m]$ can only be incremented by $\mathsf{recv}_{ND}(\hat{m})$-actions, it follows that some $\mathsf{recv}_{ND}(\hat{m})$-event π' occurs earlier than π in η. Hence, $cause_{ND}(\pi')$ is a $\mathsf{send}_{ND}(\hat{m})$-event that occurs earlier than π. Let $f(\pi)$ be the latest $\mathsf{send}_{ND}(\hat{m})$-event that is earlier than π in η. We must show that there are no recv_{OP}-events between $f(\pi)$ and π.

Assume by way of contradiction that there is some recv_{OP}-event between $f(\pi)$ and π. Let π_0 be the last such event. Let $\eta_0 \preccurlyeq \eta$ end with the state immediately following π_0. Since the action $\mathsf{recv}_{OP}(m)$ is enabled at η_1, it follows from B^r that $val(count[m], \eta_1) > val(old, \eta_1)$. We proceed to show that this cannot be the case.

From the *ND*-consistency of η and the use of *unanswered* in B^t, it follows that

$$\#(\mathsf{send}_{ND}(query), \eta_0) \geq \#(\mathsf{recv}_{ND}(query), \eta_0) \geq \sum_{w \in M_{OP}} \#(\mathsf{send}_{ND}(\hat{w}), \eta_0).$$

From B^r and the *ND*-consistency of η, it follows that

$$\begin{aligned} val(pending, \eta_0) &= \#(\mathsf{send}_{ND}(query), \eta_0) - \sum_{w \in M_{OP}} \#(\mathsf{recv}_{ND}(\hat{w}), \eta_0) \\ &\geq \sum_{w \in M_{OP}} \#(\mathsf{send}_{ND}(\hat{w}), \eta_0) - \sum_{w \in M_{OP}} \#(\mathsf{recv}_{ND}(\hat{w}), \eta_0). \end{aligned}$$

It therefore follows from the ND-consistency of η that for every $w \in M_{OP}$,

$$val(pending, \eta_0) \geq \#(\mathsf{send}_{ND}(\hat{w}), \eta_0) - \#(\mathsf{recv}_{ND}(\hat{w}), \eta_0) \geq 0. \tag{7}$$

From B^r and the fact that there are no recv_{OP}-events between π_0 and π, it follows that $val(old, \eta_1) = val(old, \eta_0)$. From B^r, $val(old, \eta_0) = val(pending, \eta_0)$. From line (7), we have $val(pending, \eta_0) \geq \#(\mathsf{send}_{ND}(\hat{m}), \eta_0) - \#(\mathsf{recv}_{ND}(\hat{m}), \eta_0)$. Since there are no $\mathsf{send}_{ND}(\hat{m})$-events between π_0 and π and η is *ND*-consistent, the number of $\mathsf{recv}_{ND}(\hat{m})$-events between π_0 and π is at most $\#(\mathsf{send}_{ND}(\hat{m}), \eta_0) - \#(\mathsf{recv}_{ND}(\hat{m}), \eta_0)$. Finally, $val(count[m], \eta_0) = 0$ and $count[m]$ is incremented only by $\mathsf{recv}_{ND}(\hat{m})$-events, so $val(count[m], \eta_1)$ is at most the number of $\mathsf{recv}_{ND}(\hat{m})$-events between π_0 and π. Consequently, $val(count[m], \eta_1) \leq val(old, \eta_1)$, a contradiction. □

LEMMA 5.2.2. *Let η be an ND-consistent execution of P_B. There exists a function g that maps each $\mathsf{send}_{ND}(\hat{m})$-event τ of η to an earlier $\mathsf{send}_{OP}(m)$-event $g(\tau)$ of η such that there are no send_{OP}-events between $g(\tau)$ and τ in η.*

PROOF. Let η be an *ND*-consistent execution of protocol P_B. Let τ be a $\mathsf{send}_{ND}(\hat{m})$-event in η. From the precondition on send_{ND}-actions, $latest \neq$ nil in the state immediately preceding τ. Since *latest* can only be set by send_{OP}-actions, some send_{OP}-event τ' occurs earlier than τ in η. Let $g(\tau)$ be the most recent send_{OP}-event that is earlier than τ. It follows immediately from B^t and the definition of g that $latest = m$ in every state between $g(\tau)$ and τ. Hence, $g(\tau)$ is a $\mathsf{send}_{OP}(m)$-event as required. $\square$

