The Data Link Layer: Two Impossibility Results

Nancy Lynch, Yishay Mansour and Alan Fekete
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract: The data link layer in a layered communication network is designed to ensure reliable data transfer over a noisy physical channel. Formal specifications are given for physical channels and data links, in terms of I/O automata. Based on these specifications, two impossibility results are proved. First, no data link protocol can tolerate crashes of the host processors on which the protocol runs. Second, any data link protocol constructed to use an arbitrary non-FIFO physical channel requires unbounded headers.

1 Introduction

Network protocols are decomposed into layers in order to reduce the complexity of their design. Each layer has a particular abstract behavior, describable in terms of a particular collection of abstract actions. This abstract behavior is provided for the use of the next higher layer, and is implemented in terms of the abstract behavior of the next lower layer. A thorough discussion of network layers can be found in [T].

The physical layer is the lowest layer in the hierarchy, and is implemented directly in terms of the physical transmission media. There are two classes of transmission media that are commonly considered, one that ensures FIFO behavior for the corresponding physical channel and the other that does not. (A physical channel is said to exhibit FIFO behavior provided that messages are received on the physical channel in the same order as they are sent.) The transmission media are noisy; therefore, the physical layer does not ensure that a message that is sent will be received.

The data link layer is the next higher layer in the network hierarchy. In contrast to the physical layer, the data link layer ensures reliable data transfer, though only across one hop in the network. This means that every message that is sent on a data link to a neighboring node is eventually received at the other end (unless a link failure occurs) and also that the data link exhibits FIFO behavior. (That is, messages are received on the data link in the same order as they are sent.)

We have taken the terminology "physical channel" and "data link" from the OSI layered communication model [Z] used by the International Standards Organization. There are many different kinds of layered networks, not all of which use the particular layers specified in the ISO model. However, most of the important layered networks have their two lowest layers very similar to those described here, although their terminology may be different. For example, the ARPANET data link layer is called the "IMP-IMP" [MW77] layer, while the SNA and DECNET data link layers are called "data link control" layers [C78,W80].

Data links are implemented using protocols that interact by communicating over physical channels. Some examples of interesting data link protocols are HDLC (proposed by ISO), SDLC (developed by

The first and third authors were supported in part by the National Science Foundation under grant CCR-86-11442, by the Office of Naval Research under contract N00014-85-K-0168 and by the Defense Advanced Research Projects Agency under contract N00014-83-K-0125. The second author was supported in part by a grant of ISEF and by the National Science Foundation under grant CCR-86-11442.

IBM) and LAPB (used by CCITT). These protocols are very similar; they all require FIFO physical channels, and they are all based on a "sliding window" automatic repeat request (ARQ) algorithm, where messages are sent in packets whose headers contain a sequence number for the message, and where acknowledgements contain the sequence number of the next message expected. Both sequence numbers are kept modulo a number that is at least one more than the size of the window, which is the maximum difference allowed between the greatest sequence number sent by the transmitter and the greatest sequence number of a message for which the sender has received an acknowledgement. The correctness of this algorithm has been proved using many different formal methods, under the assumption that the peer processes that carry out the protocol are correctly initialized. However, Baratz and Segall [BS83] show that the protocols mentioned may not reach a satisfactory initialization after the underlying physical link fails and then recovers. In [BS83] new link initialization strategies are presented, each of which can be combined with a sliding window algorithm to give a protocol that uses a small amount of memory and can tolerate an arbitrary number of link failures. The resulting protocols require access to one bit of non-volatile memory, that is, storage that retains its state across a crash of the processor on which the protocol is running.

When the physical channel does not guarantee FIFO behavior, an ARQ algorithm can still be used, so long as each message is given a distinct sequence number. The resulting algorithm (called Stenning's protocol) uses headers which may be arbitrarily long.[1]

In this paper, we give formal specifications for both the physical and data link layer, in terms of I/O automata [LT87]. Based on these specifications, we prove two impossibility results about implementing data link protocols.

First, we study the ability of a data link protocol to tolerate crashes of the host processors on which the protocol runs, without access to non-volatile storage. In the absence of non-volatile storage, a host crash can be viewed as resetting the memory of the part of the data link protocol running on that host to its distinguished initial value. We prove that it is impossible for any data link protocol to tolerate host crashes, even if the requirements of the data link protocol are stated very weakly and even if the underlying physical channel is assumed to be FIFO. This impossibilty result was conjectured in [BS83]. A very similar result has been obtained independently and concurrently by J. Spinelli (personal communication).

Second, we consider the possibility of achieving reliable data transfer with bounded headers, using a physical layer that does not ensure FIFO behavior. The headers contain information added to messages by the data link protocol before sending them on the physical channel. We prove that unbounded headers are essential for achieving correct data link behavior if the physical channels can reorder packets arbitrarily; this is the case even if the requirements on the data link are weak.

The data link protocol and the physical channel are modeled as I/O automata; thus, the formal content of our results is the nonexistence of I/O automata whose behavior has certain properties. We believe, however, that any reasonable data link protocol can be described in terms of I/O automata, and that the properties chosen accurately reflect the requirements described informally above, so that the results really assert the nonexistence of data link protocols satisfying the requirements.

The rest of the paper is organized as follows. Section 2 contains a summary of the relevant definitions from the I/O automaton model. Sections 3 and 4 contain formal specifications for the physical layer and data link layer, respectively. Section 5 describes constraints on data link protocols. Section 6 gives some specific automata that we will use as physical channels when giving the impossibility proofs. Section 7 contains our proof that no data link protocol can tolerate host crashes, and Section 8 contains our proof that unbounded headers are essential for implementing a data link layer using arbitrary non-FIFO physical channels. Finally Section 9 contains a discussion of ways in which we believe the definitions can be extended without invalidating the proofs.

[1]If there is a known bound on the time a message may remain on the link before being either lost or delivered, this may be used in conjunction with reliable clocks to derive a protocol with bounded headers.

2 The I/O Automaton Model

The *input/output automaton* model was defined in [LT87] as a tool for modeling concurrent and distributed systems. We refer the reader to [LT87] and to the expository paper [L88] for a complete development of the model, plus motivation and examples. Here, we provide a brief summary of those aspects of the model that are needed for our results.

2.1 Actions and Action Signatures

We assume a universal set of *actions*, and we refer to a particular occurrence of an action in a sequence as an *event*.

An *action signature* S is an ordered triple consisting of three pairwise-disjoint sets of actions. We write $in(S)$, $out(S)$ and $int(S)$ for the three components of S, and refer to the actions in the three sets as the *input actions*, *output actions* and *internal actions* of S, respectively. We let $ext(S) = in(S) \cup out(S)$ and refer to the actions in $ext(S)$ as the *external actions* of S. Also, we let $local(S) = out(S) \cup int(S)$, and refer to the actions in $local(S)$ as the *locally-controlled actions* of S. Finally, we let $acts(S) = in(S) \cup out(S) \cup int(S)$, and refer to the actions in $acts(S)$ as the *actions* of S. An *external action signature* is an action signature consisting entirely of external actions, that is, having no internal actions.

2.2 Input/Output Automata

An *input/output automaton* A (also called an *I/O automaton* or simply an *automaton*) consists of five components:

1. an action signature $sig(A)$,
2. a set $states(A)$ of *states*,
3. a nonempty set $start(A) \subseteq states(A)$ of *start states*,
4. a transition relation $steps(A) \subset (states(A) \times acts(sig(A)) \times states(A))$, with the property that for every state s' and input action π there is a transition (s', π, s) in $steps(A)$, and
5. an equivalence relation $part(A)$ on $local(sig(A))$, having at most countably many equivalence classes.

We refer to an element (s', π, s) of $steps(A)$ as a *step* of A. The step (s', π, s) is called an *input step* of A if π is an input action. *Output steps*, *internal steps*, *external steps* and *locally-controlled steps* are defined analogously. If (s', π, s) is a step of A, then π is said to be *enabled* in s'. Since every input action is enabled in every state, automata are said to be *input-enabled*. The partition $part(A)$ is an abstract description of the underlying components of the automaton, and is used to define fairness.

An *execution fragment* of A is a finite sequence $s_0\pi_1s_1\pi_2\ldots\pi_ns_n$ or an infinite sequence $s_0\pi_1s_1\pi_2\ldots\pi_ns_n\ldots$ of alternating states and actions of A such that $(s_i, \pi_{i+1}, s_{i+1})$ is a step of A for every i. An execution fragment beginning with a start state is called an *execution*. We denote the set of executions of A by $execs(A)$. A state is said to be *reachable* in A if it is the final state of a finite execution of A.

A *fair execution* of an automaton A is defined to be an execution α of A such that the following condition holds for each class C of $part(A)$: if α is finite, then no action of C is enabled in the final state of α, while if α is infinite, then either α contains infinitely many events from C, or else α contains infinitely many occurrences of states in which no action of C is enabled. Thus, a fair execution gives "fair turns" to each class of $part(A)$. We denote the set of fair executions of A by $fairexecs(A)$.

The *schedule* of an execution fragment α of A is the subsequence of α consisting of actions, and is denoted by $sched(\alpha)$. We say that β is a *schedule* of A if β is the schedule of an execution of A. We denote the set of schedules of A by $scheds(A)$. We say that β is a *fair schedule* of A if β is the schedule of a fair execution of A and we denote the set of fair schedules of A by $fairscheds(A)$.

The *behavior* of an execution or schedule α of A is the subsequence of α consisting of external actions, and is denoted by $beh(\alpha)$. We say that β is a *behavior* of A if β is the behavior of an execution of A. We denote the set of behaviors of A by $behs(A)$. We say that β is a *fair behavior* of A if β is the behavior of a fair execution of A and we denote the set of fair behaviors of A by $fairbehs(A)$. When an algorithm is modelled as an I/O automaton, it is the set of fair behaviors of the automaton that reflect the activity of the algorithm that is important to users.

We say that a finite behavior or schedule β of A *can leave* A *in state* s if there is a finite execution α with β as its behavior or schedule, such that the final state in α is s.

The following lemma says that no matter what has happened in any finite execution, and no matter what inputs continue to arrive from the environment, an automaton can continue to take steps to give a fair execution.

Lemma 2.1 *Let A be an I/O automaton and let γ be a sequence of input actions of A.*

1. *Suppose that α is a finite execution of A. Then there exists a fair execution α' of A such that α' is an extension of α and $beh(\alpha')|in(A) = (beh(\alpha)|in(A))\gamma$.*
2. *Suppose that β is a finite schedule of A. Then there exists a fair schedule β' of A such that β' is an extension of β and $\beta'|in(A) = (\beta|in(A))\gamma$.*

2.3 Schedule Modules

In line with our approach, where the facts about an algorithm that are important to its users are modelled by the set of fair behaviors of an automaton, we also give a formal model for a problem specification by a set of sequences of actions. More precisely, a problem will be specified by a pair consisting of an action signature and a set of sequences over the actions in that signature. (In most interesting cases, the action signature will be an external action signature.) The mathematical object used to describe a problem is called a "schedule module".

A *schedule module* H consists of two components:

1. an action signature $sig(H)$, and
2. a set $scheds(H)$ of *schedules*.

Each schedule in $scheds(H)$ is a finite or infinite sequence of actions of H.

The *behavior* of a schedule β of H is the subsequence of β consisting of external actions, and is denoted by $beh(\beta)$. We say that β is a *behavior* of H if β is the behavior of an execution of H. We denote the set of behaviors of H by $behs(H)$. We extend the definitions of fair schedules and fair behaviors to schedule modules in a trivial way, letting $fairscheds(H) = scheds(H)$ and $fairbehs(H) = behs(H)$.

We use the term *module* to designate either an automaton or schedule module. If M is a module, we sometimes write $acts(M)$ as shorthand for $acts(sig(M))$, and likewise for $in(M)$, $out(M)$, etc. If β is any sequence of actions and M is a module, we write $\beta|M$ for $\beta|acts(M)$.

2.4 Solving Problems

Now we are ready to define our notion of "solving". This notion is intended for describing the way in which particular algorithms (formalized as automata) solve particular problems (formalized as schedule modules). Let A be an automaton and H a schedule module with the same external action signature as A. Then we say that A *solves* H if $fairbehs(A) \subseteq behs(H)$.

2.5 Composition

The most useful way of combining I/O automata is by means of a composition operator, as defined in this subsection. This models the way algorithms interact, as for example when the pieces of a communication protocol at different nodes and a lower-level protocol all work together to provide a higher-level service.

