Solving Problems in the Presence of Process Crashes and Lossy Links*

Anindya Basu† Bernadette Charron-Bost‡ Sam Toueg†

September 24, 1996

Abstract

We study the effect of link failures on the solvability of problems in asynchronous systems that are subject to process crashes: given a problem that can be solved in a system with process crashes and reliable links, is the problem solvable even if links are lossy? We answer this question for two types of lossy links, and show that the answer depends on the maximum number of processes that may crash and the nature of the problem to be solved. In particular, we prove that the answer is positive if fewer than half of the processes may crash or if the problem specification does not refer to the state of processes that crash. However, in general, the answer is negative even if each link can loose only a finite number of messages.

1 Introduction

We study the effect of link failures on the solvability of problems in distributed systems. In particular, we address the following question: *given a problem that can be solved in a system where the only possible failures are process crashes, is the problem still solvable if links can also fail by losing messages?* The answer depends on several factors, including the synchrony of the system, the model of link failures, the maximum number of process failures, and the nature of the problem to be solved.

In this paper, we focus on asynchronous systems (results concerning synchronous systems will be described in a companion paper). The set of problems solvable in asynchronous systems with process crashes include *Reliable, FIFO,* and *Causal Broadcast*, and their *uniform* counterparts [Bir85, HT94], as well as *Approximate Agreement* [DLP+86], *Renaming* [ABND+90], and *k-set Agreement* [Cha90]. The question is whether such problems remain solvable (and if so how) if we add link failures.

We consider two models of lossy links: *eventually reliable* and *fair lossy*. Roughly speaking, they have the following properties: with an eventually reliable link, there is a time after which all messages sent are eventually received (messages sent before that time may be lost). Such a link can lose only a finite (but unbounded) number of messages. With a fair lossy link, if an infinite number of messages are sent, an infinite subset of these messages is received. Such a link can lose an infinite number of messages. Clearly, any algorithm that works with fair lossy links also works with eventually reliable links. Thus, to make our results as strong as possible, we assume eventually reliable links when we prove impossibility results, and fair lossy links when we show problems to be solvable.[1]

*Research partially supported by NSF grant CCR-9402896, DARPA/NASA Ames grant NAG-2-593, Air Force Material Contract F30602-94-C-0224 and ONR contract N00014-92-J-I866.

†Department of Computer Science, Upson Hall, Cornell University, Ithaca, NY 14853, USA.

‡Laboratoire d'Informatique LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, FRANCE.

[1] Eventually reliable or fair lossy links do not lead to permanent network partitioning. Such partitioning renders most interesting problems trivially impossible.

Since an eventually reliable link can lose only a finite number of messages, it may appear that one can mask these message losses by repeatedly sending copies of each message, or by piggybacking on each message all the messages that were previously sent. Such a scheme is highly inefficient, but it does seem to simulate a reliable link. So it appears that, in principle, any problem that can be solved in a system with process crashes and reliable links, remains solvable in a system with process crashes and eventually reliable links.

Our first two results concern systems where half (or more) processes may crash. We first show that the intuition described above is flawed. We do so by exhibiting a problem, *Uniform Reliable Broadcast* [HT94], that is solvable with reliable links but not with eventually reliable links. However, not all problems are like Uniform Reliable Broadcast. Our second result characterizes a large class of problems that *remain solvable* even with fair lossy links. Informally, this class consists of all the problems whose specifications refer only to the behavior of correct processes (i.e., processes that do not crash) — these are called *correct-restricted* [Gop92] or *failure-insensitive* [BN92] problems.[2] This class of problems includes Reliable, FIFO, and Causal Broadcast, and correct-restricted versions of Approximate Agreement, Renaming, and k-set Agreement. For such problems, we show how to automatically transform any algorithm that works in a system with process crashes and reliable links into one that works with process crashes and fair lossy links.

Our final result concerns systems where a majority of processes are correct. In this case, we show that any problem that is solvable with process crashes and reliable links is also solvable with process crashes and fair lossy links. We do this by showing that given a system with fair lossy links and a majority of correct processes, one can simulate a system with reliable links.

The problem of tolerating crash and/or link failures has been extensively studied (e.g., [AAF$^+$94, AE86, AGH90, BSW69, FLMS93, GA88, JV96, WZ89]). Several papers focus on a single link and on how to mask failures of that link [AAF$^+$94, BSW69]. In contrast, we study lossy links in the context of an entire system: we show that the effect of lossy links depends on the proportion of faulty processes in the system. Other works have also studied tolerating lossy links in the context of an entire network. However, most of them focus on the solution of specific problems such as end-to-end communication or broadcast [AE86, AGH90, GA88]. Moreover, some of them do not consider process crashes [GA88], while others assume that crashed processes recover [AGH90]. In contrast, our work considers the effect of lossy links and permanent process crashes on the solvability of problems in general. This brings to the fore the importance of the notion of correct-restricted problems. Concurrent work [JV96] also studies problem solvability in general, but assumes that crashed processes recover (and have no stable storage), and focuses on an impossibility result.

The paper is organized as follows. In Section 2, we define our model, including the various types of links that we consider. Sections 3 and 4 consider systems where a majority of processes may crash. We first prove that in general, reliable links cannot be simulated by eventually reliable links (Section 3). We then show that "natural" correct-restricted problems that are solvable with reliable links are also solvable with fair lossy links (Section 4). In Section 5, we consider systems where a majority of processes are correct, and show how to simulate reliable links with fair lossy links. Finally, in Section 6 we state our results more formally using a refinement of the model and the notion of translation.

2 Model

We consider asynchronous distributed systems where processes communicate by message passing via a completely connected network, and there are no bounds on relative process speeds or message transmission times.

[2] The complement of this class of problems includes all problems with *uniform* specifications [NT90].

2.1 Variables and States

We postulate an infinite universal set of *variables* $\mathcal{V}$. Each variable v in $\mathcal{V}$ can be assigned a value from the set of natural numbers $\mathbb{N}$. A *state* s is a mapping $V \rightarrow \mathbb{N}$ for some subset of variables V of $\mathcal{V}$. We say that *state* s *is over variables* V, and write $var(s) = V$. For any v in V, the *value of* v *in state* s is $s(v)$. The set of all states is denoted S.

2.2 Processes

Let $P = \{p_1, \cdots, p_n\}$ be an indexed non-empty set of n processes. Each process p_i in P is formally defined by a set of *states* $\mathcal{Q}_i$, a set of *initial states* $\mathcal{Q}_i^0 \subseteq \mathcal{Q}_i$, a set of *actions* $\mathcal{A}_i$, a *transition relation* $\mathcal{T}_i$ on $\mathcal{Q}_i \times \mathcal{A}_i$, and a *state transition function* $\delta_i : \mathcal{Q}_i \times \mathcal{A}_i \rightarrow \mathcal{Q}_i$.

The set $\mathcal{Q}_i$ is a set of states over some finite (non-empty) set of variables $V_i \subset \mathcal{V}$. We say that V_i is the *set of variables of process* p_i. We assume that the sets of variables of distinct processes are disjoint.

The set $\mathcal{A}_i$ is the set of actions that p_i can execute. There are three types of actions: *send*, *receive*, and *internal*. To define the send and receive actions, we postulate a set $\mathcal{M}(P)$ of all the possible messages that processes in P can send. We assume that each message $m \in \mathcal{M}(P)$ has a header with three fields, $sender(m) \in P$, $dest(m) \in P$, and $tag(m)$, an integer used to differentiate messages.

The sets of send and receive actions of $\mathcal{A}_i$, denoted $Send(\mathcal{A}_i)$ and $Receive(\mathcal{A}_i)$, respectively, are defined as follows: $Send(\mathcal{A}_i) = \{send(m, p_j) \mid m \in \mathcal{M}(P),\ sender(m) = p_i,\ dest(m) = p_j\}$, and $Receive(\mathcal{A}_i) = \{receive(m) \mid m \in \mathcal{M}(P),\ dest(m) = p_i\} \cup \{receive(\perp)\}$. Action $send(m, p_j)$ models the sending of message m to p_j. Action $receive(m)$ models the receipt of message m, and action $receive(\perp)$ models the failure of p_i's attempt to receive a message (because no message was sent to p_i yet, or the messages sent to p_i are "in transit", or they were "lost", etc.).