LEMMA 5.2.3. *Let η be an infinite fair ND-consistent execution of P_B containing at least one* send_{OP}*-event. Then η has infinitely many* recv_{OP}*-events.*

PROOF. Assume η is an infinite fair ND-consistent execution of P_B containing at least one send_{OP}-event. Then from some point on, $latest \neq \mathsf{nil}$. Because of the fairness of B^r, there are infinitely many $\mathsf{send}_{ND}(query)$-events. Because of the ND-consistency of η, there are infinitely many $\mathsf{recv}_{ND}(query)$-events. Because of the fairness of B^t and the assumption that $latest \neq \mathsf{nil}$, there are infinitely many send_{ND}-events. Since the message alphabet is finite, there is some m for which there are infinitely many $\mathsf{send}_{ND}(\hat{m})$-events. From the ND-consistency of η, there are infinitely many $\mathsf{recv}_{ND}(\hat{m})$-events. From B^r and its fairness, it follows now that there are infinitely many $\mathsf{recv}_{OP}(m)$-events. $\square$

LEMMA 5.2.4. *The protocol P_B implements OP on ND.*

PROOF. Consider a fair ND-consistent execution η of P_B. Since $B^t \circ B^r$ has actions that are continuously enabled, η is infinite. Let f and g be the functions whose existence is guaranteed by Lemmas 5.2.1 and 5.2.2, and define $cause_{OP}(\pi) = g(f(\pi))$ for each recv_{OP}-event π. It suffices to show that $cause_{OP}$ is a valid cause function for M_{OP} and η and that $(M_{OP}, \eta, cause_{OP})$ satisfies the axioms in $\mathscr{X}_{OP}$.

Since both f and g map events to earlier events, $cause_{OP}$ is a valid cause function for M_{OP} and η. The properties of f and g guarantee that $cause_{OP}$ maps $\mathsf{recv}_{OP}(m)$-events to $\mathsf{send}_{OP}(m)$-events, so axiom (LC1) is satisfied. If π_1 and π_2 are recv_{OP}-events and π_1 is earlier than π_2, then f and g guarantee that $cause_{OP}(\pi_1)$ is not later than $cause_{OP}(\pi_2)$, so axiom (LC4) is satisfied. If there are only finitely many send_{OP}-events, axiom (LC6) is trivially satisfied. Otherwise, there are infinitely many recv_{OP}-events by Lemma 5.2.3. Since there are no recv_{OP}-events between any $f(\pi)$ and π, f is one-to-one and has infinite range. Since g maps each $\mathsf{send}_{ND}(M_{ND}^{tr})$-event to the most recent of the earlier send_{OP}-events, g also has infinite range. Hence, $cause_{OP}$ has infinite range and axiom (LC6) is satisfied. $\square$

As before, the existence of a specific protocol implies a more general result.

THEOREM 5.2.5. *Let OP be a one-way order-preserving layer from t to r with a finite message alphabet. Let ND be a nonduplicating layer, disjoint from OP, such that $|M_{ND}^{tr}| = |M_{OP}|$ and $|M_{ND}^{rt}| = 1$. Then it is possible to construct a clean implementation of OP on ND.*

The following theorem establishes that any order-preserving layer can be implemented on a nonduplicating layer with an appropriate message alphabet.

THEOREM 5.2.6. *Let OP be an order-preserving layer with a finite message alphabet. Let ND be a nonduplicating layer, disjoint from OP, such that $|M_{ND}^{tr}| = |M_{OP}^{tr}| + 1$ and $|M_{ND}^{rt}| = |M_{OP}^{rt}| + 1$. Then it is possible to construct a clean implementation of OP on ND.*

PROOF. From Lemma 3.4.1 it follows that ND can be decomposed into disjoint nonduplicating layers ND_1 and ND_2 such that $|M_{ND_1}^{tr}| = |M_{OP}^{tr}|$,