2.5.1 Composition of Action Signatures

Let I be an index set that is at most countable. A collection $\{S_i\}_{i\in I}$ of action signatures is said to be *strongly compatible* if for all $i, j \in I$, we have

1. $out(S_i) \cap out(S_j) = \emptyset$,
2. $int(S_i) \cap acts(S_j) = \emptyset$, and
3. no action is in $acts(S_i)$ for infinitely many i.

Thus, no action is an output of more than one signature in the collection, and internal actions of any signature do not appear in any other signature in the collection.

The *composition* $S = \Pi_{i\in I} S_i$ of a collection of strongly compatible action signatures $\{S_i\}_{i\in I}$ is defined to be the action signature with $in(S) = \cup_{i\in I} in(S_i) \setminus \cup_{i\in I} out(S_i)$, $out(S) = \cup_{i\in I} out(S_i)$, and $int(S) = \cup_{i\in I} int(S_i)$. Thus, output actions are those that are outputs of any of the component signatures, and similarly for internal actions. Input actions are any actions that are inputs to any of the component signatures, but outputs of no component signature.

2.5.2 Composition of Automata

A collection $\{A_i\}_{i\in I}$ of automata is said to be *strongly compatible* if their action signatures are strongly compatible. The *composition* $A = \Pi_{i\in I} A_i$ of a strongly compatible collection of automata $A_{i i\in I}$ has the following components:

1. $sig(A) = \Pi_{i\in I} sig(A_i)$,
2. $states(A) = \Pi_{i\in I} states(A_i)$[2]
3. $start(A) = \Pi_{i\in I} start(A_i)$
4. $steps(A)$ is the set of triples (s_1, π, s_2) such that for all $i \in I$, if $\pi \in acts(A_i)$ then $(s_1[i], \pi, s_2[i]) \in steps(A_i)$, and if $\pi \notin acts(A_i)$ then $s_1[i] = s_2[i]$[3], and
5. $part(A) = \cup_{i\in I} part(A_i)$.

Since the automata A_i are input-enabled, so is their composition, and hence their composition is an automaton. Each step of the composition automaton consists of all the automata that have a particular action in their signatures performing that action concurrently, while the automata that do not have that action in their signatures do nothing. The partition for the composition is formed by taking the union of the partitions for the components. Thus, a fair execution of the composition gives fair turns to all of the classes within all of the component automata. In other words, all component automata in a composition continue to act autonomously. If $\alpha = s_0\pi_1 s_1 \ldots$ is an execution of A, let $\alpha|A_i$ be the sequence obtained by deleting $\pi_j s_j$ when π_j is not an action of A_i, and replacing the remaining s_j by $s_j[i]$.

The following basic results relate executions, schedules and behaviors of a composition to those of the automata being composed. The first result says that the projections of executions of a composition onto the components are executions of the components, and similarly for schedules, etc. The parts of this result dealing with fairness depend on the fact that at most one component automaton can impose preconditions on each action.

[2]Note that the second and third components listed are just ordinary Cartesian products, while the first component uses a previous definition.

[3]We use the notation s[i] to denote the i-th component of the state vector s

Lemma 2.2 *Let $\{A_i\}_{i\in I}$ be a strongly compatible collection of automata, and let $A = \Pi_{i\in I} A_i$. If $\alpha \in execs(A)$ then $\alpha|A_i \in execs(A_i)$ for all $i \in I$. Moreover, the same result holds for fairexecs, scheds, fairscheds, behs and fairbehs in place of execs.*

Certain converses of the preceding lemma are also true. The following lemma says that executions of component automata can be patched together to form an execution of the composition.

Lemma 2.3 *Let $\{A_i\}_{i\in I}$ be a strongly compatible collection of automata, and let $A = \Pi_{i\in I} A_i$. For all $i \in I$, let α_i be an execution of A_i. Suppose β is a sequence of actions in $ext(A)$ such that $\beta|A_i = beh(\alpha_i)$ for every i. Then there is an execution α of A such that $\beta = beh(\alpha)$ and $\alpha_i = \alpha|A_i$ for all i. Moreover, if α_i is a fair execution of A_i for all i, then α may be taken to be a fair execution of A.*

Similarly, schedules or behaviors of component automata can be patched together to form schedules or behaviors of the composition.

Lemma 2.4 *Let $\{A_i\}_{i\in I}$ be a strongly compatible collection of automata, and let $A = \Pi_{i\in I} A_i$. Let β be a sequence of actions in $acts(A)$. If $\beta|A_i \in scheds(A_i)$ for all $i \in I$, then $\beta \in scheds(A)$. Moreover, the same result holds for fairscheds, behs and fairbehs in place of scheds.*

2.6 Hiding Output Actions

We now define an operator that hides a designated set of output actions in a given automaton to produce a new automaton in which the given actions are internal. Namely, suppose A is an I/O automaton and $\Phi \subseteq ext(A)$ is any subset of the output actions of A. Then we define a new automaton, $hide_\Phi(A)$ to be exactly the same as A except for its signature component. For the signature component, we have $in(hide_\Phi(A)) = in(A)$, $out(hide_\Phi(A)) = out(A) \setminus \Phi$, and $int(hide_\Phi(A)) = int(A) \cup \Phi$.

3 The Physical Layer

The physical layer is the lowest layer in the OSI Reference Model hierarchy, and is implemented directly in terms of the physical transmission media. A standard interface to the physical layer permits implementation of the higher layers independently of the transmission media.

In a typical setting, a physical layer interacts with higher layers at two endpoints, a "transmitting station" and a "receiving station". The physical layer receives messages called "packets" from the higher layer at the transmitting station, and delivers some of the packets to the higher layer at the receiving station. The physical layer can lose packets. While it is also possible for packets to be corrupted by the transmission medium, we assume that the physical layer masks such corrupted packets using error-detecting codes. Thus, the only faulty behavior we consider is loss of packets.

In this section, we give specifications for physical layer behavior. We will specify two different kinds of physical layers, based on whether or not the channel is required to ensure FIFO delivery of packets. It is convenient to parameterize the specifications by an ordered pair (t, r) of names for the transmitting and receiving stations. The specifications will be given as schedule modules, denoted by the names $PL\text{-}FIFO^{t,r}$ and $PL^{t,r}$ respectively.

Let P be a fixed alphabet of "packets". Both $PL^{t,r}$ and $PL\text{-}FIFO^{t,r}$ have the action signature illustrated in Figure 1 and given formally as follows.

Input actions:
- $send\_pkt^{t,r}(p)$, $p \in P$
- $wake^{t,r}$
- $fail^{t,r}$
- $crash^{t,r}$

Output actions:
- $receive\_pkt^{t,r}(p)$, $p \in P$

There are no internal actions. The $send\_pkt^{t,r}(p)$ action represents the sending of packet p on the physical channel by the transmitting station, and the $receive\_pkt^{t,r}(p)$ represents the receipt of packet p by the receiving station. The $wake^{t,r}$ and $fail^{t,r}$ actions represent notification that the transmission medium has become active or inactive, respectively. Finally, the $crash^{t,r}$ action represents notification

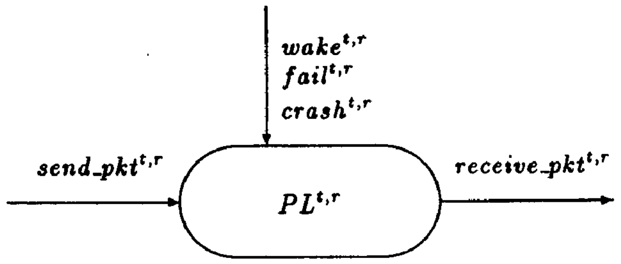

Figure 1: The Physical Layer

that the transmitting station has suffered a hardware crash failure. We will often refer to the actions in $acts(PL^{t,r})$ as *physical layer actions* (for (t, r)).

In order to define the sets of schedules for the two schedule modules, $scheds(PL^{t,r})$ and $scheds(PL\text{-}FIFO^{t,r})$, it is helpful to define a collection of auxiliary properties of sequences of physical layer actions. These will be properties reflecting the operation of a "good" physical channel in a "good" environment. We will then specify the allowed behaviors of a physical channel by requiring some of these properties to hold if others do. Let $\beta = \pi_1\pi_2...$ be a (finite or infinite) sequence of physical layer actions. We define properties for β.

We define a *crash interval* in β to be a maximal contiguous subsequence not containing a $crash^{t,r}$ event. We say that β is *well-formed* provided that in every crash interval in β, the $fail^{t,r}$ and $wake^{t,r}$ events alternate strictly, starting with $wake^{t,r}$. Thus, in a well-formed sequence, there are repeated alternating notifications that the transmission medium is active and inactive, with crashes serving as delimiters between sequences of wake and fail events. A crash event can be thought of as including a failure, in cases where the crash follows a wake with no intervening fail.

If β is a well-formed sequence of physical layer actions, then a *working interval* in β is the subsequence of β from any $wake^{t,r}$ event until the next $fail^{t,r}$ or $crash^{t,r}$ event, or until the end of β if there are no later $crash^{t,r}$ or $fail^{t,r}$ events, not including the given $wake^{t,r}$, $fail^{t,r}$ or $crash^{t,r}$ events. If β has a $wake^{t,r}$ event with no later $fail^{t,r}$ or $crash^{t,r}$ event, then the suffix of β starting after the $wake^{t,r}$ event is called an *unbounded working interval*. Note that there is at most one unbounded working interval in β.

Now we define the following properties, (PL1)-(PL6), of well-formed sequences β of physical layer actions. The first property is a restriction on the use of the physical channel saying that a packet is sent only when the channel is active.

(PL1) Every $send\_pkt^{t,r}$ event occurs in a working interval in β.

The next property is a technical restriction on the use of the physical channel saying that the packets sent are always unique. Thus the reader may think of each packet as labeled with a unique identifier; however, a practical data link layer protocol should not use this label, which is included in the model for ease of analysis, but does not correspond to any bits sent on the transmission medium.[4] The main reason we use this restriction is so that we can easily establish a correspondence between the packets sent and the packets received on the channel.

(PL2) For every packet p, there is at most one $send\_pkt^{t,r}(p)$ event in β.

The next property asserts that no single packet is received more than once.

(PL3) For every packet p, there is at most one $receive\_pkt^{t,r}(p)$ event in β.

The next property says that the physical layer only delivers packets that were previously sent.

(PL4) For every $receive\_pkt^{t,r}(p)$ event in β, there is a preceding $send\_pkt^{t,r}(p)$ event in β.

The next is the FIFO property. It says that those packets that are delivered have their *receive_pkt* events occurring in the same order as their *send_pkt* events. Note that (PL5) may be true even if a packet is delivered and some packet sent earlier is not delivered; there can be gaps in the sequence of delivered packets representing lost packets.

(PL5) (FIFO) Suppose that p and p' are two packets such that the events $\pi_{i_1} = send\_pkt^{t,r}(p)$, $\pi_{i_2} = receive\_pkt^{t,r}(p)$, $\pi_{i_3} = receive\_pkt^{t,r}(p')$ and $\pi_{i_4} = receive\_pkt^{t,r}(p')$ appear in β. Then $i_1 < i_3$ if and only if $i_2 < i_4$.

[4]In Section 5, we model formally the "header", the information in a packet that is used by a data link layer protocol, as an equivalence class to which the packet belongs.

So far, all of the properties listed have been safety properties. The final property is a liveness property. It says that if a channel remains active and repeated send events occur, then eventually some packet is delivered.

(PL6) Starting after any point in an unbounded working interval, if infinitely many $send\_pkt^{t,r}$ events occur after that point, then some $receive\_pkt^{t,r}$ event occurs after that point.

Notice that well-formedness, (PL1) and (PL2) are properties that can be guaranteed by the environment that supplies inputs to the physical channel, while (PL3)-(PL6) are properties that the channel itself can enforce. However, we only ask the physical channel to enforce them when the environment plays its part, by providing inputs that ensure well-formedness, (PL1) and (PL2). If the environment violates the input conditions, e.g., if send events happen outside of working intervals, then the specification does not constrain the behavior of the physical channel. Formally, we define the two schedule modules $PL^{t,r}$ and $PL\text{-}FIFO^{t,r}$. We have already defined $sig(PL^{t,r})$ and $sig(PL\text{-}FIFO^{t,r})$. Let $scheds(PL^{t,r})$ be the set of sequences β of physical layer actions satisfying the condition "if β is well-formed and satisfies (PL1) and (PL2) then β satisfies (PL3), (PL4) and (PL6)". Similarly, let $scheds(PL\text{-}FIFO^{t,r})$ be the set of sequences β of physical layer actions satisfying the condition "if β is well-formed and satisfies (PL1) and (PL2) then β satisfies (PL3), (PL4), the FIFO condition (PL5), and (PL6)".

