LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-447

RELIABLE COMMUNICATION OVER UNRELIABLE CHANNELS

Yehuda Afek
Hagit Attiya
Alan Fekete
Michael Fischer
Nancy Lynch
Yishay Mansour
Da-Wei Wang
Lenore Zuck

October 1992

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Reliable Communication Over Unreliable Channels

Yehuda Afek* Hagit Attiya† Alan Fekete‡ Michael Fischer§ Nancy Lynch¶
Yishay Mansour|| Da-Wei Wang§ Lenore Zuck§

September 30, 1992

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.

Keywords: layered communication, FIFO-layers, layer implementation, packet-switching network, packet alphabet.

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

---

*Computer Science Department, Tel-Aviv University, Tel-Aviv, Israel, and AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A.

†Department of Computer Science, Technion, Haifa 32000, Israel.

‡Department of Computer Science F09, University of Sydney 2006, Australia.

§Department of Computer Science, Yale University, Box 2158 Yale Station, New Haven, CT 06520, U.S.A.

¶Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA 02139, U.S.A.

||I.B.M. T. J. Watson Research Center, P.O. Box 704 Yorktown Heights, NY 19598, U.S.A.

This research was supported in part by Research Foundation for Information Technology, University of Sydney; Department of Computer Science, Tel-Aviv University; IBM Fellowship; ONR contracts N00014–85–K–0168, N00014–91–J–1046, N00014–85–K–0445, and N00014–82–K–0154; by NSF grants CCR–8611442, CCR–8915206, IRI–9015570, DCR–8405478, CCR–8910289; and by DARPA contracts N00014–89–J–1988, N00014–87–K–0825 and N00014-92-J-4033.

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 [Tan89, BG77, Zim80] 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 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. [BSW69, Ste76, AUWY82]). 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.* [AUWY82] 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 [Ste76] 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 [BSW69]. By way of contrast, no solution is possible for the case of both reordering and duplication faults [WZ89], 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 *non-duplicating* 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 non-duplicating 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 non-duplicating 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 from 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 non-duplicating 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 [HP85, MP92]) 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 [LT87, LT89], are all examples of reactive systems since they naturally generate behavior.

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 non-interacting 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 non-duplicating 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 non-duplicating 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 non-duplicating 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 [LS90]. A preliminary version of Theorem 5.7 appears in [AFWZ89]. A preliminary version of Theorem 6.3 appears in [LMF88]. Subsequent papers consider other versions of RMTP and other definitions of efficiency. For example, [MS89] 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 6.3 for a non-uniform 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 [WZ89]. The efficiency of RMTP is investigated in [TL90] 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, [FLMS91] contains an impossibility result for RMTP in the presence of crashes that lose information. Finally, [FL90] investigates the feasibility of solving RMTP with no headers at all.

2 Formal Definitions

2.1 Mathematical Preliminaries

Let α be an arbitrary finite or infinite sequence. We say that α is a sequence *over* a set A if each element of α belongs to A, and we sometimes call A an *alphabet*. We write $\alpha' \preceq \alpha$ to denote that α' is a finite prefix of α. Let B be a set of sequences. The *restriction* of α to B is the subsequence obtained from α by deleting all elements not in B. It is denoted by $\alpha|B$. We extend restrictions to sets of sequences in the usual way.

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]

We also have need in Section 6 for a more complicated partial ordering among multisets. Let k be

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

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, i.e., it is irreflexive, antisymmetric and transitive.

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

Lemma 2.1 *Let $C = Q_1 <_k Q_2 <_k \ldots$ be a possibly infinite increasing chain of multisets, and let $U = \bigcup_i dom[Q_i]$. If U is finite, then C has at most $k|U| + 1$ elements.*

Proof: Define a measure $f(Q_i) = \sum_{u \in U} \min(k, copies[Q_i](u))$. 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. ■

2.2 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. [HP85].) The communication layers and protocols that we discuss in this paper are examples of reactive systems. Intuitively, a reactive system is a black box which 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. [Kah74, Mil80, Hoa85]).

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 an alphabet that includes $acts(S)$. 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 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 2.2 *Refinement of systems 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.

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 *Let S_1 and S_2 be behaviors and α be a sequence over any alphabet. 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 holds for partial behavior-consistent sequences providing the two behaviors are disjoint.

Lemma 2.4 *Let S_1 and S_2 be disjoint behaviors and α a finite sequence over any alphabet. 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 $\alpha \prec \gamma$ for some $(S_1 \| S_2)$-consistent sequence γ. By Lemma 2.3, γ is S_1- and S_2-consistent and the implication follows.

In the other direction, assume that $\alpha \prec \gamma_1$ and $\alpha \prec \gamma_2$, 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, γ is $(S_1 \| S_2)$-consistent. The implication now follows since $\alpha \prec \gamma$. ■

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

Lemma 2.5 *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 2.6 *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 that β is $(S \| S_2)$-consistent.

■

2.3 I/O Automata

While 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 [LT87, LT89] 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 mutually 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*, i.e., every input action is enabled from every state.
4. A *fairness partition*, $fair(A)$, on A's local actions which 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. [Fra86].) 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 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 [LS89, RWZ91], 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 2.7 *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 $acts(A) \cap acts(A') = \emptyset$.