$|M^{rt}_{ND_1}| = 1$, $|M^{rt}_{ND_2}| = |M^{rt}_{OP}|$, and $|M^{tr}_{ND_2}| = 1$. From Theorem 5.2.5, it follows that there exist a clean implementation P^{tr} of OP^{tr} on ND_1 and a clean implementation P^{rt} of OP^{rt} on ND_2. Since all four layers are pairwise disjoint and the implementations are clean, P^{tr} and P^{rt} are compatible. It therefore follows from Theorem 4.2.2 that $P^{tr} \circ P^{rt}$ is a clean implementation of OP on ND. □

5.3. A SOLUTION TO RMTP. We now construct a solution to RMTP using the constructions of Sections 5.1 and 5.2 and the general composition results of Section 4.1. First we apply Theorem 4.1.2 to Theorems 5.1.2 and 5.2.6 to implement a one-way FIFO layer on a suitable nonduplicating layer. We then compose two copies of such an implemention, one in each direction, to implement a general (two-way) FIFO layer on a suitable nonduplicating layer, thereby solving RMTP.

THEOREM 5.3.1. *Let FI be a one-way FIFO layer from t to r with a finite message alphabet. Let ND be a nonduplicating layer, disjoint from FI, such that* $|M^{tr}_{ND}| = 2 \cdot |M_{FI}| + 1$ *and* $M^{rt}_{ND} = 3$. *Let M be a set of messages that is disjoint from* $M_{FI} \cup M_{ND}$ *such that* $|M| = 2 \cdot |M_{FI}| + 2$. *Then it is possible to construct a protocol that implements FI on ND with automaton A such that* $acts(A) = acts(FI) \cup acts(ND) \cup io(M)$.

PROOF. Let OP be an order-preserving layer with message alphabet M such that $|M^{tr}_{OP}| = 2 \cdot |M_{FI}|$ and $|M^{rt}_{OP}| = 2$. It follows from Theorem 5.1.2 that it is possible to construct a clean implementation P_1 of FI on OP. Since $|M^{tr}_{ND}| = 2 \cdot |M_{FI}| + 1 = |M^{tr}_{OP}| + 1$, and similarly $|M^{rt}_{ND}| = 3 = |M^{rt}_{OP}| + 1$, it follows from Theorem 5.2.6 that it is possible to construct a clean implementation P_2 of OP on ND. Since FI, OP, and ND are pairwise disjoint, and P_1 and P_2 are clean implementations, it follows from the definitions of layer implementations that P_1 and P_2 are compatible. Hence, it follows from Theorem 4.2.1 that $P_1 \circ P_2$ implements FI on ND. Let A be the automaton of $P_1 \circ P_2$. It is easy to see that $acts(A) = acts(FI) \cup acts(ND) \cup io(M)$. □

THEOREM 5.3.2. *Let FI be a FIFO layer with a finite message alphabet. Let ND be a nonduplicating layer, disjoint from FI, such that* $|M^{tr}_{ND}| = 2 \cdot |M^{tr}_{FI}| + 4$ *and* $|M^{rt}_{ND}| = 2 \cdot |M^{rt}_{FI}| + 4$. *Then it is possible to construct a protocol that implements FI on ND.*

PROOF. From Lemma 3.4.1, it follows that ND can be decomposed into two disjoint nonduplicating layers ND_1 and ND_2 such that $|M^{tr}_{ND_1}| = 2 \cdot |M^{tr}_{FI}| + 1$, $|M^{rt}_{ND_1}| = 3$, $|M^{rt}_{ND_2}| = 2 \cdot |M^{rt}_{FI}| + 1$, and $|M^{tr}_{ND_2}| = 3$. Let M_1 and M_2 be disjoint message sets such that $|M_1| = 2 \cdot |M^{tr}_{FI}| + 2$ and $|M_2| = 2 \cdot |M^{rt}_{FI}| + 2$, and assume further that M_1 and M_2 are disjoint from M_{FI} and M_{ND}. From Theorem 5.3.1, it follows that there exists a protocol P_1 with automaton A_1 that implements FI^{tr} on ND_1 such that $acts(A_1) = acts(FI^{tr}) \cup acts(ND_1) \cup io(M_1)$, and a protocol P_2 with automaton A_2 that implements FI^{rt} on ND_2 such that $acts(A_2) = acts(FI^{rt}) \cup acts(ND_2) \cup io(M_2)$. Hence, A_1 and A_2 are disjoint, so P_1 and P_2 are compatible. From Theorem 4.2.2, it now follows that $P_1 \circ P_2$ implements FI on ND. □

6. *Bounded Protocols*

The solution of RMTP presented in Section 5 is inefficient since as more *ND*-messages are lost, more are needed to transmit subsequent messages.