The transition relation $\mathcal{T}_i$ on $\mathcal{Q}_i \times \mathcal{A}_i$ specifies which actions p_i can execute from any given state: $(s, a) \in \mathcal{T}_i$ iff p_i in state $s \in \mathcal{Q}_i$ can execute action $a \in \mathcal{A}_i$. To model the fact that it is not possible for a process to block because it does not have an action to execute, we assume that for every state $s \in \mathcal{Q}_i$ there exists at least one action $a \in \mathcal{A}_i$ such that $(s, a) \in \mathcal{T}_i$. To model the fact that a process can try to receive a message, but cannot select which message to receive, we assume if $(s, a) \in \mathcal{T}_i$ and $a \in Receive(\mathcal{A}_i)$, then for all $a' \in Receive(\mathcal{A}_i)$, $(s, a') \in \mathcal{T}_i$.

The state transition function $\delta_i : \mathcal{Q}_i \times \mathcal{A}_i \rightarrow \mathcal{Q}_i$ specifies what the state of p_i is after it executes an action. More precisely, if p_i is in state $s \in \mathcal{Q}_i$ and executes action $a \in \mathcal{A}_i$, then p_i goes into state $s' = \delta_i(s, a)$.

Finally, we find it convenient to assume that in every execution, messages are "unique" (this will be made more precise in Section 2.6). To enforce this, we assume that p_i increments a message counter each time it sends a message, and that each message is tagged with the current value of this counter. More precisely, we make the following assumptions on V_i, δ_i and $\mathcal{T}_i$. The set of variables V_i of p_i has a variable msg_cntr_i. If $s' = \delta_i(s, send(m, p_j))$ and $s(msg\_cntr_i) = k$, then $s'(msg\_cntr_i) = k + 1$. Moreover, if $(s, send(m, p_j))$ is in $\mathcal{T}_i$, then $tag(m) = s(msg\_cntr_i)$.

2.3 Events and Histories

An *event of process* $p_i \in P$ is a tuple $e = (p_i, a_i, l)$ where $a_i \in \mathcal{A}_i$ and $l \in \mathbb{N}$. We say that action a_i *is associated with* event e.

A *local history of process* $p_i \in P$, denoted $H[i]$, is a finite or an infinite sequence $s_i^0\, e_i^1\, s_i^1\, e_i^2\, s_i^2 \cdots$ of alternating states and events such that: (a) if $H[i]$ is finite, it terminates with a state, (b) $s_i^0 \in \mathcal{Q}_i^0$, (c) for all $k \geq 1$, $s_i^k \in \mathcal{Q}_i$ and $e_i^k = (p_i, a_i^k, k)$, and (d) for all $k \geq 0$, $(s_i^k, a_i^{k+1}) \in \mathcal{T}_i$ and $s_i^{k+1} = \delta_i(s_i^k, a_i^{k+1})$. The *state history of a local history* $H[i]$, denoted $\overline{H}[i]$, is the sequence of states in $H[i]$, namely $s_i^0\ s_i^1\ s_i^2 \cdots$. A *history* H *of* P is a vector of local histories $< H[1], H[2], \cdots, H[n] >$. The *state history of* H *of* P, denoted $\overline{H}$, is the vector $< \overline{H}[1], \overline{H}[2], \cdots, \overline{H}[n] >$. Vector $\overline{H}$ is also called a *state trace*, or simply a *trace*.

Process p_i is *correct in history* H if $H[i]$ is infinite; otherwise we say that p_i *crashes in history* H. The set of all correct processes in history H is denoted by *correct*(H).

2.4 Event Ordering

We relate events that occur in a history using the *happens-before* (henceforth abbreviated as *before*) relation defined in [Lam78]. The before relation $\prec_H$ over events of a history H is the smallest transitive relation such that: (1) if e and e' are different events in the same local history and e occurs before e' in that local history, then $e \prec_H e'$; (2) if, for some $m \in \mathcal{M}(P)$, $e = (p_i, \mathit{send}(m, p_j), k)$ and $e' = (p_j, \mathit{receive}(m), l)$ are events in H, then $e \prec_H e'$. We write $e \preceq_H e'$ if $e \prec_H e'$ or $e = e'$.

2.5 Systems of P

Let P be a set of processes. We define $\mathcal{H}(P)$ to be the set of all histories H of P such that $\prec_H$ is a strict partial order. Let H be any history in $\mathcal{H}(P)$ and H' be any down-set of H (i.e., H' is a vector such that, for every $p_i \in P$, $H'[i]$ is a prefix of $H[i]$, and if $H'[i]$ is finite it terminates with a state). Then, H' is also a history in $\mathcal{H}(P)$.

A system $\mathcal{S}(P)$ *of* P is a subset of $\mathcal{H}(P)$. We denote by $\overline{\mathcal{S}(P)}$ the set of traces in $\mathcal{S}(P)$, i.e., the set $\{\overline{H} \mid H \in \mathcal{S}(P)\}$.

2.6 Link Properties

Let P be a set of processes. As we saw in Section 2.2, each process p_i tags each message that it sends with a counter that is incremented after each sending. This ensures that in every history $H \in \mathcal{H}(P)$, messages are unique: if $(p_i, \mathit{send}(m, p_j), k)$ and $(p_s, \mathit{send}(m', p_t), k')$ are distinct events in H, then $m \neq m'$ (either $\mathit{sender}(m) \neq \mathit{sender}(m')$ or $\mathit{tag}(m) \neq \mathit{tag}(m')$).

We say that p_i *sends* m *to* p_j *in* H if event $(p_i, \mathit{send}(m, p_j), k)$ is in $H[i]$ for some k. Similarly, p_j *receives* m *from* p_i *in* H if event $(p_j, \mathit{receive}(m), l)$ with $\mathit{sender}(m) = p_i$ is in $H[j]$ for some l.

2.6.1 Reliable Links

A reliable link does not create, duplicate, or lose messages. Formally, the link from p_i to p_j is *reliable in history* H *of* P if H satisfies:

L1: (*No Creation*) For all $m \in \mathcal{M}(P)$, if p_j receives m from p_i, then p_i sends m to p_j.

L2: (*No Duplication*) For all $m \in \mathcal{M}(P)$, p_j receives m from p_i at most once.

L3: (*No Loss*) For all $m \in \mathcal{M}(P)$, if p_i sends m to p_j, and p_j executes receive actions infinitely often,[3] then p_j receives m from p_i.

[3] This implies that p_j is correct in H.

Implementing reliable links in a (non-blocking) asynchronous system requires infinite storage for buffering messages — finite buffers can overflow and thus cause message losses. Note that in our model every process has infinite storage.

2.6.2 Lossy Links

A lossy link can lose messages in transit. We consider two types of such links. The link from p_i to p_j is *eventually reliable in history H of P* if H satisfies L1, L2 and:

L4: (*Finite Loss*) If p_j executes receive actions infinitely often, then the number of messages sent by p_i to p_j that are not received by p_j is finite.

The link from p_i to p_j is *fair lossy in history H of P* if H satisfies L1 and L2 and:

L5: (*Fair Loss*) If p_i sends an infinite number of messages to p_j, and p_j executes receive actions infinitely often, then p_j receives an infinite number of messages from p_i.

Property **L3** implies **L4**, and **L4** implies **L5**. Thus, a reliable link is also eventually reliable, and an eventually reliable link is also fair lossy. A reliable link does not "lose" messages, an eventually reliable link can lose only a finite number of messages, and a fair lossy link can lose an infinite number of messages.

2.7 Systems of P with Reliable and Lossy Links

The *system of P with at most t process crashes and reliable links*, denoted $\mathcal{S}^t_R(P)$, is the set of all histories $H \in \mathcal{H}(P)$ such that at most t processes crash in H (i.e., at most t local histories $H[i]$ of H are finite) and all links are reliable in H (i.e., for all $p_i, p_j \in P$, the link from p_i to p_j is reliable in H). The *system of P with at most t process crashes and eventually reliable links*, denoted $\mathcal{S}^t_{ER}(P)$, and the *system of P with at most t process crashes and fair lossy links*, denoted $\mathcal{S}^t_{FL}(P)$, are similarly defined. Note that for all t, $\mathcal{S}^t_R(P) \neq \emptyset$ and $\mathcal{S}^t_R(P) \subseteq \mathcal{S}^t_{ER}(P) \subseteq \mathcal{S}^t_{FL}(P) \subseteq \mathcal{H}(P)$.