A *physical channel* from t to r is any I/O automaton that solves $PL^{t,r}$. A *FIFO physical channel* from t to r is any I/O automaton that solves $PL\text{-}FIFO^{t,r}$.

In a "real-world" implementation of a physical channel using a physical transmission medium, (PL6) would not be guaranteed with absolutely certainty, but rather with extremely high probability. It seems that the probability could be sufficiently high, however, to justify our decision to ignore in the formal model the small likelihood that no packets ever get delivered on an active channel.

4 The Data Link Layer

The data link layer is the second lowest layer in the hierarchy, and is implemented using the services of the physical layer. Generally, it is implemented in terms of two physical channels, one in each direction. It provides a reliable one-hop message delivery service, which can in turn be used by the next higher layer.

We again assume that there are two endpoints, a "transmitting station" and a "receiving station". The data link layer receives messages from the higher layer at the transmitting station, and delivers them at the receiving station. The data link layer guarantees that every message that is sent is eventually received, assuming that the underlying transmission medium remains active. Furthermore, the order of the messages is preserved.

In this section, we give a specification for data link layer behavior, as a parameterized schedule module $DL^{t,r}$. Let M be a fixed infinite alphabet of "messages". The action signature $sig(DL^{t,r})$ is illustrated in Figure 2, and is given formally as follows.

Input actions:
- $send\_msg^{t,r}(m)$, $m \in M$
- $wake^{t,r}$
- $fail^{t,r}$
- $crash^{t,r}$
- $wake^{r,t}$
- $fail^{r,t}$
- $crash^{r,t}$

Output actions:
- $receive\_msg^{t,r}(m)$, $m \in M$

There are no internal actions. The $send\_msg^{t,r}(m)$ action represents the sending of message m on the data link by the transmitting station, and the $receive\_msg^{t,r}(m)$ represents the receipt of message m by the receiving station. The $wake^{t,r}$ and $fail^{t,r}$ actions represent notification that the transmission medium in the direction from t to r has become active or inactive, respectively, while the $wake^{r,t}$ and $fail^{r,t}$ actions represent similar notification for the transmission medium in the direction from r to t. The $crash^{t,r}$ and $crash^{r,t}$ actions represent notification that the transmitting or receiving station, respectively, has suffered a hardware crash failure. We will often refer to the actions in $acts(DL^{t,r})$ as *data link layer actions.*

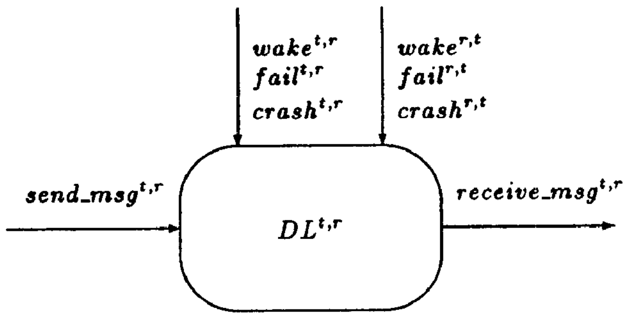

Figure 2: The Data Link Layer

In order to define the set $scheds(DL^{t,r})$, we define a collection of auxiliary properties of sequences of data link layer actions. Let $\beta = \pi_1\pi_2...$ be a (finite or infinite) sequence of data link layer actions. We define properties for β.

We define a *transmitter crash interval* in β to be a maximal contiguous subsequence not containing a $crash^{t,r}$ event, and similarly a *receiver crash interval* in β to be a maximal contiguous subsequence not containing a $crash^{r,t}$ event. We say that β is *well-formed* provided that the following two conditions hold. First, in any transmitter crash interval in β, the $fail^{t,r}$ and $wake^{t,r}$ events alternate strictly, starting with $wake^{t,r}$. Second, in any receiver crash interval in β the $fail^{r,t}$ and $wake^{r,t}$ events alternate strictly, starting with $wake^{r,t}$. Thus, for each direction of the underlying transmission medium, there are repeated alternating notifications that the transmission medium is active and inactive, with crashes serving as delimiters between sequences of wake and fail events.

If β is a well-formed sequence of data link layer actions, then a *transmitter working interval* in β is the subsequence of β from any $wake^{t,r}$ event until the next $fail^{t,r}$ or $crash^{t,r}$ event, or until the end of β if there are no later $fail^{t,r}$ or $crash^{t,r}$ events, not including the given $wake^{t,r}$, $fail^{t,r}$ or $crash^{t,r}$ events. If β has a $wake^{t,r}$ event with no later $fail^{t,r}$ or $crash^{t,r}$ event, then the suffix of β starting after the $wake^{t,r}$ event is called an *unbounded transmitter working interval.* We give analogous definitions for *receiver working interval* and *unbounded receiver* *working interval.*

Now we define the following properties, (DL1)-(DL7), of well-formed sequences β of data link layer actions. The first property says that there is eventual consistency in the notifications that occur at both ends of the link, about the status of the underlying transmission medium. That this property holds is a reasonable assumption, for example, in the usual case where the same hardware is used for the transmission medium in both directions.

(DL1) There is an unbounded transmitter working interval in β if and only if there is an unbounded receiver working interval in β.

The next five properties are analogous to properties already defined for the physical layer.

(DL2) Every $send\_msg^{t,r}$ event occurs in a transmitter working interval in β.

(DL3) For every message m, there is an most one $send\_msg^{t,r}(m)$ event in β.

(DL4) For every message m, there is an most one $receive\_msg^{t,r}(m)$ event in β.

(DL5) For every $receive\_msg^{t,r}(m)$ event in β, there is a preceding $send\_msg^{t,r}(m)$ event in β.

(DL6) (FIFO)
Suppose that m and m' are two messages such that the events $\pi_{i_1} = send\_msg^{t,r}(m)$, $\pi_{i_2} = receive\_msg^{t,r}(m)$, $\pi_{i_3} = send\_msg^{t,r}(m')$ and $\pi_{i_4} = receive\_msg^{t,r}(m')$ appear in β. Then $i_1 < i_3$ if and only if $i_2 < i_4$.

The remaining two properties describe ways in which the data link layer makes stronger guarantees than does the physical layer. The first of these says that the data link layer does not lose some messages but deliver later messages, within a single transmitter working interval.

(DL7) Suppose that $\pi_i = send\_msg^{t,r}(m)$ and $\pi_j = send\_msg^{t,r}(m')$ appear in the same transmitter working interval in β and $i < j$. If a $receive\_msg^{t,r}(m')$ event appears in β, then a $receive\_msg^{t,r}(m)$ also appears in β.

Finally, we have the data link layer liveness property. It says that all messages that are sent are delivered eventually, provided the link remains active. This property expresses the reliability of the message delivery guaranteed by the data link layer.

(DL8) If a $send\_msg^{t,r}(m)$ event occurs in an unbounded transmitter working interval in β, then there is a $receive\_msg^{t,r}(m)$ event in β.

Now we can define the schedule module $DL^{t,r}$. We have already defined $sig(DL^{t,r})$. Let $scheds(DL^{t,r})$ be the set of sequences β of data link layer actions satisfying the condition "if β is well-formed and satisfies (DL1)-(DL3) then β satisfies (DL4)-(DL8)".

Although the schedule module $DL^{t,r}$ represents the behavior one would require from an interesting data link layer, it is useful for us to define another schedule module $WDL^{t,r}$ representing weaker requirements on data link behavior. Thus, let $sig(WDL^{t,r}) = sig(DL^{t,r})$, and let $scheds(WDL^{t,r})$ be the set of sequences β of data link layer actions satisfying the condition "if β is well-formed and satisfies (DL1)-(DL3) then β satisfies (DL4), (DL5) and (DL8)".

Although this weaker specification is less interesting than $DL^{t,r}$ for describing properties of a useful data link layer, it is adequate for proving our impossibility results. It is easy to see that $WDL^{t,r}$ is a weaker specification than $DL^{t,r}$, i.e., that $scheds(DL^{t,r}) \subseteq scheds(WDL^{t,r})$. Thus, any automaton that solves $DL^{t,r}$ also solves $scheds(WDL^{t,r})$, so that the impossibility results we obtain for solving $WDL^{t,r}$ immediately imply corresponding impossibility results for solving $DL^{t,r}$.

We next prove a simple lemma which will be useful later. In the proof of this lemma we illustrate the way properties such as (DL1)-(DL8) and the basic facts about the I/O automaton model can be used to show the existence of fair behaviors of an automaton that solves the specification for a data link layer.

Lemma 4.1 *Let A be any automaton that solves $WDL^{t,r}$, and let $m \in M$. Then there is a fair schedule $\beta = \pi_1\pi_2\ldots$ of A such that $beh(\beta) = wake^{t,r}wake^{r,t}send\_msg^{t,r}(m)receive\_msg^{t,r}(m)$, $\pi_1 = wake^{t,r}$ and $\pi_2 = wake^{r,t}$.*

Proof: Since the *wake* actions are inputs of A, the sequence $\gamma = wake^{t,r}wake^{r,t}send\_msg^{t,r}(m)$

is a finite schedule of A. By Lemma 2.1, there is a fair schedule β of A that extends γ and that includes no input events of A except those in γ. We claim that $beh(\beta)$ must be the sequence $wake^{t,r} wake^{r,t} send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$.

First, note that $beh(\beta)$ is well-formed and satisfies (DL1), (DL2) and (DL3), since $beh(\gamma)$ has these properties and they are only depend on the sequence of inputs to A. Since A solves $WDL^{t,r}$, $beh(\beta)$ also satisfies (DL4), (DL5) and (DL8). Since $beh(\beta)$ only extends $beh(\gamma)$ with output actions, only $receive\_pkt^{t,r}$ actions appear in the suffix.

Since the action $send\_msg^{t,r}(m)$ occurs in an unbounded transmitter working interval in β, property (DL8) implies that the action $receive\_msg^{t,r}(m)$ appears in β. Then (DL4) and (DL5) imply that $receive\_msg^{t,r}(m)$ can only appear once, and that no other $receive\_msg^{t,r}$ event can appear. It follows that $beh(\beta)$ is $wake^{t,r} wake^{r,t} send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$.
□

5 Data Link Implementation

In this section, we define a "data link protocol", which is intended to be used to implement the data link layer using the services provided by the physical layer. A data link protocol consists of two automata, one at the transmitting station and one at the receiving station. These automata communicate with each other using two physical channels, one in each direction. They also communicate with the outside world, through the data link layer actions we defined in the previous section.

Figure 3 shows how two protocol automata and two physical channels should be connected, in a data link implementation.

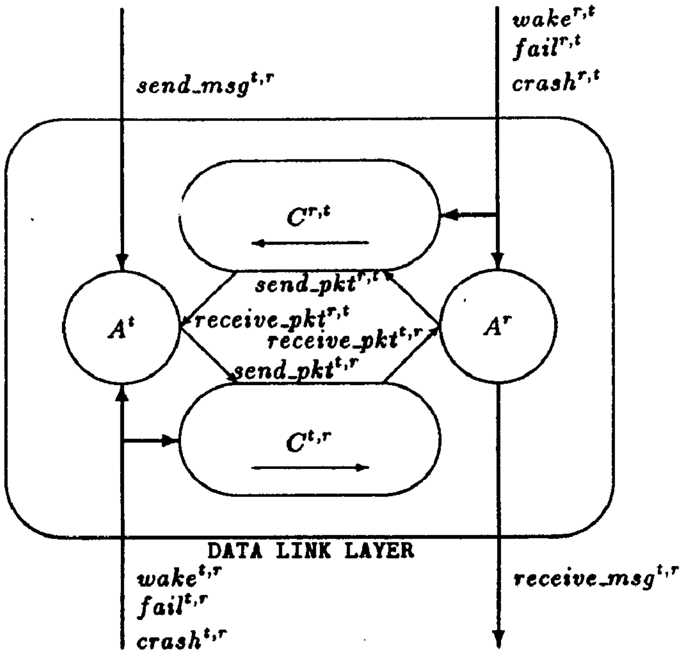

Figure 3: A Data Link Implementation

5.1 Data Link Protocols

Let t and r again be names (for the transmitting and receiving station respectively). Then a ***transmitting automaton*** for (t, r) is any I/O automaton having the following external action signature.