Consequently, the protocol runs more and more slowly as more and more *ND*-messages are lost.

One can measure, after each partial trace of the system, the number of *ND*-messages that the transmitter *must* send in order for the receiver to learn a new message, assuming a "best-case behavior" of the *ND*-layer. A solution to RMTP is bounded when this measure is bounded by a constant for a large class of partial traces. We show that there are no bounded solutions to RMTP.

6.1. BOUNDEDNESS. Let *FI* be a FIFO layer and let *ND* be a nonduplicating layer. Boundedness measures the efficiency of an RMTP solution (P, ND) in recovering from faultiness permitted by the *ND* layer. Intuitively, consider a partial trace α of the automaton of *P*. An *FI*-message can be delivered with effort k after α if there is an *ND*-consistent sequence β in which some *FI*-message, and at most k copies of *ND*-messages, are received, and $\alpha\beta$ is a partial trace. We call β a "k-good" extension of α, and a partial trace that has a k-good extension is called "k-recoverable". (The term "recoverable" is borrowed from Tempero and Ladner [1990].) Since a k-good extension is required to be *ND*-consistent, the k-recoverability of α does not depend on the ability to deliver messages that are pending at α. We call a protocol "k-bounded" if the set of k-recoverable partial traces is sufficiently large. In particular, it should include infinitely many *FI*-consistent prefixes of every trace that has infinitely many such prefixes. We remark that there is no agreement among authors on how the intuitive notion of k-boundedness should be formalized, and the technical definitions contained in the various papers on the subject differ along many dimensions. The definition we present here is a compromise between simplicity and generality.

Formally, assume (P, ND) solves RMTP for *FI*. Let A be the automaton of *P*, and let k be some integer. A sequence over $acts(A\|ND)$ is *k-good* if it is *ND*-consistent and it contains some recv_{FI}-event and at most k recv_{ND}-events. For every partial $(A\|ND)$-trace α, we say that α is *k-recoverable* if there exists a k-good sequence β such that $\alpha\beta$ is a partial $(A\|ND)$-trace. Here α represents an observation of a finite portion of an execution, and the k-recoverability of α implies that the execution can continue so that the observable portion of the continuation is k-good. The requirement that β be *ND*-consistent prevents it from being considered k-good if it depends on the delivery of *ND*-messages that are pending at the end of α. The pair (A, ND) is *k-bounded* if, for every $(A\|ND)$-trace α, if α has infinitely many *FI*-consistent prefixes, then α has infinitely many prefixes that are both *FI*-consistent and k-recoverable.

6.2. NONEXISTENCE OF A BOUNDED SOLUTION TO RMTP. Fix *FI* to be a nondegenerate one-way layer from t to r. We establish properties of general and bounded solutions to RMTP for *FI* that allow us to prove that for no k is there a k-bounded solution to RMTP for *FI*.

The first lemma states that if (P, ND) solves RMTP for *FI* and A is the automaton of *P*, then after any *FI*-consistent partial $(A\|ND)$-trace α, in order for the receiver to learn a new *FI*-message, it must receive a sequence of *ND*-messages whose multiset was not pending at α. Intuitively, if the lemma were not true, then the pending messages would be sufficient to fool the receiver into thinking a new *FI*-message had been sent, and the resulting partial $(A\|ND)$-trace would not be partial FI-consistent, contrary to the assumption that (P, ND) solves RMTP for FI.