2.8 Problem Specifications, Solving a Problem

A problem specification is often given in the form of requirements on sets of traces. To see this, consider a problem like *Consensus*. Roughly speaking, a system $\mathcal{S}(P)$ of P solves Consensus, if $\overline{\mathcal{S}(P)}$ satisfies the following conditions: (a) in every trace $\overline{H} \in \overline{\mathcal{S}(P)}$, each process has some propose and decision variables that satisfy some agreement and validity requirement (e.g., correct processes agree on the value of their decision variables, a decision value must be a proposed one, etc.), and (b) $\overline{\mathcal{S}(P)}$ must have two traces $\overline{H}_0$ and $\overline{H}_1$ such that the initial value of all the propose variables is 0 in $\overline{H}_0$, and 1 in $\overline{H}_1$. Informally, the specification of Consensus is the set of all $\overline{\mathcal{S}(P)}$ for all P, that satisfy (a) and (b). In other words, it is the set of all sets of traces that satisfy (a) and (b).

To formally define a problem specification, we first need to define the set of all traces. Recall that S is the set of all states. Let *Seq*(S) be the set of all non-empty finite and infinite sequences over S such that all the states in a sequence have the same set of variables (i.e., for each σ in *Seq*(S), and any two states s and s' in σ, $var(s) = var(s')$). If $\sigma \in Seq(S)$, $var(\sigma)$ denotes the set of variables of any state in σ. The set of all traces, denoted *Vec*(S), is $\bigcup_{k \in \mathbb{N}} \{< \sigma_1, \sigma_2, \cdots, \sigma_k > \mid \forall i, j, 1 \leq i, j \leq k, \sigma_i \in Seq(S) \text{ and } var(\sigma_i) \cap var(\sigma_j) = \emptyset\}$.

Two traces $\overline{H}$ and $\overline{H'}$ in *Vec*(S) are *compatible* if they have the same dimension, say k, and for all i, $1 \leq i \leq k$, $var(\overline{H}[i]) = var(\overline{H'}[i])$. A set of traces is *proper* if it is non-empty and all its elements are compatible. The set of all proper sets of traces is $\Sigma^* = \{\overline{\mathcal{S}} \mid \overline{\mathcal{S}} \subseteq Vec(\mathcal{S})$ and $\overline{\mathcal{S}}$ is proper$\}$. A *problem specification* (or simply a *specification*) Σ is a subset of Σ^*.

Let Σ be a problem specification, P be a set of processes, and $\mathcal{S}(P)$ be a system of P. We say that $\mathcal{S}(P)$ *solves (problem specification)* Σ, if $\overline{\mathcal{S}(P)} \in \Sigma$.

2.9 Closure under Non-Trivial Reduction

The specifications of most problems satisfy a natural closure property that we now describe. Let P be a set of processes, and $\mathcal{S} = \mathcal{S}(P)$ and $\mathcal{S}' = \mathcal{S}'(P)$ be two systems of P. Suppose that $\mathcal{S}$ solves some problem specification Σ. Is it reasonable to require that if $\mathcal{S}' \subseteq \mathcal{S}$ then $\mathcal{S}'$ solves Σ? To understand this issue, consider a specific example: let Σ be the specification of Consensus (sketched in the previous section).

Since $\mathcal{S}$ solves Σ, then $\overline{\mathcal{S}}$ satisfies condition (a) of Σ, namely, every trace $\overline{H} \in \overline{\mathcal{S}}$ satisfies agreement and validity. If $\mathcal{S}' \subseteq \mathcal{S}$, it is obvious that $\overline{\mathcal{S}'}$ also satisfies condition (a). But the set $\overline{\mathcal{S}'} \subseteq \overline{\mathcal{S}}$ may not satisfy condition (b): for example, *every* trace $\overline{H} \in \overline{\mathcal{S}'}$ may start with all the propose variables equal to 0. In this case, $\mathcal{S}'$ does not solve Σ.[4] On the other hand, if $\overline{\mathcal{S}'}$ satisfies condition (b), then $\mathcal{S}'$ indeed solves Σ.

As with Consensus, it is often the case that a system $\mathcal{S}$ solves a problem specification Σ if its set of traces $\overline{\mathcal{S}}$ satisfies two types of conditions: one on *each* trace of $\overline{\mathcal{S}}$ (e.g., condition (a) of Consensus), and one on the *set* of initial states in $\overline{\mathcal{S}}$ (e.g., condition (b) of Consensus). In such a case, any subset $\mathcal{S}'$ of $\mathcal{S}$ that keeps all the initial states of $\mathcal{S}$ also solves the problem. This motivates the following definitions and assumption.

The *initial state of a trace* $\overline{H}$, denoted $init(\overline{H})$, is the vector $< s_1^0, s_2^0, \cdots, s_k^0 >$, where k is the dimension of $\overline{H}$, and for all i, $1 \le i \le k$, s_i^0 is the first state in $\overline{H}[i]$. For any $\overline{\mathcal{S}} \in \Sigma^*$, we define $init(\overline{\mathcal{S}}) = \{init(\overline{H}) \mid \overline{H} \in \overline{\mathcal{S}}\}$, the set of all initial states of all traces in $\overline{\mathcal{S}}$.

For all $\overline{\mathcal{S}}$ and $\overline{\mathcal{S}'}$ in Σ^*, we say that $\overline{\mathcal{S}'}$ *is a non-trivial reduction of* $\overline{\mathcal{S}}$ if $\overline{\mathcal{S}'} \subseteq \overline{\mathcal{S}}$ and $init(\overline{\mathcal{S}'}) = init(\overline{\mathcal{S}})$. A specification Σ is *closed under non-trivial reduction* if $\overline{\mathcal{S}} \in \Sigma$ and $\overline{\mathcal{S}'}$ is a non-trivial reduction of $\overline{\mathcal{S}}$ implies $\overline{\mathcal{S}'} \in \Sigma$. Henceforth, we consider only such specifications.

2.10 Correct-Restricted Problem Specifications

Intuitively, a problem specification is correct-restricted if it refers only to the states of correct processes (those with infinite traces) [Gop92, BN92]. Formally, let $\overline{H}$ and $\overline{H'}$ be any two traces in $Vec(S)$ with the same dimension, say k. Traces $\overline{H}$ and $\overline{H'}$ are *correct-equivalent*, denoted $\overline{H} \stackrel{c}{\sim} \overline{H'}$, if for all i, $1 \le i \le k$, if $\overline{H}[i]$ or $\overline{H'}[i]$ is infinite then $\overline{H'}[i] = \overline{H}[i]$. For any $\overline{\mathcal{S}}$ and $\overline{\mathcal{S}'}$ in Σ^*, we say that $\overline{\mathcal{S}'}$ is *a correct-restricted extension* of $\overline{\mathcal{S}}$, denoted $\overline{\mathcal{S}'} \ge_c \overline{\mathcal{S}}$, if $\overline{\mathcal{S}'} \supseteq \overline{\mathcal{S}}$ and $\forall \overline{H'} \in \overline{\mathcal{S}'}$, $\exists \overline{H} \in \overline{\mathcal{S}} : \overline{H} \stackrel{c}{\sim} \overline{H'}$. In other words, $\overline{\mathcal{S}'}$ is obtained from $\overline{\mathcal{S}}$ by adding some traces that are correct-equivalent to those in $\overline{\mathcal{S}}$. Finally, we say that a specification Σ is *correct-restricted* if for all $\overline{\mathcal{S}}, \overline{\mathcal{S}'} \in \Sigma^* : \overline{\mathcal{S}'} \ge_c \overline{\mathcal{S}}$ and $\overline{\mathcal{S}} \in \Sigma$ implies $\overline{\mathcal{S}'} \in \Sigma$.

Reliable Broadcast (RB) and *Consensus* are examples of problems with a correct-restricted specification. Their uniform counterparts (e.g., URB in Section 3) are *not* correct-restricted (their specifications refer to *all* processes, whether correct or faulty) [HT94].

3 Reliable is Strictly Stronger than Eventually Reliable

Since an eventually reliable link can lose only a finite number of messages, it may appear that one can mask these message losses by repeatedly sending copies of each message, or by piggybacking on each message all

[4]This is not fortuitous: we do not want to allow a system to trivially "solve" Consensus by just avoiding certain initial states.

the messages that were previously sent. Such a scheme is certainly inefficient,[5] but it does seem to simulate a reliable link (akin to a data link protocol that uses retransmissions to simulate a reliable link over a lossy one). So it may appear that any problem that is solvable with reliable links is also solvable with eventually reliable links. We now show that this intuition is incorrect: in systems where a majority of processes may crash, there are natural problems that can be solved with reliable links but not with eventually reliable links.

One such problem is *Uniform Reliable Broadcast* (or simply *URB*) [HT94]. Informally, URB is defined in terms of two primitives, *broadcast* and *deliver*, that must satisfy three properties:

- *Validity:* If a correct process broadcasts a message m, then it eventually delivers m.
- *Uniform agreement:* If a process (whether correct or faulty) delivers a message m, then all correct processes eventually deliver m.
- *Integrity:* For any message m, every correct process delivers m at most once, and only if m was previously broadcast by its sender.