Input actions:
  $send\_msg^{t,r}(m)$, $m \in M$
  $receive\_pkt^{r,t}(p)$, $p \in P$
  $wake^{t,r}$
  $fail^{t,r}$
  $crash^{t,r}$
Output actions:
  $send\_pkt^{t,r}(p)$, $p \in P$

In addition, there can be any number of internal actions. That is, a transmitting automaton receives requests from the environment of the data link layer to send messages to the receiving station r. It also receives packets over the physical channel from r. Moreover, it receives notification of the status of the physical channel from t to r, and notification of crashes at the transmitting station. It sends packets to r over the physical channel to r.

Similarly, a ***receiving automaton*** for (t, r) is any I/O automaton having the following external signature.

Input actions:
  $receive\_pkt^{t,r}(p)$, $p \in P$
  $wake^{r,t}$
  $fail^{r,t}$
  $crash^{r,t}$
Output actions:
  $send\_pkt^{r,t}(p)$, $p \in P$

$receive\_msg^{t,r}(m)$, $m \in M$

Again, there can also be any number of internal actions. That is, a receiving automaton receives packets over the physical channel from t. Moreover, it receives notification of the status of the physical channel from r to t, and notification of crashes at the receiving station. It sends packets to t over the physical channel to t, and it delivers messages to the environment of the data link layer.

A *data link protocol* is a pair (A^t, A^r), where A^t is a transmitting automaton and A^r is a receiving automaton.

5.2 Correctness of Data Link Protocols

Now we are ready to define correctness of data link protocols. Informally, we say that a data link protocol is "correct" provided that when it is composed with any "correct physical layer" (i.e. a pair of physical channels from t to r and from r to t, respectively), the resulting system yields correct data link layer behavior. This reflects the fundamental idea of layering, that the implementation of one layer should not depend on the details of the implementation of other layers, so that each layer can be implemented and maintained independently. Formally, we say that a data link protocol (A^t, A^r) is *correct* provided that the following is true. For all $C^{t,r}$ and $C^{r,t}$ that are physical channels from t to r and from r to t, respectively, $hide_\Phi(D)$ solves $DL^{t,r}$, where D is the composition of A^t, A^r, $C^{t,r}$ and $C^{r,t}$, and Φ is the subset of acts(D) consisting of *send_pkt* and *receive_pkt* actions.

As mentioned earlier, our impossibility results can be proved for weaker data link requirements. Thus we also define *weak correctness* for data link protocols. This is defined exactly as for correctness, except that $hide_\Phi(D)$ is required to solve $WDL^{t,r}$ instead of $DL^{t,r}$. Obviously, any correct data link protocol is also weakly correct.

We also define what it means for a data link protocol to be *correct with respect to FIFO physical channels*; again, this is defined exactly as for correctness except that $C^{t,r}$ and $C^{r,t}$ are restricted to range over only FIFO physical channels from t to r and from r to t, respectively, rather than over arbitrary physical channels. Finally, we define a notion of *weak correctness with respect to FIFO physical channels*, for data link protocols. This is defined exactly as for correctness with respect to FIFO physical channels, except that $hide_\Phi(D)$ is required to solve $WDL^{t,r}$ instead of $DL^{t,r}$.

Obviously, any data link protocol that is correct with respect to FIFO physical channels is also weakly correct with respect to FIFO physical channels. Also, any data link protocol that is correct (resp. weakly correct) is also correct (resp. weakly correct) with respect to FIFO physical channels.

5.3 Constraints on Data Link Protocols

In this subsection, we define several constraints we wish to consider for data link protocols.

5.3.1 Message-Independence

Most data link protocols in the literature are "message-independent" in the sense that the processing done by the protocols does not depend on the contents of messages submitted by the environment. The data link protocol might break up a message into packets, and might construct header information to add to packets, but does not typically carry out drastically different processing based on the specific contents of messages. This is often expressed by saying that the data link layer treats messages (which in fact are usually structured, including, for example, headers from higher layer protocols) as uninterpreted data.

We model message-independence as follows. Let $A = (A^t, A^r)$ be a data link protocol. Let $\equiv$ be an equivalence relation on the domain $M \cup P \cup states(A^t) \cup states(A^r) \cup acts(A^t) \cup acts(A^r)$. Then A is said to be *message-independent* with respect to the equivalence relation $\equiv$ provided that the following conditions hold.

1. $\equiv$ only relates elements of the same kind, i.e., elements of M, or P, or $states(A^t)$, etc. Also, a start state cannot be related to a non-start state. Moreover, if $a \equiv a'$ for two actions a and a', then a and a' are identical except possibly for a difference in their message or packet parameter.
2. For each pair m, m' of messages, $m \equiv m'$, $send\_msg^{t,r}(m) \equiv send\_msg^{t,r}(m')$, and

$receive\_msg^{t,r}(m) \equiv receive\_msg^{t,r}(m')$.

3. For each pair p,p' of packets, $send\_pkt^{t,r}(p) \equiv send\_pkt^{t,r}(p')$ if and only if $p \equiv p'$, $receive\_pkt^{t,r}(p) \equiv receive\_pkt^{t,r}(p')$ if and only if $p \equiv p'$, $send\_pkt^{r,t}(p) \equiv send\_pkt^{r,t}(p')$ if and only if $p \equiv p'$, and $receive\_pkt^{r,t}(p) \equiv receive\_pkt^{r,t}(p')$ if and only if $p \equiv p'$.
4. For every two states q and q' with $q \equiv q'$, if action a is enabled in q then there is an action a' with $a \equiv a'$, such that a' is enabled in q'.
5. Suppose that $q \equiv q'$ and $a \equiv a'$, where action a is enabled in state q and action a' is enabled in state q'. If r is a state such that (q, a, r) is a step, then there exists a state r' such that $r \equiv r'$ and (q', a', r') is a step.

We say that data link protocol A is *message-independent* provided that it is message-independent with respect to some equivalence relation.

For a data link protocol, A, that is message-independent with respect to an equivalence relation $\equiv$, we define the set $headers(A, \equiv)$ to be the set of equivalence classes of packets. Since all the packets in a given equivalence class are treated in equivalent ways by the protocol, we can think of them as modelling the set of packets that contain a particular pattern of bits in the data link layer header. We say that A has *bounded headers* if $headers(A, \equiv)$ is a finite set.

Two sequences, $x = x_1x_2\ldots$ and $y = y_1y_2\ldots$, are said to be *equivalent* with respect to $\equiv$ if $|x| = |y|$ and for every i, $x_i \equiv y_i$.

5.3.2 Crashing

Here, we describe a "crashing" property, which says that a crash at either the transmitting or receiving station is able to cause the corresponding protocol automaton to revert back to its start state (thereby losing all processing information in its memory).

We say that a transmitting automaton A is *crashing* provided that there is a unique start state q_0 and $(q, crash^{t,r}, q_0)$ is a step of A, for every $q \in states(A)$. Similarly, we say that a receiving automaton A is *crashing* provided that there is a unique start state q_0 and $(q, crash^{r,t}, q_0)$ is a step of A, for every $q \in states(A)$. A data link protocol (A^t, A^r) is said to be *crashing* provided that A^t and A^r are both crashing.

6 Specific Physical Channels

Since the correctness of a data link protocol requires that it work when composed with any physical channels, we are able to prove the impossibility of a correct protocol satisfying certain requirements by merely demonstrating that no such protocol works when combined with a specific pair of physical channels. In this section we introduce the channels we will use. First we introduce a very permissive physical channel, which we will use in Section 8. Then we will introduce a closely related FIFO physical channel, which we will use in Section 7.

6.1 A Permissive Physical Channel

We begin by defining a particular "very permissive" physical channel. This channel can even be considered to be a "universal physical channel", in the sense of Lemma 6.2 below. This channel is not FIFO, and in Section 8 we will use it to prove that unbounded headers are needed in a protocol that uses this channel.

First, we define a set S of ordered pairs (i, j) of positive integers to be a *delivery set* provided that it satisfies the following two conditions: for each positive integer j, S includes a unique element (i, j), and for each positive integer i, it includes at most one element (i, j).

The state of the physical channel $\bar{C}^{t,r}$ has two counters, $counter_1$ and $counter_2$, an infinite delivery set S of pairs of non-negative integers, and a partial mapping $packet$ from the set of positive integers to P. The counter $counter_1$ represents the number of $send\_pkt^{t,r}$ actions, and $counter_2$ represents the number of $receive\_pkt^{t,r}$ actions, that have occurred so far. The set S determines which packets are delivered, and in what order – it contains pairs (i, j) that correlate the j-th $receive\_pkt^{t,r}$ event with the i-th $send\_pkt^{t,r}$ event. Thus the restrictions in the definition of a delivery set correspond to the requirements that a packet should not be delivered unless it was sent, and that each packet should not be delivered more than once. The mapping $packet$ associates with an integer i the packet that was sent in the i-th $send\_pkt^{t,r}$ event. Initially $counter_1$ and $counter_2$ are zero and $packet$ is undefined everywhere. The set S is initialized to an arbitrary delivery set (and remains fixed).

When a $send\_pkt^{t,r}(p)$ action occurs, the counter

counter$_1$ is incremented by one and $packet(i)$ is set to p, where i is the new value of *counter*$_1$. The precondition of $receive\_pkt^{t,r}(p)$ is that there exists i such that $packet(i) = p$ and $(i, counter_2 + 1) \in S$. The effect is to increment *counter*$_2$ by one. The *fail*, *wake* and *crash* actions have no effect. The partition puts all the output actions in a single class. We define the physical channel $\bar{C}^{r,t}$ analogously.

For $x \in \{t, r\}$ we define $\bar{x}$ so that $\bar{x} \in \{t, r\}$ and $x \neq \bar{x}$.

Lemma 6.1 *The automaton* $\bar{C}^{x,\bar{x}}$ *is a physical channel.*

Proof: We must show that $fairbehs(\bar{C}^{x,\bar{x}}) \subseteq scheds(PL^{x,\bar{x}})$. Let β be a fair behavior of $\bar{C}^{x,\bar{x}}$. If β is either not well-formed or does not satisfy (PL1) or (PL2) then it is a schedule of $PL^{x,\bar{x}}$, since there are no constraints on such schedules. So suppose β is well-formed and satisfies (PL1) and (PL2).

Suppose that (PL3) does not hold, i.e. there is a packet p for which two $receive\_pkt^{x,\bar{x}}(p)$ events occur in β. Let j_1 and j_2 denote the number of $receive\_pkt^{x,\bar{x}}$ events up to and including the first and second $receive\_pkt^{x,\bar{x}}(p)$ respectively. The precondition of $receive\_pkt^{x,\bar{x}}(p)$ implies that there are i_1 and i_2 such that $(i_1, j_1), (i_2, j_2) \in S$ and the i_1-th and i_2-th $send\_pkt^{x,\bar{x}}$ events are both $send\_pkt^{x,\bar{x}}(p)$. Since S is a delivery set, $i_1 \neq i_2$. This contradicts the assumption that β satisfies (PL2). Therefore, (PL3) is satisfied.

One
of the preconditions of the j-th $receive\_pkt^{x,\bar{x}}(p)$ is that there exists i such that $packet(i) = p$. Thus the i-th $send\_pkt^{x,\bar{x}}$ event in β is $send\_pkt^{x,\bar{x}}(p)$. Also, the $receive\_pkt^{x,\bar{x}}(p)$ occurs after $packet(i)$ is defined, i.e. after the $send\_pkt^{x,\bar{x}}(p)$ event. This implies that (PL4) is satisfied.

Suppose that β has an unbounded working interval, and fix a point in that interval just after, say, the k-th event in β. Suppose that infinitely many $send\_pkt^{x,\bar{x}}$ events occur after the given point. Let j be the number of $receive\_pkt^{x,\bar{x}}$ events in β up to the given point. Since S is a delivery set, there exists i such that $(i, j+1) \in S$. Let p be the packet appearing in the ith $send\_pkt^{x,\bar{x}}$ event in β. Then the precondition of $receive\_pkt^{x,\bar{x}}(p)$ eventually becomes true, and stays true until the action occurs. Thus, $receive\_pkt^{x,\bar{x}}(p)$ appears in β, sometime after the k-th event. Therefore, β satisfies (PL6). □

The following lemma shows that $\bar{C}^{x,\bar{x}}$ has among its behaviors all of the "sensible" failure-free schedules of the specification $PL^{x,\bar{x}}$.

Lemma 6.2 *Suppose* β *is in* $scheds(PL^{x,\bar{x}})$, *and* β *is well-formed, satisfies (PL1) and (PL2), and contains no* $fail^{x,\bar{x}}$ *or* $crash^{x,\bar{x}}$ *events. Then* $\beta \in fairbehs(\bar{C}^{x,\bar{x}})$.