LEMMA 6.2.1. *Let (P, ND) solve RMTP for FI, and let A be the automaton of P. Let α be an FI-consistent partial $(A\|ND)$-trace. Let β be a sequence such that $\alpha\beta$ is a partial $(A\|ND)$-trace and β contains a* recv_{FI}*-event. Then for some $m \in M_{ND}^{tr}$,*

$$copies[rcvd(\beta, ND)](m) > copies[pend(\alpha, ND)](m).$$

PROOF. Let $P = (A^t, A^r)$. Let α and β be sequences satisfying the conditions of the lemma. Assume, by way of contradiction, that $rcvd(\beta, ND^{tr}) \sqsubseteq pend(\alpha, ND^{tr})$.

Our proof proceeds as follows: We first show the existence of a partial $(A\|ND)$-trace $\alpha\beta_1$ such that β_1 describes the situation where all activity at the transmitter A^t stops after α and the receiver continues behaving as it did in β. Such a β_1 exists because the ND-messages sent by A^t in β are not needed to satisfy ND-consistency—the pending messages at α can be used instead. We then show that $\alpha\beta_1$ is not partial FI-consistent, contradicting the assumption that (P, ND) solves RMTP for FI.

Define $\beta_1 = \beta|A^r$. We first show that $\alpha\beta_1$ is a partial $(A\|ND)$-trace. By the disjointness of A^t and A^r, $(\alpha\beta_1)|A^t = \alpha|A^t$; hence $(\alpha\beta_1)|A^t$ is a partial $beh(A^t)$-trace. Since $(\alpha\beta_1)|A^r = (\alpha\beta)|A^r$, $(\alpha\beta_1)|A^r$ is a partial $beh(A^r)$-trace. Since $\alpha\beta_1$ is both partial $beh(A^r)$-consistent and partial $beh(A^t)$-consistent, it follows from Lemmas 2.3.3 and 2.5.2 that it is a partial $beh(A)$-trace. Since $rcvd(\beta_1, ND^{tr}) = rcvd(\beta, ND^{tr}) \sqsubseteq pend(\alpha, ND^{tr})$, it follows from Lemma 3.4.3 that $\alpha\beta_1$ is ND^{tr}-consistent. The sequence β_1 is finite and contains no $\mathsf{recv}_{ND^{rt}}$-events, therefore it is ND^{rt}-consistent. It follows now from Lemma 3.4.3 that $\alpha\beta_1$is ND^{rt}-consistent. Since $ND = ND^{tr}\diamond ND^{rt}$, Lemma 2.3.2 gives that $\alpha\beta_1$ is ND-consistent. Since $\alpha\beta_1$ is an ND-consistent partial $beh(A)$-trace, it follows from Lemma 3.4.5 that $\alpha\beta_1$ is a partial $(A\|ND)$-trace.

Since (P, ND) solved RMTP for FI, Theorem 4.1.1 shows that every sequence in $traces(A\|ND)$ is FI-consistent. Thus, $\alpha\beta_1$ is partial FI-consistent. Since α is FI-consistent and $\mathscr{X}_{FI}$ includes axioms (LC2) and (LC3), Lemma 3.2.5 implies that β_1 is partial FI-consistent. However, this contradicts the fact that β_1 is not partial FI-consistent since β_1 has no send_{FI}-actions and at least one recv_{FI}-action. $\square$

Let FI be a FIFO layer, ND be a nonduplicating layer, and let (P, ND) solve RMTP for FI. The following lemma states that at any point in P's automaton's execution, all the pending ND-messages can be lost, and P's automaton still continues to operate correctly. The proof is similar to that of Lemma 3.4.5 and is omitted.

LEMMA 6.2.2. *Let FI be a FIFO layer, let ND be a nonduplicating layer, let (P, ND) solve RMTP for FI, and let A be the automaton of P. Let α be a partial $(A\|ND)$-trace. Then there exists an ND-consistent sequence γ such that $\alpha\gamma$ is a $(A\|ND)$-trace, and $\alpha\gamma$ has infinitely many FI-consistent prefixes.*

The next lemma states that any partial execution of a k-bounded solution to RMTP can be extended so that the multiset of ND-messages that are pending increases with respect to the $<_k$ ordering.