A simple algorithm given in [HT94] solves URB with reliable links and any number of process crashes, and a standard partitioning argument shows that URB cannot be solved with eventually reliable links if a majority of processes may crash.

In the Appendix we use our model to formally specify and prove similar results about *Weak Uniform Reliable Broadcast* ($WURB$), a simple variant of URB. An informal definition of $WURB$ and the statement of these results follows. Process p_1 has a variable *message* initially set to 0 or 1. Every process p_i has a variable $delivery_i$ initially set to 0. If p_1 starts with $message = 1$ and p_1 is correct then p_1 eventually sets $delivery_1 = 1$. If p_1 sets $delivery_1 = 1$, then every correct process p_i should also set $delivery_i = 1$. Finally, if p_1 starts with $message = 0$ then no process p_i should ever set $delivery_i = 1$. The formal specification of WURB for a set of n processes, denoted Σ^n_{WURB}, is given in Section A.1 of the Appendix, where we show the following theorem:

Theorem 3.1 *1. For $0 \le t < n$, there is a set of processes P such that $\mathcal{S}^t_R(P)$ solves Σ^n_{WURB}.*

2. For $2 \le n \le 2t$, there is no set of processes P such that $\mathcal{S}^t_{ER}(P)$ solves Σ^n_{WURB}.

The above theorem implies that one cannot simulate reliable links with eventually reliable links when a majority of processes may crash. The precise statement of this impossibility result is given in Section 6.5 (Theorem 6.2), after the formal definition of simulation is given.

4 Solving Correct-Restricted Problems with Fair Lossy Links

The previous result does *not* mean that *all* problems that are solvable with reliable links are unsolvable with eventually reliable links. In fact, (most) correct-restricted problems that are solvable with reliable links are also solvable with fair lossy links, and thus with eventually reliable links. To prove this, we first introduce a new type of link that is weaker than a reliable link but stronger than an eventually reliable link — this intermediate link type is called *weakly reliable* (Section 4.1). We then show that any set of processes that solves a correct-restricted problem with reliable links also solves it with weakly reliable links (Section 4.2). Finally, we show how to simulate weakly reliable links with fair lossy links (Section 4.3). Note that weakly reliable links are introduced for technical reasons only — they may not model any "real" links.

[5] Indeed it may require the sending of an infinite number of message copies, or, alternatively, the sending of messages of infinite size.

4.1 Weakly Reliable Links: An Intermediate Model

Let P be a set of processes. The link from p_i to p_j is *weakly reliable in history H of P* if H satisfies **L1**, **L2**, and:

L6: (*No Visible Loss*) For all $m \in \mathcal{M}(P)$, if p_i sends m to p_j before some event e_l of some correct process p_l (according to $\prec_H$), and p_j executes receive actions infinitely often, then p_j receives m from p_i.

Roughly speaking, **L6** states that if the sending of a message m by p_i to p_j is "visible" to a correct process (because it is in the "causal past" of that process), then m is not lost: if p_j executes receive actions infinitely often, then it eventually receives m.

The *system of P with at most t process crashes and weakly reliable links*, denoted $\mathcal{S}^t_{WR}(P)$, is the set of all histories $H \in \mathcal{H}(P)$ such that at most t processes crash in H and all links are weakly reliable in H. Since **L3** implies **L6** and **L6** implies **L4**, we have $\mathcal{S}^t_R(P) \subseteq \mathcal{S}^t_{WR}(P) \subseteq \mathcal{S}^t_{ER}(P)$.

4.2 Solving Correct-Restricted Problems with Weakly Reliable Links

Any set of processes that solves a correct-restricted problem with reliable links also solves it with weakly reliable links. To show this formally, we first prove:

Lemma 4.1 *For any set of processes P and any t, if H is a history in $\mathcal{S}^t_{WR}(P)$ then there is a history H' in $\mathcal{S}^t_R(P)$ such that $\overline{H'} \overset{c}{\sim} \overline{H}$ and* $\mathrm{init}(\overline{H'}) = \mathrm{init}(\overline{H})$.

Proof: Let $H \in \mathcal{S}^t_{WR}(P)$. We construct H' from H by removing from H all the events that are not "visible" to correct processes in H (and deleting all the states that follow removed events). To do so, we first define $\vartheta(H) = \{\, e \mid \exists p_l \in \mathit{correct}(H),\ \exists e' \text{ in } H[l] : e \prec_H e' \}$. Intuitively, this is the set of all events that are "visible" to (i.e., in the "causal past" of) correct processes in H. Note that by transitivity of $\prec_H$, if $e' \in \vartheta(H)$ and $e \prec_H e'$ then $e \in \vartheta(H)$. We then construct H', the down-set of H that contains only the events in $\vartheta(H)$, as follows.

For each $H[i] = s_i^0\, e_i^1\, s_i^1\, e_i^2 \cdots s_i^{k-1}\, e_i^k\, s_i^k\, \cdots$:

1. If $H[i]$ is infinite, we define $H'[i] = H[i]$.
2. If $H[i]$ is finite, we define $H'[i] = s_i^0\, e_i^1\, s_i^1\, e_i^2 \cdots e_i^k\, s_i^k$ where k is the maximum index such that $e_i^k \in \vartheta(H)$. If $H[i]$ has no event in $\vartheta(H)$, then $H'[i] = s_i^0$.

From this construction it is clear that H' is a down-set of H, $\mathit{correct}(H') = \mathit{correct}(H)$, and the set of all events in H' is $\vartheta(H)$. Furthermore, $\overline{H'} \overset{c}{\sim} \overline{H}$ and $\mathit{init}(\overline{H'}) = \mathit{init}(\overline{H})$.

To show $H' \in \mathcal{S}^t_R(P)$, note first that $H' \in \mathcal{H}(P)$ (because it is the down-set of a history in $\mathcal{H}(P)$), and that at most t processes crash in H' (because $\mathit{correct}(H') = \mathit{correct}(H)$ and, since $H \in \mathcal{S}^t_{WR}(P)$, at most t processes crash in H). It remains to show that all links are reliable in H', i.e., for every two processes p_i and p_j, properties **L1**, **L2**, and **L3** hold in H'. We first note that since $H \in \mathcal{S}^t_{WR}(P)$, it satisfies **L1**, **L2**, and **L6**.

[**L2**] (*No Duplication*). Since H satisfies **L2**, for all $m \in \mathcal{M}(P)$, p_j receives m from p_i at most once in H. Since H' is a down-set of H, p_j receives m from p_i at most once in H'.

[**L1**] (*No Creation*). Suppose p_j receives m from p_i in H', and let e_j be the corresponding receive event. Since H' is a down-set of H, p_j receives m from p_i in H. Since H satisfies **L1**, p_i sends m to p_j in H; let e_i be the corresponding event. We have $e_i \prec_H e_j$. Since e_j is in H', $e_j \in \vartheta(H)$, and so $e_i \in \vartheta(H)$. Thus, e_i is also in H'. In other words, p_i sends m to p_j in H', as we needed to show.

[L3] (*No Loss*) Suppose p_i sends m to p_j in H', and p_j executes receive actions infinitely often in H'. We must show that p_j receives m from p_i in H'. First note that p_j executes receive actions infinitely often, and hence is correct, in *both* H' and H; moreover, $H'[j] = H[j]$. Let e_i be the event corresponding to p_i sending m to p_j in H'. By construction of H', $e_i \in \vartheta(H)$. Thus, there is an event e_l that occurs at some correct process p_l in H, such that $e_i \prec_H e_l$. Since H satisfies **L6**, and p_j executes receive actions infinitely often in H, p_j receives m from p_i in H, and therefore in H'. $\Box_{Lemma\ 4.1}$

Theorem 4.2 *Let Σ be any correct-restricted problem specification. For any set of processes P and any t, $\mathcal{S}^t_R(P)$ solves Σ if and only if $\mathcal{S}^t_{WR}(P)$ solves Σ.*

Proof: Let Σ be any correct-restricted specification, and $\mathcal{S} = \mathcal{S}^t_{WR}(P)$ and $\mathcal{S}' = \mathcal{S}^t_R(P)$. Suppose $\mathcal{S}$ solves Σ. Note that $\overline{\mathcal{S}'} \subseteq \overline{\mathcal{S}}$, and, by Lemma 4.1, *init*$(\overline{\mathcal{S}'}) =$ *init*$(\overline{\mathcal{S}})$. Thus, $\overline{\mathcal{S}'}$ is a non-trivial reduction of $\overline{\mathcal{S}}$. Since $\mathcal{S}$ solves Σ, $\mathcal{S}'$ also solves Σ.