2.4 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 2.8 that composition is an associative and commutative operation on mutually compatible automata, and we show in Lemma 2.9 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 [LT87]. It establishes that composition of automata is associative and commutative, modulo renaming of states of the resulting automata.

Lemma 2.8 *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 [LT87]; it shows that the behavior of the composition of two automata is just the parallel composition of the behaviors of the two automata.

Lemma 2.9 *Let A and B be compatible automata. Then $beh(A \circ B) = beh(A) \parallel beh(B)$.*

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 $S \parallel A = A \parallel S$ to be the behavior $S \parallel beh(A)$. It follows immediately from Lemma 2.3 that every sequence α is $(S \parallel A)$-consistent if and only if it is both S- and $beh(A)$-consistent. Hence, if $ext(A) \supseteq acts(S)$, then $traces(S \parallel A)$ 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, non-duplicating 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 non-duplicating 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 to formalize them all as special cases of a general notion of *communication layer.*

3.1 Communication Layers

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 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, i.e., 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.

Thus, we consider a communication layer to be a particular kind of behavior, as defined in Section 2.2, whose actions consist of sends and receives of messages. The communication layer specifies the allowable traces of any communication subsystem that correctly implements the layer.

3.1.1 Layer Definition

Formally, a *communication layer* L between a *site* t and a *site* r consists of:

1. A pair of disjoint sets, M_L^{tr} and M_L^{rt}. The set M_L^{tr} consists of the messages that can travel from site t to r, and the set M_L^{rt} consists of the messages that can travel from site r to t. We denote their union by M_L, and we refer to any $m \in M_L$ as an L-*message*.

2. A behavior $beh(L)$, where $acts(beh(L)) = \{\mathsf{send}, \mathsf{recv}\} \times M_L$.

We say that L is *non-degenerate* if $M_L \neq \emptyset$.

We sometimes write L to refer to $beh(L)$, so for example, $acts(L)$ is the set of actions in $beh(L)$. We call $(\mathsf{send}, m) \in acts(L)$ a *send* action and denote it by $\mathsf{send}(m)$. Similarly, we call $(\mathsf{recv}, m) \in acts(L)$ a *receive* action and denote it by $\mathsf{recv}(m)$. We extend **send** and **recv** to sets $M \subseteq M_L$ of messages in the obvious way, i.e., $\mathsf{send}(M) = \{\mathsf{send}(m) : m \in M\}$ and $\mathsf{recv}(M) = \{\mathsf{recv}(m) : m \in M\}$. 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 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

$$acts_L^t = \mathsf{send}(M_L^{tr}) \cup \mathsf{recv}(M_L^{rt}),$$

and the set of actions that take place at site r is

$$acts_L^r = \mathsf{send}(M_L^{rt}) \cup \mathsf{recv}(M_L^{tr}).$$

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.

Figure 1: A communication layer.

3.1.2 Operations Involving Layers

We extend relations and operations defined for behaviors to layers in the obvious way. Let L_1 and L_2 be 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 layers L_1 and L_2 is the layer $L = L_1 \diamond L_2$, where $beh(L) = beh(L_1) \parallel beh(L_2)$ and the message sets of L are the unions of the corresponding message sets of L_1 and L_2.

3.1.3 One-Way Layers

A 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 in the reverse direction, from r to t, is similarly defined.

The layer L can be naturally *decomposed* into two one-way layers. L^{tr}, the *restriction of L to the t-to-r direction*, is the one-way layer from t to r such that $M_{L^{tr}}^{tr} = M_L^{tr}$, $M_{L^{tr}}^{rt} = \emptyset$, and $traces(L^{tr}) = traces(L)|acts(L^{tr})$. L^{rt}, the *restriction of L to the r-to-t direction*, is defined similarly. The layers L^{tr} and L^{rt} are obviously disjoint. Moreover, $traces(L) \subseteq traces(L^{tr} \diamond L^{rt})$. Equality holds when the two directions of L are "independent", that is, when every trace of one direction can be interleaved with any trace of the other direction to yield a trace of L. We say that L is *complete* if $L = L^{tr} \diamond L^{rt}$. Note that every one-way layer is complete. In this paper, we consider only complete layers.

3.2 Axioms for Communication Layers

The layers we consider have a set of traces characterized by certain axioms relating the occurrences of **send** and **recv** actions. Here we give six axioms which are used in the next subsection to define the layers of interest.

Let L be a layer and let α be a sequence of actions in $acts(L)$. We term the pair $\pi = (i, \alpha)$ an *event* (of α) if the i^{th} element α_i of α exists. We call $a = \alpha_i$ the *action* of π and say that π is an a-event. Thus, an a-event is a particular occurrence of action a in the sequence α. We call π a **send**-event if π is a **send**(m)-event for some message m, and similarly for **recv**-events. We often identify an event with its action.

The axioms below refer to a correspondence between **recv**-events and **send**-events in α. This correspondence, formalized by a total function *cause* from **recv**-events to **send**-events of α, indicates the **send**-event that is considered to be "responsible" for each **recv**-event. We say that the pair $(\alpha, cause)$ *satisfies* the axiom if the axiom holds for the pair.