We can combine the permissive physical channels with an arbitrary data link protocol, as follows. If A is a data link protocol, then let $\bar{D}(A)$ be the composition of A^t, A^r, $\bar{C}^{t,r}$ and $\bar{C}^{r,t}$. Also let $\bar{D}'(A) = hide_{\Phi}(\bar{D}(A))$, where Φ is the subset of $acts(\bar{D}(A))$ consisting of *send_pkt* and *receive_pkt* actions.

6.2 A Permissive FIFO Physical Channel

We also define a particular permissive FIFO physical channel, which we will use in the argument of Section 7. We define $\hat{C}^{t,r}$ to be identical to $\bar{C}^{t,r}$ except that the start states are restricted to be those in which the delivery set S is monotone, that is, there are no pairs (i_1, j_1) and (i_2, j_2) in S with $i_1 < i_2$ and $j_1 \geq j_2$. Similarly, we define $\hat{C}^{r,t}$.

Since every finite (resp. fair) execution of $\hat{C}^{t,r}$ is also a finite (resp. fair) execution of $\bar{C}^{t,r}$ we see that $\hat{C}^{t,r}$ is a physical channel. The restriction on the delivery set ensures that it is a FIFO physical channel.

If $A = (A^t, A^r)$ is a data link protocol, let $\hat{D}(A)$ be the composition of A^t, A^r, $\hat{C}^{t,r}$ and $\hat{C}^{r,t}$. Also let $\hat{D}'(A) = hide_{\Phi}(\hat{D}(A))$ where Φ is the subset of $acts(\hat{D}(A))$ consisting of *send_pkt* and *receive_pkt* actions.

6.3 Properties of the Permissive Physical Channels

We collect here some simple properties of the channels just defined, for use in Sections 7 and 8.

We begin this subsection with a useful definition. Namely, we define a partial function $del(S, (i, j))$ that takes a delivery set S and a pair $(i, j) \in S$, and returns a new delivery set S'. The new set S' represents the result of deleting the given pair from the set, and is defined as follows. (1) For every $j' < j$, $(i', j') \in S'$ iff $(i', j') \in S$. (2) $(i, j) \notin S'$.

(3) For every $j' > j$, $(i', j') \in S'$ iff $(i', j'+1) \in S$. We extend the function *del* so that its second argument is any finite subset of S rather than just a single pair, in the natural way: $del(S, X \cup \{(i,j)\}) = del(del(S,X),(i,j))$. Notice that if S is a monotone delivery set, so is $del(S,X)$.

We say a state of $\bar{C}^{x,\bar{x}}$ or $\hat{C}^{x,\bar{x}}$ is *clean* if (i) S does not contain any pair (i,j) with $i \leq counter_1$ and $j > counter_2$, and (ii) S contains $(counter_1 + k, counter_2 + k)$ for all $k > 0$. The intuition is that the channel is empty, and from now on will act FIFO with no losses. The next lemma is proved by altering the delivery set without changing those pairs (i,j) with $j \leq counter_2$.

Lemma 6.3 *If β is a schedule of $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) then there is a state s of $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) such that β can leave $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) in s and s is clean.*

If s is a state of $\bar{C}^{x,\bar{x}}$ or $\hat{C}^{x,\bar{x}}$, we say that a sequence of packets $Q = q_1 q_2 \ldots q_k$ is *waiting* in a state s if for all l such that $1 \leq l \leq k$ there is an integer i_l such that $packet(i_l) = q_l$ and $(i_l, counter_2 + l) \in S$ in s.

We have the fundamental property that a channel can deliver a sequence of packets that are waiting in its state.

Lemma 6.4 *Let s be a state of $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) and $Q = q_1 q_2 \ldots q_k$ a sequence of packets such that Q is waiting in s. Then there is an execution fragment starting with state s with schedule $receive\_pkt^{x,\bar{x}}(q_1) \ldots receive\_pkt^{x,\bar{x}}(q_k)$.*

We now give a lemma that shows that certain schedules can leave a channel in a state where packets are waiting.

Lemma 6.5 *If β is a schedule of $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) and γ is a sequence of input actions of $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) such that $Q = q_1 q_2 \cdots q_n$ is the sequence of packets sent in γ, then $\beta\gamma$ is a schedule of $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) that can leave $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) in a state in which Q is waiting.*

By surgery on S (using the del function) we obtain the following lemma which expresses the ability of the channels to lose any packets that have not been delivered.

Lemma 6.6 *If β is a schedule of $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) that can leave $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) in a state s in which Q is waiting, and Q' is a subsequence of Q, then there is a state s' such that β can leave $\bar{C}^{x,\bar{x}}$ (resp. $\hat{C}^{x,\bar{x}}$) in s' and Q' is waiting in s'.*

We have an extra result for the non-FIFO channels. We say that a packet p is *in transit* from x to $\bar{x}$ in a sequence β of actions provided that $send\_pkt^{x,\bar{x}}(p)$ occurs in β and $receive\_pkt^{x,\bar{x}}(p)$ does not occur in β. We have the result that any sequence of packets in transit can be waiting in the channel.

Lemma 6.7 *Let β be a schedule of $\bar{C}^{x,\bar{x}}$, and Q a sequence of distinct packets. If each packet in the sequence is in transit from x to $\bar{x}$ in β, then β can leave $\bar{C}^{x,\bar{x}}$ in a state s such that Q is waiting in s.*

7 Tolerating Host Crashes

In a data link protocol a useful property would be the ability of the protocol to tolerate a host crash. A host crash causes all the memory at the host to be lost. (In our model this is reflected by setting the state of the automaton in that host to its distinguished initial state.) Baratz and Segall [BS83] conjectured that no such protocol is possible. The link initialization protocol of [BS83] cannot tolerate host crashes as we have defined them. However if there is a single non-volatile bit (a bit that is not reset during the host crash) the [BS83] protocol is correct. We prove that no message-independent data link protocol can tolerate arbitrary host crashes (without access to non-volatile memory).

The essense of our proof is to take a data link protocol that is alleged to be crashing, message-independent and weakly correct, and to find two executions of the system that leave the transmitting and receiving automata in equivalent states, although in one every message has been delivered and in the other there is an undelivered message. The protocol must eventually deliver the missing message in any fair extension of the second execution, even if no more inputs arrive from the environment. An equivalent extension of the first execution will cause some message to be delivered, although every message sent had already been delivered. This contradicts the claimed correctness of the protocol.

Recall that for $x \in \{t, r\}$ we define $\bar{x}$ so that $\bar{x} \in \{t, r\}$ and $x \neq \bar{x}$, and we define $\hat{D}'(A)$ to be the result of composing data link protocol A with the permissive FIFO physical channels $\hat{C}^{t,r}$ and $\hat{C}^{r,t}$ and then hiding the sending and receiving of packets. For $\alpha = s_0\pi_1 s_1 \ldots \pi_n s_n$ a finite execution of $\hat{D}'(A)$ and k an integer with $0 \leq k \leq n$ let us define the following: $in_A(\alpha, x, k)$ is the sequence of packets received by A^x during the first k steps of α; $out_A(\alpha, x, k)$ is the sequence of packets sent by A^x during the first k steps of α; $state_A(\alpha, x, k)$ is the state of A^x in s_k; $acts_A(\alpha, x, k)$ is the sequence of actions of A^x during the first k steps of α.

We now state the main lemmas we will use to prove the result of this section.

The first lemma shows that one can modify the suffix of an extension of one execution to give an extension of another, if the two executions end with the data link protocol automata in equivalent states. This modification may alter states and actions, but only into equivalent states and actions. This lemma can be proved by an easy induction on j, using the definition of message-independence.

Lemma 7.1 *Let $A = (A^t, A^r)$ be a message-independent data link protocol. Let $\alpha = s_0\pi_1 s_1 \ldots \pi_n s_n$ and $\hat{\alpha} = \hat{s}_0\hat{\pi}_1\hat{s}_1 \ldots \hat{\pi}_k\hat{s}_k$ be finite executions of $\hat{D}'(A)$ with the following properties: $state_A(\alpha, x, n) \equiv state_A(\hat{\alpha}, x, k)$ for $x \in \{t, r\}$, and in both s_n and $\hat{s}_k$, both physical channels are clean. Suppose $\hat{\alpha}_1 = \hat{s}_0\hat{\pi}_1\hat{s}_1 \ldots \hat{\pi}_k\hat{s}_k\hat{\pi}_{k+1}\hat{s}_{k+1} \ldots \hat{\pi}_{k+i}\hat{s}_{k+i}$ is a finite execution of $\hat{D}'(A)$ that is an extension of $\hat{\alpha}$. Then there exists a finite execution $\alpha_1 = s_0\pi_1 s_1 \ldots \pi_n s_n \pi_{n+1} s_{n+1} \ldots \pi_{n+i} s_{n+i}$ that is an extension of α such that for all j with $1 \leq j \leq i$, $\hat{\pi}_{k+j} \equiv \pi_{n+j}$ and $state_A(\alpha, x, n+j) \equiv state_A(\hat{\alpha}, x, k+j)$ for $x \in \{t, r\}$.*

The next lemma will be crucial in the inductive proof of Lemma 7.3. Speaking informally, we use it to "pump up" the sequence of packets waiting in the channels, as illustrated in Figure 4. If a schedule can leave the system so that waiting in one physical channel is a sequence of packets equivalent to the packets delivered across that channel in a reference execution, then we can extend the schedule by crashing the destination host and replaying that host's part of the reference execution, and this can leave the system so that a sequence of packets is waiting in the other physical channel, equivalent to the packets sent by the host in the reference execution.

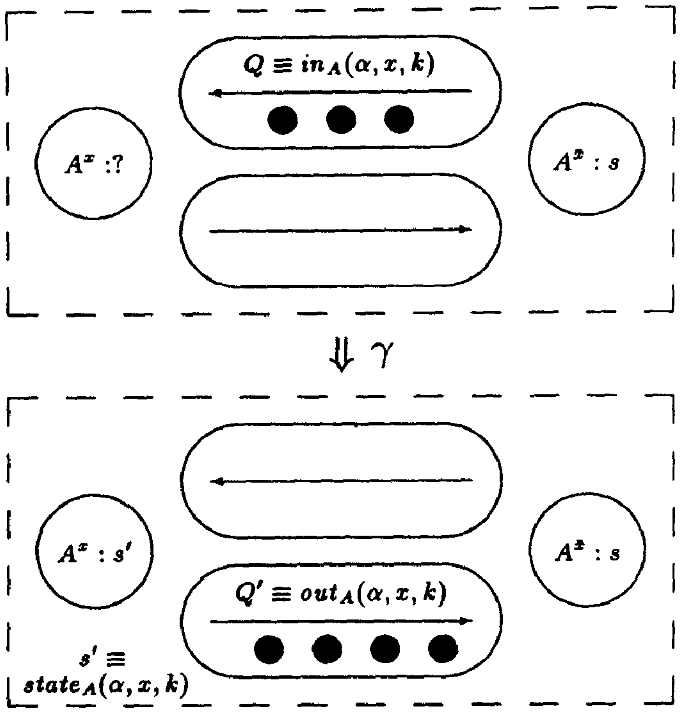

Figure 4: Illustration for Lemma 7.2

Lemma 7.2 *Let $A = (A^r, A^t)$ be a message-independent, crashing data link protocol. Let $\alpha = s_0\pi_1 s_1 \ldots \pi_n s_n$ be an execution of $\hat{D}'(A)$ such that $\pi_1 = wake^{t,r}$, $\pi_2 = wake^{r,t}$ and no wake, fail or crash events occur in $\pi_3 \ldots \pi_n$. Suppose $x \in \{t, r\}$, k is an integer with $2 \leq k \leq n$ and β is a finite schedule of $\hat{D}'(A)$ with the following properties:*

1. *$beh(\beta)$ is well-formed, satisfies (DL1)-(DL3), and contains unbounded transmitter and receiver working intervals, and*
2. *β can leave $\hat{D}'(A)$ in a state where the state of $A^{\bar{x}}$ is s, and a sequence Q of distinct packets is waiting in the state of $\hat{C}^{\bar{x},x}$ such that $Q \equiv in_A(\alpha, x, k)$.*

Then there is a sequence γ of actions of A^x with the following properties:

1. *$\beta\gamma$ is a finite schedule of $\hat{D}'(A)$,*
2. *$beh(\beta\gamma)$ is well-formed, satisfies (DL1)-(DL3) and contains unbounded transmitter and receiver working intervals,*
3. *$\gamma \equiv crash^{x,\bar{x}} acts_A(\alpha, x, k)$, and*
4. *$\beta\gamma$ can leave $\hat{D}'(A)$ in a state where the state of $A^{\bar{x}}$ is s, the state of A^x is s' such that $s' \equiv$*

$state_A(\alpha, x, k)$, and a sequence Q' of distinct packets is waiting in the state of $\hat{C}^{x,\bar{x}}$ such that $Q' \equiv out_A(\alpha, x, k)$.