Conversely, suppose $\mathcal{S}'$ solves Σ. We must show that $\mathcal{S}$ solves Σ. We know that $\overline{\mathcal{S}} \supseteq \overline{\mathcal{S}'}$. By Lemma 4.1, $\forall \overline{H} \in \overline{\mathcal{S}},\ \exists \overline{H'} \in \overline{\mathcal{S}'} : \overline{H'} \overset{c}{\sim} \overline{H}$. Thus, $\overline{\mathcal{S}}$ is a correct-restricted extension of $\overline{\mathcal{S}'}$. Since $\mathcal{S}'$ solves Σ and Σ is correct-restricted, $\mathcal{S}$ also solves Σ. $\Box_{Theorem\ 4.2}$

4.3 Simulating Weakly Reliable Links with Fair Lossy Links

Fair lossy links can be used to simulate weakly reliable links. To show this, we describe two procedures, *wr_send*(m, p_j) and *wr_recv*(m), that satisfy the properties of weakly reliable links when executed in any system with fair lossy links. We only give an informal description of this simulation and its proof here (a more formal treatment is postponed to later sections). Since our focus is on solvability (rather than efficiency), we describe the simplest *wr_send*(m, p_j) and *wr_recv*(m) simulation procedures that are sufficient to carry our result. These primitives are inefficient, indeed they assume infinite storage, and infinite message sizes.

To simulate weakly reliable links, we must ensure that properties **L1**, **L2**, and **L6** are satisfied. Roughly speaking, **L6** (No Visible Loss) stipulates that if a process p_i sends a message m to a process p_j and this sending is in the "causal past" of some correct process p_l, then if p_j executes receive actions infinitely often, it eventually receives m. We can achieve this property with fair lossy links as follows: process p_l maintains a list of all the messages that were sent in its causal past, and this list eventually includes the message m above. In addition, p_l sends this list to every process infinitely often, and in particular to process $p_j =$ *dest*(m). Since p_l is correct, property **L5** (Fair Loss) of the links from p_l to p_j ensures that p_j eventually receives (a list that contains) m.

The *wr_send*(m, p_j) and *wr_recv*(m) procedures (for process p_i) given in Figure 1 are based on the simple idea described above. Every process p_i maintains a queue *Prev_Sends*$_i$ that contains all the messages that were *wr_sent* in its "causal past". In order to *wr_send* a message m to process p_j, p_i simply appends m to its queue *Prev_Sends*$_i$. In addition, process p_i executes a Send Task to broadcast *Prev_Sends*$_i$ after every internal action as well as after every return from a *wr_send* or a *wr_recv* procedure. Note that if p_i is correct, it broadcasts *Prev_Sends*$_i$ infinitely often.

To *wr_recv* a message, p_i first executes a receive. If it receives some queue of messages *Prev_Sends* from some other process, p_i appends *Prev_Sends* to *Prev_Sends*$_i$. The *wr_recv* procedure now returns the first message in *Prev_Sends*$_i$ with destination p_i that p_i has not yet *wr_recv*d (it returns the null message $\perp$ otherwise).

We now sketch an informal proof that the *wr_send* and *wr_recv* procedures satisfy the properties of weakly reliable links, namely **L1**, **L2**, and **L6**.

Lemma 4.3 *For every process p_i, the queue* Prev_Sends$_i$ *is non-decreasing.*

Proof: Obvious. $\Box_{Lemma\ 4.3}$

Simulation code for process p_i:

Variables
Prev_Sends$_i$: a queue of messages, initially empty

Procedure *wr_send*(m, p_j) {simulating a send over a Weakly Reliable link}
append m to *Prev_Sends*$_i$
end Procedure

Procedure *wr_recv*(m) {simulating a receive over a Weakly Reliable link}
receive(*Prev_Sends*)
if *Prev_Sends* $\neq \perp$ **then** append *Prev_Sends* to *Prev_Sends*$_i$
if *Prev_Sends*$_i$ has a message m' such that
dest$(m') = p_i$ and p_i has not yet executed *wr_recv*(m')
then $m :=$ first such message in *Prev_Sends*$_i$
else $m := \perp$
end Procedure

Send Task {executed after every internal action and every *wr_send* and *wr_recv*}
for $j = 1, \cdots, n$ **do** *send*(*Prev_Sends*$_i, p_j$)

Figure 1: Simulating Weakly Reliable links with Fair Lossy links

Lemma 4.4 *For every process p_i, a message m is in* Prev_Sends$_i$ *only if* sender(m) wr_send*s* m.

Proof: The underlying links are fair lossy and thus do not create messages (property **L1** of fair lossy links). The result is now clear from the way *Prev_Sends*$_i$ is maintained, and can be obtained by a tedious induction that is omitted here. $\square_{Lemma\ 4.4}$

Lemma 4.5 *For every process p_i that executes* wr_recv *infinitely often, if m is in* Prev_Sends$_i$ *and* dest$(m) = p_i$*, then p_i* wr_recv*s* m.

Proof: Every time p_i executes *wr_recv*, it *wr_recv*s the *first* message in the queue *Prev_Sends*$_i$ with destination p_i that it has not yet *wr_recv*d (if such a message exists). It is now clear that once m is in the queue *Prev_Sends*$_i$, p_i will eventually *wr_recv* m. $\square_{Lemma\ 4.5}$

Lemma 4.6 *If p_i* wr_send*s m to p_j before some event e of a correct process p_l, then m is eventually in* Prev_Sends$_l$.

Proof: If $p_i = p_l$, process p_i appends m to *Prev_Sends*$_i$ during the execution of *wr_send*(m, p_j). If not (i.e., $p_i \neq p_l$), then since p_i *wr_sends* m to p_j before event e of p_l, by the definition of the before relation, there must exist some messages $m_0, m_1, \cdots, m_{k-1}$ and processes $p_i = p_{i_0}, p_{i_1}, \cdots, p_{i_{k-1}}, p_{i_k} = p_l$ such that:

0. p_{i_0} *wr_sends* m_0 to p_{i_1}, and
1. either $m = m_0$, or p_{i_0} *wr_sends* m to p_j before p_{i_0} *wr_sends* m_0 to p_{i_1}, and
2. p_{i_j} *wr_sends* m_j to $p_{i_{j+1}}$ before $p_{i_{j+1}}$ *wr_recvs* m_j, for $0 \leq j \leq k-1$, and

3. p_{i_j} *wr_recv*s m_{j-1} before p_{i_j} *wr_send*s m_j to $p_{i_{j+1}}$, for $1 \leq j \leq k-1$, and

4. either p_{i_k} *wr_recv*s m_{k-1} at the same time as event e of p_{i_k} occurs (i.e., the two events are the same), or it *wr_recv*s m_{k-1} before e.

Let p and q be any two processes, and m any message such that p *wr_send*s m to q, and q *wr_recv*s m. From Figure 1, it is easy to see that the queue *Prev_Sends*$_p$ of p immediately after p *wr_send*s m (note that this *Prev_Sends*$_p$ already contains m) is contained in the queue *Prev_Sends*$_q$ of q immediately after q *wr_recv*s m. From this observation, Lemma 4.3, and (1)-(4) above, we conclude that:

1. The queue *Prev_Sends*$_{i_0}$ immediately after p_{i_0} *wr_send*s m_0 to p_{i_1} contains m, and

2. the queue *Prev_Sends*$_{i_j}$ immediately after p_{i_j} *wr_send*s m_j to $p_{i_{j+1}}$ is contained in the queue *Prev_Sends*$_{i_{j+1}}$ immediately after $p_{i_{j+1}}$ *wr_recv*s m_j, for $0 \leq j \leq k-1$, and

3. the queue *Prev_Sends*$_{i_j}$ immediately after p_{i_j} *wr_recv*s m_{j-1} is contained in the queue *Prev_Sends*$_{i_j}$ immediately after p_{i_j} *wr_send*s m_j to $p_{i_{j+1}}$, for $1 \leq j \leq k-1$, and

4. the queue *Prev_Sends*$_{i_k}$ immediately after p_{i_k} *wr_recv*s m_{k-1} is is contained in the queue *Prev_Sends*$_{i_k}$ immediately after the event e.