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

LC2 *[No prescience]* For each **recv**-event π in α, the corresponding event $cause(\pi)$ occurs prior to π in α.

LC3 *[No duplication]* The *cause* function is one-to-one.

LC4 *[No losses]* The *cause* function is onto.

LC5 *[No reordering]* If π and ϕ are **recv**-events and $cause(\pi)$ precedes $cause(\phi)$ in α, then π precedes ϕ in α.

LC6 *[Progress]* For each $m \in M_L$, if α contains infinitely many **send**(m)-events, then α contains infinitely many **recv**(m)-events.

Let L be a communication layer and let $\mathcal{X}$ be a subset of axioms (LC1), ..., (LC6) that includes (LC1) and (LC2). A sequence α of actions over $acts(L)$ is said to be an *$\mathcal{X}$-trace* if there exists a total function *cause* from **recv**-events to **send**-events of α such that the pair (α, cause) satisfies each of the axioms in $\mathcal{X}$.

A layer L is an $\mathcal{X}$*-layer* if $\mathit{traces}(L)$ is exactly the set of *all* $\mathcal{X}$-traces over $\mathit{acts}(L)$. Note that for fixed $\mathcal{X}$, there are many $\mathcal{X}$-layers, differing in message domains.

3.3 Three Layer Families

We now define FIFO, non-duplicating, and order-preserving layers.

Let $\mathcal{X}_{FI}$ be the set consisting of axioms (LC1)–(LC6). A *FIFO* layer is any complete layer L such that L^{tr} and L^{rt} are both $\mathcal{X}_{FI}$-layers. 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 $\mathcal{X}_{OP}$ be the set consisting of axioms (LC1), (LC2), (LC5), and (LC6). An *order-preserving* layer is any complete layer L such that L^{tr} and L^{rt} are both $\mathcal{X}_{OP}$-layers. 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. If zero copies are received, the message is said to be *lost.* If two or more copies are received, the message is said to be *duplicated.* In any case, for each message sent, any copy that is received arrives after the message was sent and before any later message traveling in the same direction is received. In other words, messages can be lost or duplicated but not reordered. In addition, if infinitely many copies of any message are sent, then infinitely many copies of that message are also received.

Let $\mathcal{X}_{ND}$ be the set consisting of axioms (LC1), (LC2), (LC3), and (LC6). A *non-duplicating* layer is any complete $\mathcal{X}_{ND}$-layer. Thus, the traces that are considered appropriate for a non-duplicating layer are those in which, for every message sent, at most one copy is received, and no message is received before it is sent. A message that is sent but never received is said to be *lost.* If infinitely many copies of any message are sent, then infinitely many copies of that message are also received.

Note that, in contrast to non-duplicating layers, traces of FIFO layers and order-preserving layers do not necessarily satisfy the defining axioms. In particular, they might not satisfy (LC5) with respect to messages going in opposite directions. Yet, these layers are composed of two one-way layers, each of whose traces do satisfy the defining axioms.

3.4 Properties of FIFO and Non-duplicating Layers

In a FIFO layer FI, the *cause* function is one-to-one and onto since it is one-to-one and onto in each direction. Hence, the removal of an FI-consistent prefix from an FI-consistent sequence yields an FI-consistent sequence.

Lemma 3.1 *Let FI be a FIFO layer and let α and $\alpha\beta$ be FI-consistent sequences. Then β is FI-consistent.*

The following lemma shows that in a non-duplicating 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 3.2 *Let ND be a non-duplicating 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 non-duplicating layers. ■

Let ND be a non-duplicating 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, $dom[rcvd(\alpha, ND)] = \{m \in M_{ND} : \mathsf{recv}_{ND}(m)$ 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) and (LC3) 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 3.3 *Let ND be a non-duplicating 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: The *cause* function for α is easily extended to $\alpha\beta$ by mapping $\mathsf{recv}(p)$-events in β to $\mathsf{send}(p)$-events in α that are not already in the range of *cause*. The conditions on the multisets ensure that this is possible. ■

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

Lemma 3.4 *Let ND be a non-duplicating 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 $\gamma \preceq \beta'$.*

Proof: The proof relies on the fact that a non-duplicating 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 2.4 for the composition of an automaton with a non-disjoint layer.

Lemma 3.5 *Let A be an automaton and ND a non-duplicating layer. Let α be a finite sequence over any set $B \supseteq acts(A \,||\, ND)$. Then*

$$\alpha \textit{ is partial } (A \,||\, ND)\textit{-consistent iff } \alpha \textit{ is partial } beh(A)\textit{-consistent and } \alpha \textit{ is } ND\textit{-consistent.}$$

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 $\alpha|A \,||\, ND \preceq \eta|A \,||\, ND$. Such an execution η is constructed along the same lines as the proof of Theorem 2.7. 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 (LC6) is satisfied. ■

4 Implementation of Layers

The idea of a layered architecture (cf. [Tan89, BG77, Zim80]) is to implement a given layer L_1 on top of another given layer L_2. An implementation consists of a protocol A having the proper interface to L_1 and L_2. When the protocol is expressed as an I/O automaton, 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 the automaton 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.