Proof: As notation, let $(s_0\pi_1 s_1 \ldots \pi_k s_k)|A^x = t_0\phi_1 t_1 \ldots \phi_l t_l$, so that $\phi_1\phi_2\ldots\phi_l = acts_A(\alpha, x, k)$, $t_l = state_A(\alpha, x, k)$, the sequence of packets sent in $\phi_1 \ldots \phi_l$ is $out_A(\alpha, x, k)$ and the sequence of packets received in $\phi_1 \ldots \phi_l$ is $in_A(\alpha, x, k)$. Also let $Q = q_1 q_2 \ldots q_{l'}$.

First we construct inductively an execution $\tilde{\alpha}_1$ of A^x. To begin, let $s'_0\pi'_1 s'_1 \ldots \pi'_{j-1}s'_{j-1}$ be some execution of A^x with schedule $\beta|A^x$; such an execution exists because $\beta|A^x$ is a schedule of A^x by Lemma 2.2. Put $\pi'_j = crash^{x,\bar{x}}$ and put $s'_j = t_0$, the initial state of A^x. Since A^x is crashing, (s'_{j-1}, π'_j, s'_j) is a step of A^x. Put $\pi'_{j+1} = wake^{x,\bar{x}} = \phi_1$ and $s'_{j+1} = t_1$. Then $(s'_j, \pi'_{j+1}, s'_{j+1})$ is a step of A^x since (t_0, ϕ_1, t_1) is.

So suppose that we have so far constructed $s'_0\pi'_1 s'_1 \ldots \pi'_{j+i}s'_{j+i}$ for i such that $1 \le i < l$, so that $s'_{j+i} \equiv t_i$. We show how to define π'_{j+i+1} and then how to define s'_{j+i+1}.

1. If $\phi_{i+1} = receive\_pkt^{\bar{x},x}(p)$ then put $\pi'_{j+i+1} = receive\_pkt^{\bar{x},x}(q_h)$ where h is chosen so that ϕ_{i+1} is the h-th $receive\_pkt^{\bar{x},x}$ event in α. By the assumption that $Q \equiv in_A(\alpha, x, k)$, we have $\pi'_{j+i+1} \equiv \phi_{i+1}$. Since the automaton is input-enabled, π_{j+i+1} is enabled in s_{j+i}.
2. If $\phi_{i+1} = send\_msg^{t,r}(m)$ (which can only happen if $x = t$) then put $\pi'_{j+i+1} = send\_msg^{t,r}(m')$ where m' is any message such that $send\_msg^{t,r}(m')$ does not occur in $\pi'_1\pi'_2 \ldots \pi'_{j+i}$. This is possible by the assumption that there is an infinite alphabet of messages. By the assumption of message-independence, $\pi'_{j+i+1} \equiv \phi_{i+1}$. Since the automaton is input-enabled, π_{j+i+1} is enabled in s_{j+i}.
3. If ϕ_{i+1} is a locally controlled action of A^x then let π'_{j+i+1} be a locally controlled action that is enabled in s'_{j+i} such that $\pi'_{j+i+1} \equiv \phi_{i+1}$. This is possible by the assumption of message-independence, since $s'_{j+i} \equiv t_i$ and ϕ_{i+1} is enabled in t_i.

By the assumption that ϕ_{i+1} is not a *wake*, *fail* or *crash* event, these exhaust the possibilities. Now choose s'_{j+i+1} so that $(s'_{j+i}, \pi'_{j+i+1}, s'_{j+i+1})$ is a step of A^x and $s'_{j+i+1} \equiv t_{i+1}$, which is possible by the assumption of message-independence, since $(t_i, \phi_{i+1}, t_{i+1})$ is a step of A^x and π'_{j+i+1} was chosen in every case to ensure that it was equivalent to ϕ_{i+1} and enabled in s'_{j+i}.

Completing the construction above gives a finite execution $\alpha_1 = s'_0\pi'_1 \ldots \pi'_{j+l}s'_{j+l}$ of A^x. Let $\gamma = \pi'_j\pi'_{j+1} \ldots \pi'_{j+l}$. By the construction we see $\gamma \equiv crash^{x,\bar{x}}\phi_1 \ldots \phi_l = crash^{x,\bar{x}}acts_A(\alpha, x, k)$. Since $beh(\beta)$ is well-formed, and γ begins with $crash^{x,\bar{x}}wake^{x,\bar{x}}$ and contains no subsequent *crash*, *wake* or *fail* events, we see that $beh(\beta\gamma)$ is well-formed. Similarly $beh(\beta\gamma)$ satisfies (DL1)-(DL3) and contains unbounded transmitter and receiver working intervals.

Now $\beta\gamma|A^x$ is just $\pi'_1\pi'_2 \ldots \pi'_{j+l}$, so $\beta\gamma$ is a finite schedule of A^x that can leave A^x in a state $s'_{j+l} \equiv t_l = state_A(\alpha, x, k)$. Also $\beta\gamma|A^{\bar{x}}$ is just $\beta|A^{\bar{x}}$ which is a finite schedule of $A^{\bar{x}}$ that can leave A^x in state s. Now $\gamma|\hat{C}^{\bar{x},x}$ is by construction $receive\_pkt^{\bar{x},x}(q_1) \ldots receive\_pkt^{\bar{x},x}(q_{l'})$ and since β can leave A^x in a state where Q is waiting in $\hat{C}^{\bar{x},x}$ we see by Lemma 6.4 that $\beta\gamma|\hat{C}^{\bar{x},x}$ is a finite schedule of $\hat{C}^{\bar{x},x}$. Finally $\gamma|\hat{C}^{x,\bar{x}}$ consists of $crash^{x,\bar{x}}$ followed by a sequence of $send\_pkt^{x,\bar{x}}$ actions which is equivalent to to the sequence of $send\_pkt^{x,\bar{x}}$ actions in $\phi_1\phi_2 \ldots \phi_l$. By Lemma 6.5, $\beta\gamma|\hat{C}^{x,\bar{x}}$ is a finite schedule of $\hat{C}^{x,\bar{x}}$ that can leave $\hat{C}^{x,\bar{x}}$ in a state in which a sequence Q' of packets is waiting, where Q' is equivalent to $out_A(\alpha, x, k)$.

Now we apply Lemma 2.3 to deduce that $\beta\gamma$ is a finite schedule of $\hat{D}'(A)$ that can leave $\hat{D}'(A)$ in a state where the state of $A^{\bar{x}}$ is s, the state of A^x is equivalent to $state_A(\alpha, x, k)$ and a sequence equivalent to $out_A(\alpha, x, k)$ is waiting in the state of $\hat{C}^{x,\bar{x}}$. □

The next lemma shows that we can find an execution that ends with the data link protocol in states equivalent to those in any suitable given execution, but with a sequence of packets equivalent to those sent in the original execution waiting in the channels.

Lemma 7.3 *Let A be a message-independent, crashing data link protocol. Let $\alpha = s_0\pi_1 s_1 \ldots \pi_n s_n$ be an execution of $\hat{D}'(A)$ such that $\pi_1 = wake^{t,r}$, $\pi_2 = wake^{r,t}$ and no wake, fail or crash events occur in $\pi_3 \ldots \pi_n$. Suppose k is an integer with $2 \le k \le n$. Let x denote the station such that*

$\pi_k \in acts(A^x)$. Then there is a finite schedule β of $\hat{D}'(A)$ with the following properties:

1. *$beh(\beta)$ is well-formed, satisfies (DL1)-(DL3), and contains unbounded transmitter and receiver working intervals, and*
2. *β can leave $\hat{D}'(A)$ in a state where the state of A^x is equivalent to $state_A(\alpha, x, k)$, the state of $A^{\bar{x}}$ is equivalent to $state_A(\alpha, \bar{x}, k)$, and a sequence Q of distinct packets is waiting in the state of $\hat{C}^{x,\bar{x}}$ such that $Q \equiv out_A(\alpha, x, k)$.*

Proof: Assume inductively that we have proved the lemma for all smaller values of k.

If all the actions $\pi_3, \ldots, \pi_k$ are in $acts(A^x)$, then $out_A(\alpha, \bar{x}, k)$ must be the empty sequence, and therefore we deduce that $in_A(\alpha, x, k)$ is also empty. Also $state_A(\alpha, \bar{x}, k)$ must be equal to $state_A(\alpha, \bar{x}, 2)$ Thus the sequence $wake^{\bar{x},x} wake^{x,\bar{x}}$ is a finite schedule of $\hat{D}'(A)$ with well-formed behavior satisfying (DL1)-(DL3) and containing unbounded tranmitter and receiver working intervals, that can leave $A^{\bar{x}}$ in state $state_A(\alpha, \bar{x}, k)$ with a sequence equivalent to $in_A(\alpha, x, k)$ waiting in $\hat{C}^{\bar{x},x}$. We can therefore apply Lemma 7.2 to obtain β.

Otherwise let j be the greatest integer such that $2 < j < k$ and $\pi_j \in acts(A^{\bar{x}})$. Then $in_A(\alpha, x, k)$ is a subsequence of $out_A(\alpha, \bar{x}, j)$, and $state_A(\alpha, \bar{x}, k)$ must equal $state_A(\alpha, \bar{x}, j)$. By using the assumed truth of the lemma for the smaller value j we get a schedule β_1 with well-formed behavior satisfying (DL1)-(DL3) and containing unbounded transmitter and receiver working intervals that can leave $A^{\bar{x}}$ in state equivalent to $state_A(\alpha, \bar{x}, j)$ with a sequence equivalent to $out_A(\alpha, \bar{x}, j)$ waiting in $\hat{C}^{\bar{x},x}$. By Lemma 6.6, β_1 can also leave $\hat{D}'(A)$ in a state with $A^{\bar{x}}$ in a state equivalent to $state_A(\alpha, \bar{x}, j)$, and with a sequence equivalent to $in_A(\alpha, x, k)$ waiting in $\hat{C}^{\bar{x},x}$. We can therefore apply Lemma 7.2 to obtain β. □

We can now use the previous lemma to find a schedule of a crashing message-independent data link protocol that can lead to states equivalent to those at the end of a given execution, but in which a message has been sent but not received.

Lemma 7.4 *Let $A = (A^t, A^r)$ be a message-independent, crashing data link protocol. Let $\alpha = s_0 \pi_1 s_1 \ldots \pi_n s_n$ be an execution of $\hat{D}'(A)$, such that $\pi_1 = wake^{t,r}$, $\pi_2 = wake^{r,t}$ and $beh(\alpha) = wake^{t,r} wake^{r,t} send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$. Then there is a finite schedule β of $\hat{D}'(A)$ with the following properties:*

1. *$beh(\beta)$ is well-formed and satisfies (DL1)-(DL3),*
2. *$beh(\beta)$ ends in $send\_msg^{t,r}(m_1)$ for some m_1,*
3. *β can leave $\hat{D}'(A)$ in a state where the state of A^t is equivalent to $state_A(\alpha, t, n)$, the state of A^r is equivalent to $state_A(\alpha, r, n)$, and the state of each physical channel is clean.*

Proof: Let n' denote the greatest integer less than or equal to n such that $\pi_{n'} \in acts(A^r)$. Lemma 7.3 yields a finite schedule β' of $\hat{D}'(A)$ with the following properties: $beh(\beta')$ is well-formed, satisfies (DL1)-(DL3), and contains unbounded transmitter and receiver working intervals, and β' can leave $\hat{D}'(A)$ in a state where the state of A^r is equivalent to $state_A(\alpha, r, n')$, and a sequence Q of distinct packets is waiting in the state of $\hat{C}^{r,t}$ such that $Q \equiv out_A(\alpha, r, n')$.

Since the sequence $in_A(\alpha, t, n)$ is a subsequence of $out_A(\alpha, r, n')$, we can use Lemma 6.6 to see that β' can also leave $\hat{D}'(A)$ in a state where the state of A^r is equivalent to $state_A(\alpha, r, n')$, and a sequence Q' is waiting in the state of $\hat{C}^{r,t}$ such that $Q' \equiv in_A(\alpha, t, n)$.