LEMMA 6.2.3. *Let (P, ND) be a k-bounded solution to RMTP for FI, and let A be the automaton of P. Let α be a partial $(A \| ND)$-trace. Then there exists a partial $(A \| ND)$-trace α' such that $pend(\alpha, ND^{tr}) <_k pend(\alpha', ND^{tr})$.*

PROOF. From Lemma 6.2.2, it follows that there exists an ND-consistent sequence γ such that $\alpha\gamma$ is an $(A \| ND)$-trace and $\alpha\gamma$ has infinitely many FI-consistent prefixes. Since (P, ND) is k-bounded, infinitely many of the FI-consistent prefixes of $\alpha\gamma$ are k-recoverable. Thus, there exists an FI-consistent k-recoverable $\alpha_1 = \alpha\gamma'$ such that $\alpha \preccurlyeq \alpha_1 \preccurlyeq \alpha\gamma$. The k-recoverability of α_1 implies that there exists a k-good sequence β such that $\alpha_1 \beta$ is a partial $(A \| ND)$-trace. From Lemma 6.2.1, it follows that, for some $m \in M_{ND}^{tr}$,

$$copies[pend(\alpha_1, ND)](m) < copies[rcvd(\beta, ND)](m). \tag{8}$$

We fix m to be such a message for the remainder of this proof. From (8), β contains a $\mathsf{recv}_{ND}(m)$-action. Since β is ND-consistent, it follows that β also contains a $\mathsf{send}_{ND}(m)$-action; hence it has a prefix of the form $\beta_1 \mathsf{send}_{ND}(m)$. Let $\alpha' = \alpha_1 \beta_1 \mathsf{send}_{ND}(m)$. Obviously, α' is a partial $(A \| ND)$-trace. It remains to show that $pend(\alpha, ND^{tr}) <_k pend(\alpha', ND^{tr})$.

Since β is k-good, it contains at most k recv_{ND}-actions, so from (8) we have

$$copies[pend(\alpha_1, ND)](m) < k. \tag{9}$$

From Lemma 3.4.2, every prefix of β, in particular β_1 and $\beta_1 \mathsf{send}_{ND}(m)$, are ND-consistent. It therefore follows from Lemma 3.4.4 that

$$\begin{aligned} copies[pend(\alpha_1, ND)](m) &\leq copies[pend(\alpha_1 \beta_1, ND)](m) \\ &< copies[pend(\alpha', ND)](m). \end{aligned} \tag{10}$$

Since γ' is a prefix of γ, Lemma 3.4.2 gives that γ' is ND-consistent. By Lemma 3.4.5, α is ND-consistent. By Lemma 3.4.4, $\alpha_1 = \alpha\gamma'$ and $\alpha' = \alpha_1 \beta_1 \mathsf{send}_{ND}(m)$, are ND-consistent, and

$$pend(\alpha, ND) \sqsubseteq pend(\alpha_1, ND) \sqsubseteq pend(\alpha', ND). \tag{11}$$

Since ND consists of two disjoint layers, ND^{tr} and ND^{rt}, it follows from (11) that $pend(\alpha, ND^{tr}) \sqsubseteq pend(\alpha', ND^{tr})$. Similarly, since $m \in M_{ND}^{tr}$, it follows that (9) and (10) still hold when restricted to the one-way layer ND^{tr}. Consequently,

$$pend(\alpha, ND^{tr}) <_k pend(\alpha', ND^{tr}). \qquad \square$$

The following theorem establishes that any k-bounded solution of RMTP for a one-way FIFO layer requires the underlying nonduplicating layer to have an infinite message alphabet in the same direction.

THEOREM 6.2.4. *Let FI be a nondegenerate one-way FIFO layer from t to r, and let (P, ND) be a k-bounded solution to RMTP for FI. Then M_{ND}^{tr} is infinite.*

PROOF. Let A be the automaton of P. Let α_0 be the empty sequence (which is trivially ND-consistent). A simple induction using Lemma 6.2.3 establishes that there exists an infinite sequence $\alpha_0, \alpha_1, \ldots$ of finite ND-consistent partial $(A \| ND)$-traces such that for every $i \geq 0$, $pend(\alpha_i, ND^{tr}) <_k pend(\alpha_{i+1}, ND^{tr})$. Lemma 2.2.1 therefore implies that M_{ND}^{tr} is infinite. $\square$

A trivial corollary of Theorem 6.2.4 is:

COROLLARY 6.2.5. *Let FI be a nondegenerate FIFO layer, and let* (P, ND) *be a k-bounded solution to RMTP for FI. Then* M_{ND} *is infinite.*

It follows that there is no k-bounded solution to RMTP for *FI* that uses a finite *ND*-message alphabet.