Formally, let L_1 and L_2 be layers. We say that an automaton A is *compatible with* L_1 *on* L_2 if L_1 and L_2 are disjoint layers, 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 compatible with L_1 on L_2 and $A \parallel L_2 \lhd beh(L_1)$. Thus, if $\beta \in traces(A)$ is L_2-consistent, then it is also L_1-consistent. The following theorem shows that this condition on sequences exactly characterizes the notion of implementability.

Theorem 4.1 *Let automaton A be compatible 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, i.e., that $A \parallel L_2 \lhd beh(L_1)$.

Let $S = A \parallel L_2$. 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 $A \parallel L_2 \lhd beh(L_1)$. That is, A implements L_1 on L_2. ∎

4.1 Properties of Layer Implementations

This subsection presents two general composition results on layer implementations that are useful in modularizing communication protocols. The first expresses a transitivity property that allows one layer to be implemented on another by means of intermediate layers. The second allows the parallel composition of disjoint implementations of disjoint layers.

Theorem 4.2 *Let L_1, L_2, and L_3 be mutually disjoint layers. Suppose[2] that A implements L_1 on L_2, and B implements L_2 on L_3. Assume further that $acts(A) \cap acts(B) = acts(L_2)$. Then $A \circ B$ implements L_1 on L_3.*

[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.

Proof: From the assumptions of the theorem, it follows that A and B are compatible automata and that $A \circ B$ is compatible 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 2.6 applied to (2) yields

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

By Lemma 2.2, (1) and (3) yield

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

By Lemmas 2.5 and 2.9,

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

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 4.3 *Let L_1, L_2, K_1, and K_2 be pairwise disjoint 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. ■

Our main interest in Theorem 4.3 is to allow a complete layer L, one for which $L = L^{tr} \,\|\, L^{rt}$, to be implemented by first decomposing L into its two one-way components L^{tr} and L^{rt}, implementing each separately on disjoint layers K_1 and K_2, respectively, and then combining the two implementations to yield an implementation of L on $K_1 \diamond K_2$. The following corollary justifies this method.

Corollary 4.4 *Let L, K_1, and K_2 be pairwise disjoint layers, and assume L is complete. Suppose A_1 and A_2 are disjoint automata such that A_1 implements L^{tr} on K_1 and A_2 implements L^{rt} on K_2. Then $A_1 \circ A_2$ implements L on $K_1 \diamond K_2$.*

Proof: Obvious from Theorem 4.3 and the fact that L is complete. ■

4.2 The Reliable Message Transmission Problem

The intuition behind RMTP is that a solution not only should implement L_1 on L_2, but it should consist of two "independent" processes A^t and A^r which run at sites t and r, respectively. The only way they should interact is indirectly, by sending messages back and forth through layer L_2. Our formal model is general enough to describe implementations which have "hidden channels" between the sites. This subsection provides the definitions needed to rule out such unwanted "solutions".

Let A^t and A^r be automata and let L be a layer. We call the pair (A^t, A^r) a *distributed protocol* with respect to L if A^t and A^r are disjoint automata, $acts(A^t) \supseteq acts^t(L)$, and $acts(A^r) \supseteq acts^r(L)$. Thus, all of A^t's actions can be associated with site t and all of A^r's actions can be associated with site r. We call $A^t \circ A^r$ the *automaton* of (A^t, A^r).

We say that an automaton A is *distributable* with respect to L if there exists a distributed protocol (A^t, A^r) with respect to L whose automaton is A, and we call (A^t, A^r) a *distributed decomposition* of A. A distributable implementation of layer L_1 on layer L_2 is illustrated in Figure 2.

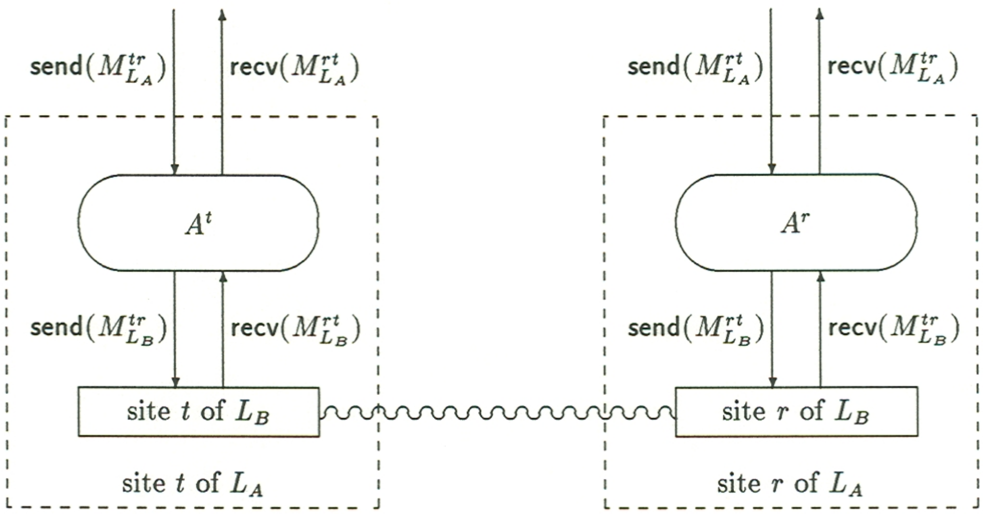

Figure 2: Implementation of layer L_A on layer L_B.