We can now apply Lemma 7.2 to obtain a sequence γ such that $\beta'\gamma$ is a finite schedule of $\hat{D}'(A)$, $beh(\beta'\gamma)$ is well-formed and satisfies (DL1)-(DL3), $\gamma \equiv crash^{t,r} acts_A(\alpha, t, n)$, and $\beta'\gamma$ can leave $\hat{D}'(A)$ in a state where the state of A^r is equivalent to $state_A(\alpha, r, n')$ and the state of A^t is equivalent to $state_A(\alpha, t, n)$. By using Lemma 6.3 to modify the states of the channels, we see $\beta'\gamma$ can also leave $\hat{D}'(A)$ in a state with all the properties listed already, and also both physical channels clean. We put $\beta = \beta'\gamma$.

We now note, using the definition of n', that $state_A(\alpha, r, n') = state_A(\alpha, r, n)$. Since γ is equivalent to $crash^{t,r} acts_A(\alpha, t, n)$ and $beh(acts_A(\alpha, t, n)) = beh(\alpha)|A^t$, we have that $beh(\beta)$ ends in $crash^{t,r} wake^{t,r} send\_msg^{t,r}(m_1)$ for some m_1. Since $beh(\beta)$ is well-formed and satisfies (DL1)-(DL3), we are done. □

Finally we can use the results above to prove our impossibility theorem.

Theorem 7.5 *There is no data link protocol that is weakly correct with respect to FIFO physical channels, and is message-independent and crashing.*

Proof: Assume that $A = (A^t, A^r)$ is such a protocol.

First we observe that there is a finite execution $\alpha = s_0\pi_1 s_1 \ldots \pi_n s_n$ of $\hat{D}'(A)$ with the following properties: $beh(\alpha) = wake^{t,r} wake^{r,t} send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$ for some m, $\pi_1 = wake^{t,r}$, $\pi_2 = wake^{r,t}$, and in s_n each physical channel is clean. The existence of such an α is proved by using Lemma 4.1 to get an execution with the required behavior, truncating it after the state following the $receive\_msg^{t,r}(m)$ event (to make it finite), and finally using Lemma 6.3 to alter the component of each state of each physical channel, without altering the schedule, so as to leave the physical channels clean.

Next we appeal to Lemma 7.4 to obtain a finite execution $\hat{\alpha} = \hat{s}_0\hat{\pi}_1\hat{s}_1 \ldots \hat{\pi}_k\hat{s}_k$ of $\hat{D}'(A)$ with the following properties: $beh(\hat{\alpha})$ is well-formed, satisfies (DL1)-(DL3), ends in $send\_msg^{t,r}(m_1)$ for some m_1, $state_A(\hat{\alpha}, x, k) \equiv state_A(\alpha, x, n)$ for $x \in \{t, r\}$, and each physical channel is clean in $\hat{s}_n$.

By Lemma 2.1, there is a fair execution of $\hat{D}'(A)$ that extends $\hat{\alpha}$ and contains no additional inputs to $\hat{D}'(A)$. The behavior of this extension is well-formed and satisfies (DL1)-(DL3) since $beh(\hat{\alpha})$ has these properties, and they are not affected by output actions. Thus the behavior of this extension must satisfy (DL8). Since $send\_msg^{t,r}(m_1)$ is followed in the extension by no input action of $\hat{D}'(A)$, it occurs in an unbounded transmitter working interval. The extension therefore contains $receive\_msg^{t,r}(m_1)$ by (DL8). Thus the suffix of the extension after $\hat{\alpha}$ contains at least one $receive\_msg^{t,r}$ event, and it contains no input actions of $\hat{D}'(A)$. Let m_2 be the message parameter in the first $receive\_msg^{t,r}$ event in the suffix of the extension. By truncating the extension after this $receive\_msg^{t,r}(m_2)$ event, we obtain a finite execution $\hat{\alpha}_1$ of $\hat{D}'(A)$ with the following properties: it extends $\hat{\alpha}$, and $beh(\hat{\alpha}_1) = beh(\hat{\alpha}) receive\_msg^{t,r}(m_2)$.

Applying Lemma 7.1 to the executions α, $\hat{\alpha}$ and $\hat{\alpha}_1$, we deduce the existence of a finite execution α_1 of $\hat{D}'(A)$ such that α_1 extends α and the actions in the suffix of α_1 after α are equivalent to those in the suffix of $\hat{\alpha}_1$ after $\hat{\alpha}$. Thus α_1 has the following properties: it extends α, and $beh(\alpha_1) = beh(\alpha) receive\_msg^{t,r}(m_3)$ for some m_3. Note that $beh(\alpha_1)$ is well-formed and satisfies (DL1)-(DL3).

Now we use Lemma 2.1 to get a fair extension of α_1 with no additional inputs. This extension (whose behavior is well-formed and satisfies (DL1)-(DL3)) contains no additional outputs by (DL4) and (DL5). Thus this fair extension has behavior equal to $beh(\alpha_1)$. Thus we have shown that the sequence $wake^{t,r} wake^{r,t} send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$ $receive\_msg^{t,r}(m_3)$ is a fair behavior of $\hat{D}'(A)$.

If $m_3 \neq m$ this fair behavior does not satisfy (DL5), since it contains $receive\_msg^{t,r}(m_3)$ but no $send\_msg^{t,r}(m_3)$. If $m_3 = m$ this fair behavior does not satisfy (DL4) since it contain two events $send\_msg^{t,r}(m)$. In either case, since the fair behavior is well-formed and does satisfy (DL1)-(DL3), we have found a contradiction with the assumption that $\hat{D}'(A)$ solves $WDL^{t,r}$. □

8 Using Bounded Headers With Non-FIFO Channels

In this section, we consider the case where the physical channel need not be FIFO; non-FIFO physical channels make the design of data link protocols more difficult than FIFO physical channels. We show that it is impossible to have a weakly correct, message-independent data link protocol that has bounded headers.

8.1 k-bounded Protocols

Our impossibility proof requires a technical restriction, that the protocol be "k-bounded". This restriction means that for any message, there is some execution in which at most k packets are used to transmit the message. Most practical protocols are in fact 1-bounded. The formal definition of k-boundedness is made in terms of the permissive physical channel $\bar{C}^{t,r}$ defined earlier.

We require a preliminary definition. Namely, a sequence of data link layer actions β is *valid* if (1) β is well-formed, (2) β satisfies (DL1) to (DL5) and (DL8), and (3) a *wake* event, but no *fail* or *crash* events, occur in β.

The following lemmas give basic properties of valid sequences.

Lemma 8.1 *Let β be a valid sequence of data link layer actions. Let m be a message. If $send\_msg^{t,r}(m)$ occurs in β then $receive\_msg^{t,r}(m)$ occurs in β.*

Proof: Suppose a $send\_msg^{t,r}(m)$ occurs in β. By (DL1) the $send\_msg^{t,r}(m)$ event occurs in a transmitter working interval in β. Since there are no *fail* or *crash* events in β, this working interval is unbounded. Since (DL8) is satisfied, a $receive\_msg^{t,r}(m)$ also occurs in β. □

Lemma 8.2 *Let β be a valid sequence of data link layer actions and let m be a message such that $send\_msg^{t,r}(m)$ does not occur in β. Then $\beta send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$ is a valid sequence.*

Recall that $\bar{D}'(A) = hide_{\Phi}(\bar{D}(A))$, where $\bar{D}(A)$ is the composition of A^t, A^r, $\bar{C}^{t,r}$ and $\bar{C}^{r,t}$, and Φ is the subset of $acts(\bar{D}(A))$ consisting of *send_pkt* and *receive_pkt* actions.

We say that A is *k-bounded* if the following condition holds for every finite schedule β of $\bar{D}'(A)$ such that $beh(\beta)$ is valid, and for every message m such that $send\_msg^{t,r}(m)$ does not occur in β: there is a schedule $\beta\gamma$ of $\bar{D}'(A)$ such that

1. $beh(\gamma) = send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$,
2. γ does not include any $receive\_pkt^{t,r}(p)$ actions such that $send\_pkt^{t,r}(p)$ occurs in β, and
3. the number of $receive\_pkt^{t,r}$ events in γ is at most k.

Suppose that A is a k-bounded data link protocol. Let β be a finite schedule of $\bar{D}'(A)$ such that $beh(\beta)$ is valid and let m be a message such that $send\_msg^{t,r}(m)$ does not occur in β. Then define $packet\_set_A(m,\beta)$ to be the set of packets received from t by r in some particular γ such that $\beta\gamma$ is a schedule of $\bar{D}'(A)$, $beh(\gamma) = send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$, γ does not include any $receive\_pkt^{t,r}(p)$ actions such that $send\_pkt^{t,r}(p)$ occurs in β, and such that the number of $receive\_pkt^{t,r}$ events in γ is at most k. Such a γ exists by the definition of k-boundedness.

8.2 The Proof

The essence of this section is to take a supposed message-independent, k-bounded weakly correct data link protocol with bounded headers, and to produce a schedule in which every message sent has been delivered, but a large collection of packets is in transit, in fact, a collection equivalent to the set of packets which can be used to transmit a new message. If those packets in transit are now delivered, the receiving automaton will announce delivery of a message although none was sent that has not been delivered already, contradicting the assumed weak correctness of the protocol.

We begin by defining a partial order between sets of packets, with a parameter k, with respect to an equivalence relation $\equiv$, in the following way: $T <_{k,\equiv} T'$ if: (1) $T \subseteq T'$, and (2) there exists a packet p, such that $p \in T'$, $p \notin T$ and the number of packets $p' \in T$ such that $p \equiv p'$ is less than k.

When the equivalence relation, $\equiv$, is clear from the context we use the notation $<_k$ for $<_{k,\equiv}$.

We now prove the crucial inductive step that we will use to "pump up" the collection of packets in transit.

Lemma 8.3 *Let k be an integer. Let A be a weakly correct k-bounded data link protocol that is message-independent with respect to $\equiv$. Let β be a finite schedule of $\bar{D}'(A)$ such that $beh(\beta)$ is valid, and let T be a set of packets that are in transit in $\beta|\bar{C}^{t,r}$. Then at least one of the following holds.*

1. *There exists a message m such that $send\_msg^{t,r}(m)$ does not occur in β and there is a one-to-one mapping, f, from the packets in $packet\_set_A(m,\beta)$ to the packets in T, such that $p \equiv f(p)$ for all p.*
2. *There is a finite schedule $\beta\gamma$ of $\bar{D}'(A)$ such that:*
   - (a) *$beh(\beta\gamma)$ is valid,*
   - (b) *γ does not include any $receive\_pkt^{t,r}(p)$ action such that $send\_pkt^{t,r}(p)$ occurs in β, and*
   - (c) *there exists a set T' of packets in transit in $\beta\gamma|\bar{C}^{t,r}$, where $T <_k T'$.*

Proof: Fix k, A, β and T as in the hypotheses. Let m be any message such that $send\_msg^{t,r}(m)$ does not occur in β. Since A is k-bounded, there exists a sequence γ_1 such that $\beta\gamma_1$ is a schedule of $\bar{D}'(A)$, $beh(\gamma_1) = send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$, γ_1 does not include any $receive\_pkt^{t,r}(p)$ events such that $send\_pkt^{t,r}(p)$ occurs in β, and the packets delivered from t to r in γ_1 are $packet\_set_A(m,\beta)$ and therefore are at most k in number. It follows from Lemma 8.2 that $beh(\beta\gamma_1)$ is valid.

If for every packet p in $packet\_set_A(m, \beta)$ there are at least k packets p' in T such that $p' \equiv p$, then by standard results in combinatorics there is a one-to-one function f from packets in $packet\_set_A(m, \beta)$ to packets in T, such that $f(p) \equiv p$ for all p. In such a case (1) holds.

Otherwise, we can find some packet p_0 in $packet\_set_A(m, \beta)$ such that there are fewer than k packets p' in T such that $p_0 \equiv p'$. Since γ_1 contains $receive\_pkt^{t,r}(p_0)$ and no message sent in β is delivered in γ_1, γ_1 also contains $send\_pkt^{t,r}(p_0)$. Let ρ denote the prefix of γ_1 up to and including $send\_pkt^{t,r}(p_0)$. We claim that there exists a sequence $\hat{\rho}$ such that using $\gamma = \rho\hat{\rho}$, $\beta\gamma$ satisfies (2).

In case either $receive\_msg^{t,r}(m)$ is in ρ or $send\_msg^{t,r}(m)$ is not in ρ, $\hat{\rho}$ can be taken to be the empty sequence. (In the former case, Lemma 8.2 implies that $beh(\beta\rho)$ is valid.) So suppose that $send\_msg^{t,r}(m)$ is in ρ and $receive\_msg^{t,r}(m)$ is not in ρ.