Since m is in *Prev_Sends*$_i$ immediately after p_i *wr_send*s m, by chaining the above facts we conclude that m is contained in the queue *Prev_Sends*$_{i_k}$ of process $p_{i_k} = p_l$ immediately after the event e. $\Box_{Lemma\ 4.6}$

Theorem 4.7 *The simulation procedures* wr_send *and* wr_recv *satisfy the three properties* **L1**, **L2**, *and* **L6** *of weakly reliable links.*

Proof:

L1 (No Creation): Suppose p_i *wr_recv*s m from p_j. From the code of the *wr_recv* procedure it is clear that m is in *Prev_Sends*$_i$ and *dest*$(m) = p_i$. From Lemma 4.4, we conclude that p_j *wr_send*s m to p_i.

L2 (No Duplication): Obvious from the *wr_recv* procedure.

L6 (No Visible Loss): Suppose p_i *wr_send*s m to p_j before some event e of a correct process p_l, and p_j executes *wr_recv* actions infinitely often. We need to show that p_j *wr_recv*s m. Since p_j executes *wr_recv* actions infinitely often, from Figure 1 it is clear that p_j must execute receive actions infinitely often. By Lemma 4.6, m is eventually in *Prev_Sends*$_l$. By Lemma 4.3, m remains in *Prev_Sends*$_l$ forever. Since p_l is correct it sends its (current) queue *Prev_Sends*$_l$ infinitely often to p_j (in the Send Task), and these are the only messages that it sends to p_j. Only a finite number of these queues do not contain m. Since the link from p_l to p_j satisfies property **L5** (Fair Loss), p_j receives an infinite number of the queues that p_l sends to p_j. Thus, p_j eventually receives a queue that contains m, and from this receipt onwards, m is in *Prev_Sends*$_j$. By Lemma 4.5, p_j *wr_recv*s m. $\Box_{Theorem\ 4.7}$

This completes our informal proof that the *wr_send* and *wr_recv* procedures in Figure 1 simulate weakly reliable links using fair lossy links. By Theorem 4.2, if a correct-restricted problem is solvable with reliable links, then it is also solvable with weakly reliable links. Combining these two results, we conclude that for correct-restricted problems, fair lossy links are "as good as" reliable links in terms of problem solvability. A more precise statement of this claim is postponed to Section 6.6.

5 Simulating Reliable Links with Fair Lossy Links when $n > 2t$

Fair lossy links can be used to simulate reliable links, provided $n > 2t$ (i.e., a majority of processes are correct). To show this, we describe two procedures, *rel_send*(m, p_j) and *rel_recv*(m), that simulate the

Simulation code for process p_i:

Variables
Prev_Sends$_i$: a queue of messages, initially empty
Proc_Ack$_i$: a set of processes, initially empty

```
Procedure rel_send(m, p_j)                          {simulating a send over a reliable link}
    append m to Prev_Sends_i
    Proc_Ack_i := ∅
    while |Proc_Ack_i| < t + 1 do          {till m is echoed by at least t + 1 distinct processes}
        for j = 1, ..., n do send(Prev_Sends_i, p_j)
        receive(Prev_Sends)
        if Prev_Sends ≠⊥ then
            append Prev_Sends to Prev_Sends_i
            if m in Prev_Sends then Proc_Ack_i := Proc_Ack_i ∪ {sender(Prev_Sends)}
end Procedure

Procedure rel_recv(m)                               {simulating a receive over a reliable link}
    if Prev_Sends_i has a message m' such that
        dest(m') = p_i and p_i has not yet executed rel_recv(m')
    then m := first such message in Prev_Sends_i
    else m :=⊥
end Procedure

Send-Receive Task           {executed after every internal action and every rel_send and rel_recv}
    for j = 1, ..., n do send(Prev_Sends_i, p_j)                      {broadcast Prev_Sends_i}
    receive(Prev_Sends)                                               {receive a message}
    if Prev_Sends ≠⊥ then append Prev_Sends to Prev_Sends_i
```

Figure 2: Simulating Reliable links with Fair Lossy links when $n > 2t$

properties of reliable links when the underlying links are fair lossy and $n > 2t$. The simulation procedures that we give are simple but inefficient (they require infinite storage and infinite message sizes). The simulation and its correctness proof are informally described; a more formal treatment is deferred to later sections.

To simulate reliable links, every process p_i maintains a queue of messages *Prev_Sends*$_i$ that stores all messages that were *rel_sent* in the "causal past" of p_i. In addition, p_i executes a Send-Receive Task after every internal action as well as after every return from a *rel_send* or a *rel_recv* procedure. This task broadcasts *Prev_Sends*$_i$ and executes a receive. If any *Prev_Sends* queue is received, it is appended to the *Prev_Sends*$_i$ queue.

In order to *rel_send* a message m to a process p_j, process p_i invokes the *rel_send*(m, p_j) procedure. In this procedure, p_i first appends m to its queue *Prev_Sends*$_i$ and then repeatedly broadcasts *Prev_Sends*$_i$. When p_i receives echoes of m (inside *Prev_Sends* queues) from at least $t + 1$ distinct processes, p_i returns from *rel_send*(m, p_j). At this point at least one correct process, say p_l, has m in its *Prev_Sends*$_l$ queue. Since p_l sends its *Prev_Sends*$_l$ queue to p_j infinitely often, property **L5** of fair lossy links ensures that if p_j executes receive actions infinitely often, p_j eventually receives a *Prev_Sends*$_l$ queue containing m. Note that this holds even if p_i crashes after it returns from the *rel_send*(m, p_j) procedure.

In order to *rel_recv* a message, p_i invokes the *rel_recv* procedure which returns the first message in *Prev_Sends*$_i$ with destination p_i that p_i has not yet *rel_recv*d (it returns the null message $\perp$ otherwise).

We now sketch an informal proof that the *rel_send* and *rel_recv* procedures indeed satisfy the properties of reliable links, namely **L1**, **L2**, and **L3**.

Lemma 5.1 *For every process p_i, the queue* Prev_Sends$_i$ *is non-decreasing.*

Lemma 5.2 *No process blocks forever in the* rel_send *procedure.*

Proof: The proof is by contradiction. Suppose that some process p_i blocks (by spinning forever in the **while** loop) while executing *rel_send*(m, p_j). From Figure 2, p_i sends *Prev_Sends*$_i$ (which contains m) to all processes infinitely often. Moreover, each correct process executes receive actions infinitely often (even if it is itself blocked while executing a *rel_send*). By property **L5** of fair lossy links, each such correct process eventually receives *Prev_Sends*$_i$ (containing m) either during an execution of the Send-Receive Task or during an execution of the *rel_send* procedure, and it then echoes m repeatedly forever (inside a *Prev_Sends* that it sends infinitely often). By property **L5** of fair lossy links, p_i receives echoes of m from every correct process, i.e., from at least $t+1$ distinct processes (because $n > 2t$), and p_i cannot block forever in the *rel_send*(m, p_j) procedure — a contradiction. $\Box_{Lemma\ 5.2}$

Lemma 5.3 *For every process p_i, a message m is in* Prev_Sends$_i$ *only if* sender(m) rel_send*s* m.

Lemma 5.4 *For every process p_i that executes* rel_recv *infinitely often, if m is in* Prev_Sends$_i$ *and* dest$(m) = p_i$*, then p_i* rel_recv*s* m.

Theorem 5.5 *The simulation procedures* rel_send *and* rel_recv *satisfy the three properties* **L1**, **L2**, *and* **L3** *of reliable links.*

Proof:

L1 and **L2**: Similar to the proofs for **L1** and **L2** in Section 4.3.

L3 (No Loss): Suppose p_i invokes the *rel_send* procedure to send a message m to p_j, and p_j executes *rel_recv* actions infinitely often. If p_j *rel_recv*s message m, **L3** is satisfied. Now suppose that p_j does not *rel_recv* m. There are two cases:

1. Process p_i crashes while executing the *rel_send*(m, p_j) procedure, i.e., before returning from that procedure. In this case, we pretend that p_i crashes just before invoking *rel_send*(m, p_j) (i.e., just before *rel-sending* m). This simulates a reliable link where p_i crashes just before sending m — a behavior consistent with **L3**.