7. *Conclusions*

In this paper, we have considered the problem of reliable communication over unreliable channels. We have presented both an algorithm and an impossibility result. On the one hand, we have demonstrated that, seemingly contrary to popular belief, there exists a correct protocol that uses only finite packet alphabets. On the other hand, we have demonstrated that any such protocol must exhibit serious degradation of performance, as more and more messages are lost and delayed. This raises the question of whether *practical* finite-alphabet protocols can exist for channels that can lose and reorder packets. The answer to this questions probably lies in the interpretation of the term "practical".

If "practical" means maintaining a bandwidth similar to the underlying channels, then the performance of our protocol is horrendous. Moreover, this is not simply a shortcoming of our protocol, but, as our impossibility result shows, it is an inherent limitation. The impossibility result says that any finite-alphabet protocol must require a large number of packets to send each message; this imposes a large penalty on the bandwidth of the channel. Later theoretical work has strengthened the claim that communicating with bounded headers over a channel that can reorder packets must incur a severe bandwidth penalty. The interested reader is referred to in Mansour and Schieber [1992], Tempero and Ladner [1990], and Wang and Zuck [1989], where a variety of impossibility results related to ours are shown.

On the other hand, the development of newer, extremely high bandwidth, communication channels raises the serious possibility that a communication protocol could be considered reasonably efficient even though it reduces the bandwidth of the underlying channel. Even then, our impossibility result shows that no *fixed* reduction in bandwidth can be maintained; rather, the reduction must worsen over time.

As usual, it is necessary to be cautious in making practical inferences from the theoretical results, for the theoretical results are based on a set of assumptions that might be weakened in practice. For example, we have assumed that the protocols must be *asynchronous*; however, simple and efficient protocols can be constructed that use information about real time, in the form of local processor clocks and bounds on the lifetime of packets (e.g., Sunshine and Dalal [1978]). Also, we have assumed that the protocols must always work correctly; however, efficient randomized protocols can be constructed that allow a small fixed probability of error (e.g., Goldreich et al. [1989]). A challenging problem is to find models that are realistic, yet are simple enough to admit theoretical analysis.

ACKNOWLEDGMENTS. We would like to thank Baruch Awerbuch and Ewan Tempero for useful discussions on this work. We also would like to thank

Jennifer Welch for her helpful comments on early drafts of our paper. Special thanks are also due to the designers of the (usually) reliable Internet, over which many of our conversations about this research were held.

REFERENCES

AFEK, Y. AND GAFNI, E. 1988. End-to-end communication in unreliable networks. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing* (Toronto, Ont., Canada, Aug. 15–17). ACM, New York, pp. 131–148.

AHO, A., ULLMAN, J., WYNER, A., AND YANNAKAKIS, M. 1982. Bounds on the size and transmission rate of communication protocols. *Comp. Math. Appl.*, *8*, 3, 205–214.

ATTIYA, H., FISCHER, M., WANG, D.-W., AND ZUCK, L. 1989. Reliable communication using unreliable channels. Manuscript.

BARTLETT, K., SCANTLEBURY, R., AND WILKINSON, P. 1969. A note on reliable full-duplex transmission over half-duplex links. *Commun. ACM 12*, 5 (May), 260–261.

BOCHMANN, G., AND GECSEI, J. 1977. A unified method for the specification and verification of protocols. In *Information Processing 77* (B. Gilchrist, ed.). North-Holland, Amsterdam, The Netherlands, pp. 229–234.

FEKETE, A., AND LYNCH, N. 1990. The need for headers: An impossibility result for communication over unreliable channels. In *CONCUR '90*, Lecture Notes in Computer Science, vol. 458. Springer-Verlag, New York, pp. 199–215.