By Lemma 6.3, there is an execution α' of $\bar{D}'(A)$ such that $sched(\alpha) = \beta\rho$ and $\bar{C}^{t,r}$ is clean in the final state of α. By Lemma 2.1, there is a fair execution α'' of $\bar{D}'(A)$ such that α'' extends α' and contains no input events of $\bar{D}'(A)$ except those in α'. Let $beh(\alpha'') = \beta\rho\rho'$. Thus $\beta\rho\rho'$ is a fair schedule of $\bar{D}'(A)$.

Since A is weakly correct and $beh(\beta\rho\rho')$ is well-formed and satisfies (DL1)-(DL3), $beh(\beta\rho\rho')$ also satisfies (DL8). Since $send\_msg^{t,r}(m)$ occurs in $beh(\beta\rho\rho')$, (DL8) implies that $receive\_msg^{t,r}(m)$ also occurs in $beh(\beta\rho\rho')$. Let $\hat{\rho}$ be the prefix of ρ' ending with $receive\_msg^{t,r}(m)$. We claim that $\hat{\rho}$ has the needed properties.

First, since every message sent in β is received in β, and the only message sent in ρ is m, $\hat{\rho}$ contains no $receive\_msg^{t,r}$ events except $receive\_msg^{t,r}(m)$ by (DL4) and (DL5). Thus $beh(\beta\rho\hat{\rho}) = beh(\beta)send\_msg^{t,r}(m)receive\_msg^{t,r}(m)$ which is valid by Lemma 8.2. Second, since $\hat{\rho}$ is the schedule of an execution fragment that begins with $\bar{C}^{t,r}$ in a clean state $\rho\hat{\rho}$ does not include any $receive\_pkt^{t,r}(p)$ such that $send\_pkt^{t,r}(p)$ occurs in β. Finally, the choice of $T' = T \cup \{p_0\}$ satisfies the third claim. □

Using the above we can find a schedule in which every message sent has been delivered, but where a large collection of packets are in transit.

Lemma 8.4 *Let k be an integer. Let $A = (A^t, A^r)$ be a weakly correct k-bounded data link protocol that is message-independent with respect to $\equiv$, and has bounded headers. Then there exist a finite schedule β of $\bar{D}'(A)$, a set T of packets, and a message m such that the following conditions are true. (1) $beh(\beta)$ is valid, (2) every packet in T is in transit in $\beta|\bar{C}^{t,r}$, (3) $send\_msg^{t,r}(m)$ does not occur in β, and (4) there is a one-to-one mapping, f, from the packets in $packet\_set_A(m, \beta)$ to the packets in T, such that $p \equiv f(p)$ for all p.*

Proof: Let H be the finite set $headers(A, \equiv)$. By the definition of the partial order $<_{k,\equiv}$, the maximum length of a chain of sets in the $<_{k,\equiv}$ order is at most $k \cdot |H|$.

Starting with β_1 as the schedule $wake^{t,r}wake^{r,t}$, and T_1 as the empty set, we apply Lemma 8.3 repeatedly, obtaining β_i and $T_i, i = 2, \ldots$, as long as case (2) of the lemma holds. Since the construction insures that $T_i <_{k,\equiv} T_{i+1}$ for all $i \geq 1$, eventually case (1) of Lemma 8.3 must hold. That is, for some fixed i, β_i is a schedule of $\bar{D}'(A)$, all packets in the set T_i are in transit in β_i, and $beh(\beta_i)$ is valid; moreover, there exists a message m such that $send\_msg^{t,r}(m)$ does not occur in β_i and there is a one-to-one mapping, f, from the packets in $packet\_set_A(m, \beta_i)$ to the packets in T_i, such that $p \equiv f(p)$ for all p. Taking $\beta = \beta_i$ yields the result. □

Now we use the schedule given by the previous lemma to prove the impossibility result of this section.

Theorem 8.5 *There is no weakly correct data link protocol that is message-independent, has bounded headers, and is k-bounded for some k.*

Proof: Assume the contrary, and let $A = (A^t, A^r)$ be a data link protocol that satisfies all these conditions. Let H be the finite set $headers(A, \equiv)$. The proof is done by creating a schedule of $\bar{D}'(A)$ in which, for some message m, either $receive\_msg^{t,r}(m)$ appears twice, or a $receive\_msg^{t,r}(m)$ occurs although a $send\_msg^{t,r}(m)$ event does not occur.

Choose m, β and T satisfying Lemma 8.4. By the conclusions of that lemma and the definition of $packet\_set_A$, there exists a sequence γ_1 of actions such that $\beta\gamma_1$ is a schedule of $\bar{D}'(A)$,

$beh(\gamma_1) = send\_msg^{t,r}(m) receive\_msg^{t,r}(m)$, γ_1 does not include any $receive\_pkt^{t,r}(p)$ actions such that $send\_pkt^{t,r}(p)$ occurs in β, all the packets in T are in transit in β, and there is a one-to-one mapping, f, from the set of packets delivered at r in γ_1 to the set T such that $p \equiv f(p)$ for all p. We modify the schedule $\beta\gamma_1$ to reach the contradiction.

We will now construct a sequence γ_2 such that: (1) $\beta\gamma_2$ is a schedule of $\bar{D}'(A)$, (2) every $receive\_pkt^{t,r}(p)$ action in γ_2 has a $send\_pkt^{t,r}(p)$ in β, and (3) γ_2 is equivalent to $\gamma_1|A^r$.

Let α be an execution of A^r such that $sched(\alpha) = (\beta\gamma_1)|A^r$. We first construct a new execution α' of A^r and then define γ_2 so that $sched(\alpha') = (\beta|A^r)\gamma_2$.

The construction of α' is done by induction on the lengths of prefixes of α. Suppose $\alpha = s_0\pi_1 s_1 \cdots s_j$ and let α' be expressed in the form $\alpha' = s'_0\pi'_1 s'_1 \cdots s'_j$. For each i, the construction will ensure that $s_i \equiv s'_i$ and $\pi_i \equiv \pi_i$ '

As the basis, define α and α' to be identical up to and including the state just after the portion having schedule $\beta|A^r$. Now suppose that $s'_0\pi'_1 s'_1 \cdots s'_i$ has already been defined and consider $\pi'_{i+1}s'_{i+1}$.

If π_{i+1} is a $receive\_pkt^{t,r}(p)$ action, then define π'_{i+1} to be $receive\_pkt^{t,r}(f(p))$. By assumption on f, $p \equiv f(p)$, so that $receive\_pkt^{t,r}(p) \equiv receive\_pkt^{t,r}(f(p))$, i.e., $\pi_{i+1} \equiv \pi'_{i+1}$.

If π_{i+1} is a locally-controlled action of A^r, then since $s_i \equiv s'_i$ the message-independence assumption implies that there is an action equivalent to π_{i+1} that is enabled in s'_i; let π'_{i+1} be this action.

Note that these exhaust the possibilities because $beh(\gamma_1|A^r) = receive\_msg^{t,r}(m)$, so π_{i+1} cannot be $wake^{r,t}$, $fail^{r,t}$ or $crash^{r,t}$. Having defined π_{i+1}, we now define s_{i+1}. Since $s_i \equiv s'_i$ and $\pi'_{i+1} \equiv \pi_{i+1}$, the message-independence assumption implies that there is a state s such that $s \equiv s_{i+1}$ and (s'_i, π'_{i+1}, s) is a step of A^r. Let $s'_{i+1} = s$. This completes the construction of α'.

Now fix γ_2 so that $sched(\alpha') = (\beta|A^r)\gamma_2$. Then we claim that γ_2 has the required properties. Properties (2) and (3) are immediate from the construction, as is the fact that $(\beta\gamma_2)|A^r$ is a schedule of A^r. By construction, no action in γ_2 is in $acts(A^t)$, so $(\beta\gamma_2)|A^t = \beta|A^t$ which is a schedule of A^t. Since $\beta|\bar{C}^{r,t}$ is a schedule of $\bar{C}^{r,t}$, and by construction $\gamma_2|\bar{C}^{r,t}$ is just a sequence of $send\_pkt^{r,t}$ actions which are inputs to $\bar{C}^{r,t}$, we deduce that $(\beta\gamma_2)|\bar{C}^{r,t}$ is a schedule of $\bar{C}^{r,t}$. Finally notice that $\beta|\bar{C}^{t,r}$ is a schedule of $\bar{C}^{t,r}$, and $\gamma_2|\bar{C}^{t,r}$ is a sequence of $receive\_pkt^{t,r}$ actions for packets that are in transit from t to r in β. By Lemmas 6.7 and 6.4 $(\beta\gamma_2)|\bar{C}^{t,r}$ is a schedule of $\bar{C}^{t,r}$. Then Lemma 2.4 yields Property (1), completing the proof of our claim.

Since the action $receive\_msg^{t,r}(m)$ occurs in $\gamma_1|A^r$ and $\gamma_2 \equiv \gamma_1|A^r$, there is some message m' such that the action $receive\_msg^{t,r}(m')$ occurs in γ_2. Fix m' for the remainder of the proof.

By Lemma 2.1 there is a fair schedule $\beta\gamma_2\gamma_3$ of $\bar{D}'(A)$ such that γ_3 contains no inputs to $\bar{D}'(A)$. This has behavior that is well-formed and satisfies (DL1)-(DL3). Since $beh(\beta)$ is valid, for every message m_i such that $send\_msg^{t,r}(m_i)$ occurs in β, the event $receive\_msg^{t,r}(m_i)$ also occur in β. The action $receive\_msg^{t,r}(m')$ appears in $\beta\gamma_2\gamma_3$. If the action $send\_msg^{t,r}(m')$ appears in β, then a $receive\_msg^{t,r}(m')$ event also occurs in β, so $beh(\beta\gamma_2\gamma_3)$ does not satisfy (DL4). On the other hand, if the action $send\_msg^{t,r}(m')$ does not appear in β, then since no $send\_msg^{t,r}$ events occur in $\gamma_2\gamma_3$, we see that $beh(\beta\gamma_2\gamma_3)$ does not satisfy (DL5). Either case yields a contradiction with the assumption that $\bar{D}'(A)$ solves $WDL^{t,r}$. □

Note that the execution constructed in the preceding impossibility proof did not include any fail or crash actions. In fact, we could just as well have proved the result for a simpler sort of data link layer specification, not including fail or crash actions at all.

9 Discussion

The formal definitions we have given such as "message-independence" and "having bounded headers" seem to us to capture the essential features of the corresponding intuitive concepts as they appear in real network protocols, while also making the proofs easy. Alternative definitions could be given in some cases. We here mention a few points about these.

First, one might consider protocols where some simple information about the message content was used, for example the length might determine the number of packets needed to contain the message. This could be modelled by allowing different messages to be in different equivalence classes. All that seems needed for the proofs we have given to remain valid is the existence of some class that contains enough different messages. In the final version of this paper we expect to extend all the proofs to this case.

Second, one might consider protocols where the number of different headers used in the packets that transmit the first n messages is a function of n, rather than a constant as in a protocol with bounded headers. Stenning's protocol uses a new header for each new message, that is, the number of headers used grows linearly with n. We expect to model this in the final version of this paper, and repeat the proof given in Section 8 to show that using a sublinear number of headers is impossible if the physical channels might not be FIFO.

Acknowledgements

We would like to thank Baruch Awerbuch for many discussions. We also would like to thank Jennifer Welch for her helpful comments on several versions of the paper.

REFERENCES

[BS83] Baratz A. and Segall A., "Reliable Link Initialization Procedures," *Proceedings of the 3rd IFIP Workshop on Protocol Specification, Testing and Verification*, May 1983. To appear in *IEEE Transaction on Communication*, February 1988.

[C78] Cyper, R. J., *Communications Architecture for Distributed Systems*, Addison-Wesley, 1978.

[L88] Lynch N., "I/O Automata: A Model for Discrete Event Systems," *Proceedings of the 22nd Annual Conference on Information Sciences and Systems*, March 1988.

[LT87] Lynch N. A. and Tuttle M. R., "Hierarchical Correctness Proofs for Distributed Algorithms," *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing* pp. 137-151, August 1987.

[MW77] McQuillan, J. M., and Walden, D. C. "The ARPA Network Design decisions" *Comput. Networks*, vol. 1, pp. 243-289, August 1977.

[T] Tanenbaum A., *Computer Networks*, Prentice Hall, 1981.

[W80] Wecker, S., "DNA: the Digital Network Architecture", *IEEE Transactions on Communication*, vol. COM-28, pp. 510-526, April 1980.

[Z80] Zimmermann, H. "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection", *IEEE Transactions on Communication*, vol. COM-28, pp. 425-432, April 1980.