The following is obvious from the definition and shows that distributed protocols can be composed in the natural way.

Lemma 4.5 *Let L be a layer, and let (A^t, A^r) and (B^t, B^r) be distributed protocols with respect to L such that $A = A^t \circ A^r$ and $B = B^t \circ B^r$ are compatible. Then A^t and B^t are compatible, A^r and B^r are compatible, and $A \circ B$ is distributable with distributed decomposition $(A^t \circ B^t,\ A^r \circ B^r)$.*

Let A be an automaton, FI a FIFO layer, and ND a non-duplicating layer. The pair (A, ND) *solves the reliable message transmission problem (RMTP) for FI* if A is distributable with respect to ND, and A implements FI on ND.

The following lemma expresses an important property of RMTP solutions (A, ND) for FI. Namely, at any time t when A is running with an ND-channel, it is possible for the execution to continue forever so as to correctly process infinitely many FI-messages, even if all ND-messages that were pending at time t are lost. Moreover, the new execution can be chosen to have infinitely many FI-consistent prefixes. The proof is similar to that of Lemma 3.5 and is omitted.

Lemma 4.6 *Let FI be a FIFO layer, let ND be a non-duplicating layer, and let A implement FI on ND. Let α be a partial $(A \parallel ND)$-trace. Then there exists an ND-consistent sequence γ such that $\alpha\gamma$ is a $(A \parallel ND)$-trace, and $\alpha\gamma$ has infinitely many FI-consistent prefixes.*

5 A Solution to RMTP

We construct a solution to RMTP for an arbitrary FIFO layer FI with a finite message alphabet. Following [AG88], 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* non-duplicating 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 $M^{tr}_{OP} = M_{FI} \times \{0,1\}$ and $M^{rt}_{OP} = \{0,1\}$. We construct an automaton A which is distributable with respect to OP, with distributed decomposition (A^t, A^r), that implements FI on OP. The automaton A is the I/O automaton version of the Alternating Bit Protocol [BSW69].

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, [LS92].) 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 sent 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 a value to the receiver, the transmitter sends it repeatedly, tagged with a bit corresponding to the parity of the index of that value 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 value to be conveyed, and b is the current value of *flag*. The transmitter stops sending the current value and starts sending the next value in the sequence when it receives an acknowledgement for the current value. The acknowledgement 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 value when it receives a 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 value 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 value, it acknowledges it

<table>
<tr>
<td>

Transmitter A^t

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

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

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

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

</td>
<td>

Receiver A^r

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

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

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

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

</td>
</tr>
</table>

Figure 3: A distributed implementation of FI on OP.

by repeatedly sending the parity of the index of the value just received. A^r accomplishes that by repeatedly sending *flag*.

Standard arguments about the Alternating Bit Protocol (see, for example, [HZ87]) can be used to show the following correctness theorem.

Lemma 5.1 *The automaton $A^t \circ A^r$ 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 (A^t, A^r) above can be easily modified as to implement FI on OP'. This argument and Lemma 5.1 imply the following theorem.

Theorem 5.2 *Let FI be a one-way FIFO layer from t to r. 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 protocol that is distributable with respect to OP and implements FI on OP. Moreover, the automaton of the protocol has no internal actions.*

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

Theorem 5.3 *Let FI be a FIFO layer. Let OP be an order-preserving layer, disjoint from FI, such that* $|M^{tr}_{OP}| = 2\cdot|M^{tr}_{FI}|+2$ *and* $|M^{rt}_{OP}| = 2\cdot|M^{rt}_{FI}|+2$. *Then it is possible to construct a protocol that is distributable with respect to OP and implements FI on OP. Moreover, the automaton of the protocol has no internal actions.*

Proof: Let OP_1 and OP_2 be a decomposition of OP to disjoint layers such that $|M^{tr}_{OP_1}| = 2\cdot|M^{tr}_{FI}|$, $|M^{rt}_{OP_1}| = 2$, $|M^{rt}_{OP_2}| = 2\cdot|M^{rt}_{FI}|$, and $|M^{tr}_{OP_2}| = 2$. From Theorem 5.3, it follows that there exist two disjoint distributable automata, A^{tr} and A^{rt}, such that A^{tr} implements FI^{tr} on OP_1, A^{rt} implements FI^{rt} on OP_2, and neither A^{tr} nor A^{rt} has internal actions. From Corollary 4.4 it follows that $A = A^{tr} \circ A^{rt}$ implements FI on OP. It follows from Lemma 4.5 that A is distributable. It also follows from Theorem 5.3 and the definition of automata composition that A has no internal actions. ■

5.2 Implementation of an Order-preserving Layer on a Non-duplicating 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 non-duplicating layer with $M^{tr}_{ND} = M_{OP} \times \{0\}$ and $M^{rt}_{ND} = \{query\}$. For every $m \in M_{OP}$, we abbreviate the pair $(m, 0) \in M^{tr}_{ND}$ by $\hat{m}$. We construct an automaton B which is distributable with respect to ND, with distributed decomposition (B^t, B^r), that implements OP on ND. The automaton B implements the idea of a "probe" as introduced in [AG88].