FEKETE, A., LYNCH, N., MANSOUR, Y., AND SPINELLI, J. 1993. The impossibility of implementing reliable communication in the face of crashes. *J. ACM*, *40*, 5 (Nov.), 1087–1107.

FRANCEZ, N. 1986. *Fairness*. Spring-Verlag, New York.

GOLDREICH, O., HERZBERG, A., AND MANSOUR, Y. 1989. Source to destination communication in the presence of faults. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (Edmonton, Alb., Canada, Aug. 14–16). ACM, New York, pp. 85–101.

HAREL, D., AND PNUELI, A. 1985. On the development of reactive systems. In *Logics and Models of Concurrent Systems* (K. R. Apt, ed.). Lecture Notes in Computer Science. Springer-Verlag, New York, pp. 477–498.

HALPERN, J. Y., AND ZUCK, L. D. 1992. A little knowledge goes a long way: Knowledge-based derivations and correctness proofs for a family of protocols. *J. ACM*, *39*, 3, 449–478.

HOARE, C. A. R. 1985. *Communication Sequential Processes*. Prentice-Hall, Englewood Cliffs, N.J.

KAHN, G. 1974. The semantics of a simple language for parallel programming. In *Inf. Proc. 74*, North-Holland, New York, pp. 471–475.

LAM, S., AND SHANKAR, A. 1990. Specifying modules to satisfy interfaces. Tech. Rep. CS-TR-2082.3. Department of Computer Science, Univ. Maryland at College Park, College Park, Md., June.

LYNCH, N., MANSOUR, Y., AND FEKETE, A. 1988. Data link layer: Two impossibility results. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. Toronto, Ont., Canada, Aug. 15–17, ACM, New York, pp. 149–170.

LYNCH, N., AND SAIAS, I. 1992. Distributed algorithms: Lecture notes for 6.852. Research Seminar Series RSS 16. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., (Feb.).

LYNCH, N., AND STARK, E. (1989). A proof of the Kahn principle for Input/Output automata. *Inf. Comput.* 82, 1 (July), 81–92.

LYNCH, N., AND TUTTLE, M. 1987. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing* (Vancouver, B.C., Canada, Aug. 10–12), ACM, New York, pp. 137–151. Expanded version available as Tech. Rep. MIT/LCS/TR-387. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Mass.

LYNCH, N., AND TUTTLE, M. 1989. An introduction to Input/Output automata. *CWI Quarterly 2* 3 (Sept). 219–246.

MANNA, Z., AND PNUELI, A. 1992. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, New York.

MANSOUR, Y., AND SCHIEBER, B. 1992. The intractability of bounded protocols for on-line sequence transmission over non-FIFO channels. *J. ACM*, *39*, 4 (Oct.), 783–799.

MILNER, R. 1980. A Calculus of Communicating Systems. In *Lecture Notes in Computer Science*, vol. 92. Springer-Verlag, New York.

REINGOLD, N., WANG, D.-W., AND ZUCK, L. D. 1992. Games I/O automata play. In *CONCUR '92*. Lecture Notes in Computer Science, vol. 630. Springer-Verlag, New York, Aug., pp. 325–339.

STENNING, M. 1976. A data transfer protocol. *Computer Net. 1*, 99–110.

SUNSHINE, C., AND DALAL, Y. 1978. Connection management in transport protocols. *Computer Net. 2*, 454–473.

TANNENBAUM, A. 1989. *Computer Networks*. Prentice-Hall, Englewood Cliffs, N.J.

TEMPERO, E., AND LADNER, R. 1990. Tight bounds for weakly bounded protocols. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, (Quebec City, Que., Canada, Aug. 22–24). ACM, New York, pp. 205–218.

WANG, D.-W., AND ZUCK, L. 1989. Tight bounds for the sequence transmission problem. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing* (Edmonton, Alb., Canada). ACM, New York, pp. 73–83.

ZIMMERMANN, H. 1980. OSI reference model—The ISO model of architecture for open systems interconnection. *IEEE Trans. Commun. COM-28*, 425–432.

RECEIVED APRIL 1991; REVISED OCTOBER 1992; ACCEPTED SEPTEMBER 1993