2. Process p_i does not crash while executing the *rel_send*(m, p_j) procedure. By Lemma 5.2, p_i returns from *rel_send*(m, p_j). When this occurs, $|$*Proc_Ack*$_i| \geq t+1$, and so *Proc_Ack*$_i$ contains at least one correct process, say p_l. From Figure 2, p_i received from p_l a queue *Prev_Sends* that contains m. By property **L1** of fair lossy links, m is in *Prev_Sends*$_l$. By Lemma 5.1, m remains in *Prev_Sends*$_l$ forever. Since p_l is correct, from Figure 2, p_l sends *Prev_Sends*$_l$ to p_j infinitely often. Only a finite number of these queues do not contain m. By property **L5** (Fair Loss), p_j receives an infinite number of these queues. Thus, p_j eventually receives a queue that contains m. From this receipt onwards, m is in *Prev_Sends*$_j$. Since $dest(m) = p_j$, by Lemma 5.4, p_j *rel_recv*s m — a contradiction to our assumption that p_j does not *rel_recv* m. Thus, case (2) cannot occur. $\Box_{Theorem\ 5.5}$

This completes our informal proof that when $n > 2t$, the *rel_send* and *rel_recv* procedures in Figure 2 simulate reliable links using fair lossy links. A more precise statement of this result is postponed to Section 6.7.

6 Simulation and Translation: Model and Results

In the previous sections we have informally proved that: (1) in general, eventually reliable links cannot simulate reliable links, (2) when $n > t$, fair lossy links can simulate weakly reliable links, and (3) when $n > 2t$, fair lossy links can simulate reliable links. To state these results more precisely, we refine our model and define the notions of simulation and translation [NT90].

6.1 Augmentation

The state of a process that simulates another one has two components: the *simulated variables* (all the variables of the simulated process) and the *simulation variables* (some bookkeeping variables that are used to carry out the simulation). These two sets of variables are disjoint. So if p' simulates p, the variables of p' include those of p. To formalize this, we introduce the following definitions.

We say that state s' *augments state* s, $s' \geq_a s$, if the set of variables of s' includes all the variables of s, and the variables of s have the same value in s and s'. Formally, $s' \geq_a s$ if $var(s') \supseteq var(s)$ and for all $v \in var(s)$, $s(v) = s(v')$.

Let s_1 and s_2 be two states over disjoint sets of variables V_1 and V_2. Then $s = (s_1, s_2)$ denotes the state over $V_1 \cup V_2$, such that $\forall v \in V_1, s(v) = s_1(v)$ and $\forall v \in V_2, s(v) = s_2(v)$. Note that if $s' \geq_a s$ then $s' = (s, r)$ for some state r over $var(s') \setminus var(s)$.

We extend the notion of augmentation to sequences of states, and then to vectors of sequences of states (i.e., to traces), in the natural way. Recall that S is the set of all possible states. For any two sequences of states σ and σ' in $Seq(S)$, we say that σ' *augments* σ, and write $\sigma' \geq_a \sigma$, if they have the same length, and every element s' in σ' augments the corresponding element s in σ.

6.2 Stuttering

When a process simulates another one, it may execute several actions to simulate a single action of the simulated process. Thus, a simulation "stretches" the trace of the simulated process: a segment $s_1\ s_2$ of a trace can be stretched into some "stuttering" version $s_1\ \cdots\ s_1\ s_2\ \cdots\ s_2$ [Lam83]. For any two sequences of states σ and σ', we say that σ' *is a stuttering of* σ, and write $\sigma' \geq_s \sigma$, if (a) either both σ' and σ are infinite or they are both finite, and (b) σ' can be obtained from σ by repeated applications of the following operation: for any state s in σ, replace s by any non-empty finite sequence of the form $s\ \cdots\ s$.

6.3 Specifications Closed under Stuttering and Augmentation

As we saw, simulation leads to both stuttering *and* augmentation: the trace of the simulating process is a stuttered and augmented version of the trace of the simulated process. For any two sequences of states σ and σ' in $Seq(S)$, we write $\sigma' \geq_{sa} \sigma$ if there is a sequence $\sigma_0 \in Seq(S)$ such that $\sigma' \geq_a \sigma_0$ and $\sigma_0 \geq_s \sigma$. Similarly, for any two traces $\overline{H'}$ and $\overline{H}$ in $Vec(S)$, we write $\overline{H'} \geq_{sa} \overline{H}$ if they have the same dimension, say k, and for all $1 \leq i \leq k$, $\overline{H'}[i] \geq_{sa} \overline{H}[i]$. Finally, for all $\overline{\mathcal{S}},\ \overline{\mathcal{S}'} \in \Sigma^*$, we say that $\overline{\mathcal{S}'}$ *is a stuttered and augmented version of* $\overline{\mathcal{S}}$, and write $\overline{\mathcal{S}'} \geq_{sa} \overline{\mathcal{S}}$, if there is a mapping τ from $\overline{\mathcal{S}'}$ *onto* $\overline{\mathcal{S}}$ such that $\forall \overline{H'} \in \overline{\mathcal{S}'}, \overline{H'} \geq_{sa} \tau(\overline{H'})$. Note that all the $\geq_{sa}$ relations defined above are transitive.

We focus on problem specifications that are insensitive to stuttering (i.e., state repetitions) and augmentation (i.e., state extensions). Formally, a specification Σ *is closed under stuttering and augmentation* if:

$$\forall \overline{\mathcal{S}},\ \overline{\mathcal{S}'} \in \Sigma^*\ :\ \overline{\mathcal{S}'} \geq_{sa} \overline{\mathcal{S}} \text{ and } \overline{\mathcal{S}} \in \Sigma \text{ implies } \overline{\mathcal{S}'} \in \Sigma.$$

Many natural problems, including Consensus and URB (and its weaker version described in Section 3), have specifications in this class.

6.4 Simulation and Translation

Let P and P' be any two sets of processes, and $\mathcal{S} = \mathcal{S}(P)$ and $\mathcal{S}' = \mathcal{S}'(P')$ denote any two systems of P and P', respectively. Intuitively, $\mathcal{S}'$ simulates $\mathcal{S}$ if the traces in $\overline{\mathcal{S}'}$ are stuttered and augmented versions of a subset of the traces in $\overline{\mathcal{S}}$ that has the same initial states as $\overline{\mathcal{S}}$. In other words, $\mathcal{S}'$ simulates $\mathcal{S}$ if $\overline{\mathcal{S}'}$ is a stuttered and augmented version of a non-trivial reduction $\overline{\mathcal{S}_0}$ of $\overline{\mathcal{S}}$. Formally, $\mathcal{S}'$ *simulates* $\mathcal{S}$ iff:

$$\exists \mathcal{S}_0 \subseteq \mathcal{S} : \mathit{init}(\overline{\mathcal{S}_0}) = \mathit{init}(\overline{\mathcal{S}}) \text{ and } \overline{\mathcal{S}'} \geq_{sa} \overline{\mathcal{S}_0}.$$

Note that $\overline{H'} \geq_{sa} \overline{H}$ implies $\mathit{correct}(H') = \mathit{correct}(H)$. Thus, a simulation does not crash any process or mask any process failures. Moreover, it can be shown that the "simulates" relation is transitive.

Observation 6.1 *Let Σ be any specification closed under stuttering and augmentation. If $\mathcal{S}'$ simulates $\mathcal{S}$ and $\mathcal{S}$ solves Σ, then $\mathcal{S}'$ also solves Σ.*

Proof: Since $\mathcal{S}'$ simulates $\mathcal{S}$, $\overline{\mathcal{S}'} \geq_{sa} \overline{\mathcal{S}_0}$ for some non-trivial reduction $\overline{\mathcal{S}_0}$ of $\overline{\mathcal{S}}$. Thus, $\overline{\mathcal{S}} \in \Sigma$ implies that $\overline{\mathcal{S}_0} \in \Sigma$. Since $\overline{\mathcal{S}'} \geq_{sa} \overline{\mathcal{S}_0}$, and Σ is closed under stuttering and augmentation, $\overline{\mathcal{S}'} \in \Sigma$. $\Box_{Observation\ 6.1}$

For any set P of processes, we use the notation $\mathcal{S}^t_X(P)$ where $X \in \{R, WR, ER, FL\}$ to denote any one of the systems $\mathcal{S}^t_R(P)$, $\mathcal{S}^t_{WR}(P)$, $\mathcal{S}^t_{ER}(P)$, and $\mathcal{S}^t_{FL}(P)$. A *translation from X links to Y links for systems with n processes and at most t crashes*, denoted $X \stackrel{n,t}{\rightarrow} Y$, is a translation function $\mathcal{T}$ that maps any set P of n processes into a set $P' = \mathcal{T}(P)$ of n processes such that $\mathcal{S}^t_Y(P')$ simulates $\mathcal{S}^t_X(P)$.