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 conveys to the receiver a sequence of values with the property that, if blocks of the same value are collapsed to a single value, the resulting sequence is a subsequence of the OP-messages given to the transmitter. The receiver then outputs a subsequence of the conveyed values. It follows from the definition of an order-preserving layer that the resulting sequence of send_{OP} and recv_{OP} actions is an OP-trace. Thus, unlike the automaton A of Section 5.1, queues are not needed since both transmitter and receiver are "allowed" to drop values from the sequence.

The transmitter sends a value to the receiver only in response to a *query* from the receiver. The value 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* exactly once, the transmitter keeps a variable *unanswered* which is incremented whenever a new *query* is received, and decremented whenever a value is sent.

The receiver continuously sends *querys* to the transmitter, keeping track, in *pending*, of the number of unanswered *querys*. The receiver counts, in $count[m]$, the number of copies of each value m received since the last time it output a value (or from the beginning of the run if no value has yet been output). At the beginning, and whenever a new value is output, the receiver sets *old* to *pending*. When $count[m] > old$, the receiver knows that m was the value 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 finiteness of M_{OP} and the fact that the transmitter will always respond to *query* messages imply that the receiver will output infinitely many values (unless there is no send_{OP}-event in the run).

The arguments above now allow us to claim the following correctness result for this implementation.

Transmitter B^t	**Receiver B^r**
Variables: *latest*, an element of $M_{OP} \cup \{\mathsf{nil}\}$, initially nil *unanswered*, a nonnegative integer, initially 0 $\mathsf{send}_{OP}(m)$, $m \in M_{OP}$: **effect:** $latest := m$ $\mathsf{recv}_{ND}(query)$: **effect:** $unanswered := unanswered + 1$ $\mathsf{send}_{ND}(\hat{m})$, $m \in M_{OP}$: **precondition:** $unanswered > 0$ $m = latest \neq \mathsf{nil}$ **effect:** $unanswered := unanswered - 1$	**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 $\mathsf{recv}_{OP}(m)$, $m \in M_{OP}$: **precondition:** $count[m] > old$ **effect:** $count[w] := 0$ **for all** $w \in M_{OP}$ $old := pending$ $\mathsf{send}_{ND}(query)$: **effect:** $pending := pending + 1$ $\mathsf{recv}_{ND}(\hat{m})$, $m \in M_{OP}$: **effect:** $pending := pending - 1$ $count[m] := count[m] + 1$

Figure 4: A distributed implementation of *OP* on *ND*.

Lemma 5.4 *The automaton $B^t \circ B^r$ implements OP on ND.*

As before, renaming arguments, together with Lemma 5.4, establish the following theorem.

Theorem 5.5 *Let OP be a one-way order-preserving layer from t to r with a finite message alphabet. Let ND be a non-duplicating layer, disjoint from OP, such that $|M^{tr}_{ND}| = |M_{OP}|$ and $|M^{rt}_{ND}| = 1$. Then it is possible to construct a protocol that is distributable with respect to ND and implements OP on ND. Moreover, the automaton of the protocol has no internal actions.*

The following theorem establishes that any order-preserving layer can be implemented on a non-duplicating layer with an appropriate message alphabet. Its proof is similar to the proof of Theorem 5.3 and is omitted.

Theorem 5.6 *Let OP be an order-preserving layer with a finite message alphabet. Let ND be a non-duplicating layer, disjoint from OP, such that $|M^{tr}_{ND}| = |M^{tr}_{OP}| + 1$ and $|M^{rt}_{ND}| = |M^{rt}_{OP}| + 1$. Then it is possible to construct a protocol that is distributable with respect to ND and implements OP on ND. Moreover, the automaton of the protocol has no internal actions.*

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.

Theorem 5.7 *Let FI be a FIFO layer with a finite message alphabet. Let ND be a non-duplicating layer, disjoint from FI, such that* $|M^{tr}_{ND}| = 2 \cdot |M^{tr}_{FI}| + 3$ *and* $|M^{rt}_{ND}| = 2 \cdot |M^{rt}_{FI}| + 3$*. Then is possible to construct a protocol that is distributable with respect to ND and implements FI on ND.*

Proof: Let OP be an order-preserving layer, disjoint from FI and ND, such that $|M^{tr}_{OP}| = 2 \cdot |M^{tr}_{FI}| + 2$ and $|M^{rt}_{OP}| = 2 \cdot |M^{rt}_{FI}| + 2$. From Theorem 5.3, it is possible to construct a protocol which is distributable with respect to OP that implements FI on OP. Moreover, A, the automaton of the protocol, has no internal actions. Since $|M^{tr}_{ND}| = 2 \cdot |M^{tr}_{FI}| + 3 = |M^{tr}_{OP}| + 1$, and similarly $|M^{rt}_{ND}| = |M^{rt}_{OP}| + 1$, it follows from Theorem 5.6 that it is possible to construct a protocol which is distributable with respect to ND that implements OP on ND. Moreover, B, the automaton of the protocol, has no internal actions. From Theorem 4.2, it follows that $A \circ B$ implements FI on ND. It follows from Lemma 4.5 that $A \circ B$ is distributable with respect to ND. ■

The Alternating Bit protocol attaches an extra bit to each message in order to distinguish the current and previous messages. A more efficient encoding can accomplish the same end with only a single additional message in each direction. This allows the number of OP-messages in each direction to be reduced to 3 plus the number of FI-messages in that direction. Consequently, RMTP can be solved with an ND-layer for which $|M^{tr}_{ND}| = |M^{tr}_{FI}| + 4$ and $|M^{rt}_{ND}| = |M^{rt}_{FI}| + 4$.

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 non-duplicating layer. Boundedness measures the efficiency of an RMTP solution in recovering from faultiness permitted by the ND layer. Intuitively, consider a partial trace α. 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 [TL90].) 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 (A, ND) solves RMTP for FI, and let k be some integer. A sequence over $acts(A \parallel ND)$ 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 $(A \parallel ND)$-trace α, we say that α is ***k-recoverable*** if there exists a k-good sequence β such that $\alpha\beta$ is a partial $(A \parallel ND)$-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 $(A \parallel ND)$-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 non-degenerate one-way layer from t to r. We establish two 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 (A, ND) solves RMTP for FI, then after any FI-consistent partial $(A \parallel ND)$-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 $(A \parallel ND)$-trace would not be partial FI-consistent, contrary to the assumption that (A, ND) solves RMTP for FI.

Lemma 6.1 *Let (A, ND) solve RMTP for FI. Let α be an FI-consistent partial $(A \parallel ND)$-trace. Let β be a sequence such that $\alpha\beta$ is a partial $(A \parallel ND)$-trace and β contains a recv_{FI}-event. Then for some $p \in M_{ND}^{tr}$,*

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

Proof: Let (A^t, A^r) be a distributed decomposition of A. 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 $(A \parallel ND)$-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 (A, ND) solves RMTP for FI.

Define $\beta_1 = \beta|A^r$. We first show that $\alpha\beta_1$ is a partial $(A \parallel ND)$-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. The sequence $\alpha\beta_1$ is both partial $beh(A^r)$-consistent and partial $beh(A^t)$-consistent, so it follows from Lemmas 2.4 and 2.9 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 3.3 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 3.3 that $\alpha\beta_1$ is ND^{rt}-consistent. Since $ND = ND^{tr} \diamond ND^{rt}$, Lemma 2.3 gives that $\alpha\beta_1$ is ND-consistent. Since $\alpha\beta_1$ is an ND-consistent partial $beh(A)$-trace, it follows from Lemma 3.5 that $\alpha\beta_1$ is a partial $(A \,\|\, ND)$-trace.

Since (A, ND) solves RMTP for FI, Theorem 4.1 shows that every sequence in $traces(A \,\|\, ND)$ is FI-consistent. Thus, $\alpha\beta_1$ is partial FI-consistent. Since α is FI-consistent, Lemma 3.1 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. ■

The second lemma states that if (A, ND) solves RMTP for FI, then for every partial $(A \,\|\, ND)$-trace α, there exists another partial $(A \,\|\, ND)$-trace at which the multiset of pending ND^{tr}-messages is greater, in the ordering $>_k$, than the multiset of ND^{tr}-messages pending at α.

Lemma 6.2 *Let (A, ND) be a k-bounded solution to RMTP for FI. 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 4.6, 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 (A, 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 $\alpha \preceq \alpha_1 \prec \alpha\gamma$. 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 6.1 it follows that, for some $p \in M^{tr}_{ND}$,

$$copies[pend(\alpha_1, ND)](p) < copies[rcvd(\beta, ND)](p). \tag{7}$$

We fix p to be such a message for the remainder of this proof. From (7), β contains a $\mathsf{recv}_{ND}(p)$-action. Since β is ND-consistent, it follows that β also contains a $\mathsf{send}_{ND}(p)$-action; hence it has a prefix of the form $\beta_1\mathsf{send}_{ND}(p)$. Let $\alpha' = \alpha_1\beta_1\mathsf{send}_{ND}(p)$. 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 (7) we have

$$copies[pend(\alpha_1, ND)](p) < k. \tag{8}$$

From Lemma 3.2, every prefix of β, in particular β_1 and $\beta_1\mathsf{send}_{ND}(p)$, are ND-consistent. It therefore follows from Lemma 3.4 that

$$copies[pend(\alpha_1, ND)](p) \leq copies[pend(\alpha_1\beta_1, ND)](p) < copies[pend(\alpha', ND)](p). \tag{9}$$

Since γ' is a prefix of γ, Lemma 3.2 gives that γ' is ND-consistent. By Lemma 3.5, α is ND-consistent. By Lemma 3.4, $\alpha_1 = \alpha\gamma'$ and $\alpha' = \alpha_1\beta_1\mathsf{send}_{ND}(p)$, are ND-consistent, and

$$pend(\alpha, ND) \sqsubseteq pend(\alpha_1, ND) \sqsubseteq pend(\alpha', ND). \tag{10}$$

Since ND consists of two disjoint layers, ND^{tr} and ND^{rt}, it follows from (10) that $pend(\alpha, ND^{tr}) \sqsubseteq pend(\alpha', ND^{tr})$. Similarly, since $p \in M^{tr}_{ND}$, it follows that (8) and (9) still hold when restricted to the one-way layer ND^{tr}. Consequently,

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

■

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

Theorem 6.3 *Let FI be a non-degenerate one-way FIFO layer from t to r, and let (A, ND) be a k-bounded solution to RMTP for FI. Then M^{tr}_{ND} is infinite.*

Proof: Let α_0 be the empty sequence (which is trivially ND-consistent). A simple induction using Lemma 6.2 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 2.1 therefore implies that M^{tr}_{ND} is infinite. ■

A trivial corollary of Theorem 6.3 is:

Corollary 6.4 *Let FI be a non-degenerate FIFO layer, and let (A, 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 [MS89, TL90, WZ89] 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., [SD78]). 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., [HGM89]). 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

[AFWZ89] H. Attiya, M. Fischer, D. Wang, and L. Zuck. Reliable communication using unreliable channels. Manuscript, 1989.

[AG88] Y. Afek and E. Gafni. End-to-end communication in unreliable networks. In *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 117–130, 1988.

[AUWY82] A. Aho, J. Ullman, A. Wyner, and M. Yannakakis. Bounds on the size and transmission rate of communication protocols. *Comp. & Maths. with Appls.*, 8(3):205–214, 1982.

[BG77] G. Bochmann and J. Gecsei. A unified method for the specification and verification of protocols. In B. Gilchrist, editor, *Information Processing 77*, pages 229–234. North-Holland Publishing Co., 1977.

[BSW69] K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, May 1969.

[FL90] A. Fekete and N. Lynch. The need for headers: An impossibility result for communication over unreliable channels. In *CONCUR 90: Theories of Concurrency, LNCS 458*, pages 199–215, 1990.

[FLMS91] A. Fekete, N. Lynch, Y. Mansour, and J. Spinelli. The impossibility of implementing reliable communication in the face of crashes. Technical Memo TM–355c, Laboratory for Computer Science, Massachusetts Institute of Technology, September 1991. To appear in JACM.

[Fra86] Nissim Francez. *Fairness.* Spring-Verlag, New York, 1986.

[HGM89] A. Herzberg, O. Goldreich, and Y. Mansour. Source to destination communication in the presence of faults. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pages 85–102, 1989.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[HP85] David Harel and Amir Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, Lecture Notes in Computer Science, pages 477–498. Springer-Verlag, 1985.

[HZ87] J. Halpern and L. Zuck. A little knowledge goes a long way: Simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 269–280, 1987. Journal version to appear in J. ACM.

[Kah74] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing, 74*, pages 471–475, Amsterdam, 1974. North Holland.

[LMF88] N. Lynch, Y. Mansour, and A. Fekete. Data link layer: Two impossibility results. In *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 149–170, August 1988.

[LS89] N. Lynch and E. Stark. A proof of the Kahn principle for Input/Output automata. *Information and Computation*, 82(1):81–92, July 1989.

[LS90] S. Lam and A. Shankar. Specifying modules to satisfy interfaces. Technical Report CS-TR-2082.3, Department of Computer Science, University of Maryland at College Park, June 1990.

[LS92] N. Lynch and I. Saias. Distributed algorithms: Lecture notes for 6.852. Research Seminar Series RSS 16, Laboratory for Computer Science, Massachusetts Institute of Technology, February 1992.

[LT87] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, August 1987. Expanded version available as Technical Report MIT/LCS/TR-387, Laboratory for Computer Science, Massachusetts Institute of Technology.

[LT89] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[Mil80] Robin Milner. *A calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, New York, 1980.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, New York, 1992.

[MS89] Y. Mansour and B. Schieber. The intractability of bounded protocols for non-FIFO channels. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pages 59–72, August 1989.

[RWZ91] N. Reingold, D.-W. Wang, and L. D. Zuck. Games i/o automata play. Technical Report YALE/DCS/TR 857, Yale University, November 1991. To appear in CONCUR '92.

[SD78] C. Sunshine and Y. Dalal. Connection management in transport protocols. *Computer Networks*, 2:454–473, 1978.

[Ste76] M. Stenning. A data transfer protocol. *Computer Networks*, 1:99–110, 1976.

[Tan89] A. Tannenbaum. *Computer Networks*. Prentice Hall, 1989.

[TL90] E. Tempero and R. Ladner. Tight bounds for weakly bounded protocols. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 205–218, August 1990.

[WZ89] D. Wang and L. Zuck. Tight bounds for the sequence transmission problem. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pages 73–83, August 1989.

[Zim80] H. Zimmermann. OSI reference model—the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communication*, COM–28:425–432, April 1980.