6.5 Impossibility of Translation $R \stackrel{n,t}{\rightarrow} ER$ for $n \leq 2t$

Consider a system with at least two processes where a majority of processes may crash (i.e., $2 \leq n \leq 2t$). In Section 3, we stated that the problem of Weak Uniform Reliable Broadcast can be solved with reliable links but not with eventually reliable links (cf. Theorem A.1 in the Appendix). This implies that reliable links cannot be simulated with eventually reliable links. More precisely:

Theorem 6.2 *For $2 \leq n \leq 2t$, there is no translation $R \stackrel{n,t}{\rightarrow} ER$.*

Proof: For contradiction, suppose there is a translation $R \stackrel{n,t}{\rightarrow} ER$ for some n and t, $2 \leq n \leq 2t$. Let $\mathcal{T}$ be the translation function of $R \stackrel{n,t}{\rightarrow} ER$. As noted earlier, Σ^n_{WURB} (the specification of Weak Uniform Reliable Broadcast for n processes) is closed under stuttering and augmentation. By Theorem 3.1(1), there is a set of processes P such that $\mathcal{S}^t_R(P)$ solves Σ^n_{WURB}. Let $P' = \mathcal{T}(P)$. By the definition of translation, $\mathcal{S}^t_{ER}(P')$ simulates $\mathcal{S}^t_R(P)$. By Observation 6.1, $\mathcal{S}^t_{ER}(P')$ also solves Σ^n_{WURB} — a contradiction to Theorem 3.1(2). $\Box_{Theorem\ 6.2}$

6.6 Translation $WR \stackrel{n,t}{\rightarrow} FL$

In Section 4.3, we informally showed that two procedures, *wr_send* and *wr_recv*, can be used to simulate weakly reliable links using fair lossy links. Based on these procedures, we can define a translation $\mathcal{T} = WR \stackrel{n,t}{\rightarrow} FL$

that maps any set of n processes P into a set of n processes $P' = \mathcal{T}(P)$ such that $\mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S}^t_{WR}(P)$. Roughly speaking, P' is obtained from P by replacing the send and receive actions of processes in P, with the actions of the *wr_send* and *wr_recv* procedures, respectively. A precise description of the mapping from P to P' that defines the translation $\mathcal{T}$, together with a proof of correctness, is given in the Appendix (cf. Figure 4 and Theorem A.18). We can now state our main result:

Theorem 6.3 *Let Σ be any specification that is correct-restricted, and closed under stuttering and augmentation. Let $\mathcal{T}$ be the translation referred to above. For any set of processes P, if $\mathcal{S}^t_R(P)$ solves Σ then $\mathcal{S}^t_{FL}(P')$ solves Σ where $P' = \mathcal{T}(P)$.*

Proof: Suppose $\mathcal{S}^t_R(P)$ solves Σ. Since Σ is correct-restricted, Theorem 4.2 implies that $\mathcal{S}^t_{WR}(P)$ also solves Σ. Let $P' = \mathcal{T}(P)$. By the definition of $WR \overset{n,t}{\to} FL$, $\mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S}^t_{WR}(P)$. Since $\mathcal{S}^t_{WR}(P)$ solves Σ, and Σ is closed under stuttering and augmentation, Observation 6.1 implies that $\mathcal{S}^t_{FL}(P')$ solves Σ. $\square_{Theorem\ 6.3}$

6.7 Translation $R \overset{n,t}{\to} FL$ for $n > 2t$

In Section 5, we informally proved that procedures *rel_send* and *rel_recv* can be used to simulate reliable links using fair lossy links when a majority of processes are correct. These procedures are the basis of a formal translation $\mathcal{T} = R \overset{n,t}{\to} FL$ for any $n > 2t$. Roughly speaking, P' is obtained from P by replacing the send and receive actions of P by the actions of the *rel_send* and *rel_recv* procedures, respectively. A precise description of the mapping from P to P' that defines the translation $\mathcal{T}$, together with a proof of correctness, is given in the Appendix (cf. Figure 5 and Theorem A.31).

Theorem 6.4 *Let Σ be any specification closed under stuttering and augmentation. Let $\mathcal{T}$ be the translation referred to above. For any set P of $n > 2t$ processes, if $\mathcal{S}^t_R(P)$ solves Σ then $\mathcal{S}^t_{FL}(P')$ solves Σ where $P' = \mathcal{T}(P)$.*

A Appendix

A.1 Possibility and Impossibility of Weak Uniform Reliable Broadcast

We now define Σ^n_{WURB}, the specification of $WURB$ for n processes, and prove Theorem 3.1. Specification Σ^n_{WURB} is the set of all sets of traces $\overline{\mathcal{S}} \in \Sigma^*$ that satisfy the following conditions:

1. For every trace $\overline{H} \in \overline{\mathcal{S}}$:
   (a) the dimension of $\overline{H}$ is n, and
   (b) all the local states in $\overline{H}[1]$ have variable *message*, and
   (c) for all i, $1 \leq i \leq n$, all the local states in $\overline{H}[i]$, have variable *delivery*$_i$, and
   (d) in the initial state of $\overline{H}[1]$, *message* $= 0$ or 1, and for all i, $1 \leq i \leq n$, in the initial state of $\overline{H}[i]$, *delivery*$_i = 0$, and
   (e) if *message* $= 1$ in the initial state of $\overline{H}[1]$ and $\overline{H}[1]$ is infinite, then *delivery*$_1 = 1$ in some state in $\overline{H}[1]$, and
   (f) if *delivery*$_1 = 1$ in some state in $\overline{H}[1]$, then *delivery*$_i = 1$ in some state of every infinite $\overline{H}[i]$, and
   (g) if *message* $= 0$ in the initial state of $\overline{H}[1]$, then there is no i, $1 \leq i \leq n$, such that *delivery*$_i = 1$ in some state in $\overline{H}[i]$.
2. There are (at least) two histories $\overline{H}_0$ and $\overline{H}_1$ in $\overline{\mathcal{S}}$ such that *message* $= 0$ in the initial state of $\overline{H}_0[1]$, and *message* $= 1$ in the initial state of $\overline{H}_1[1]$.

Theorem A.1 *1. For $0 \leq t < n$, there is a set of processes P such that $\mathcal{S}^t_R(P)$ solves Σ^n_{WURB}.*

2. For $2 \leq n \leq 2t$, there is no set of processes P such that $\mathcal{S}^t_{ER}(P)$ solves Σ^n_{WURB}.

Proof:

Part (1): It is easy to see that if links are reliable, the algorithm in Figure 3 solves $WURB$ for n processes and any number of process crashes. From this algorithm, for any n and t, one can formally construct a set of n processes P such that $\mathcal{S}^t_R(P)$ solves Σ^n_{WURB}. The construction is straightforward, but tedious and thus omitted.

Part (2): The proof is by a standard partitioning argument. For contradiction, suppose there exists n and t, $2 \leq n \leq 2t$, and a set of processes P such that $\mathcal{S}^t_{ER}(P)$ solves Σ^n_{WURB}, i.e., $\overline{\mathcal{S}^t_{ER}(P)}$ satisfies conditions (1) and (2) of this specification. Furthermore, for every i, let V_i denote the set of variables of process p_i.

From conditions (2) and (1.a), (1.b), (1.c), and (1.d), we deduce that:

1. P is a set of n processes, and
2. variable *message* is in V_1, and
3. for every i, $1 \leq i \leq n$, variable *delivery*$_i$ is in V_i, and
4. p_1 has (at least) two initial states $s^0_1,\ s^1_1 \in \mathcal{Q}^0_1$ such that $s^0_1($*message*$) = 0$ and $s^1_1($*message*$) = 1$, and $s^0_1($*delivery*$_1) = s^1_1($*delivery*$_1) = 0$, and
5. for every i, $2 \leq i \leq n$, p_i has some initial state $s_i \in \mathcal{Q}^0_i$ such that $s_i($*delivery*$_i) = 0$.

code for p_1:
  Variables
    message : initially 0 or 1
    $delivery_1$: initially 0
  if *message* $= 1$ **then**
    forall $p_i \neq p_1$ **do** *send*$(1, p_i)$
    $delivery_1 := 1$

code for all $p_i \neq p_1$:
  Variables
    $delivery_i$: initially 0
  do forever
    receive(*msg*) $\{msg = \perp$ indicates no message was received $\}$
    if *msg* $\neq \perp$ **then** $delivery_i := 1$

Figure 3: Algorithm for Weak Uniform Reliable Broadcast

Partition P into two sets of processes of size at most t each, namely, $P_1 = \{p_1, \cdots, p_k\}$ and $P_2 = \{p_{k+1}, \cdots, p_n\}$, where $k = \lfloor n/2 \rfloor$. We now construct three histories H_1, H_2, and H_3 in $\mathcal{S}^t_{ER}(P)$, such that $\overline{H}_3$ violates one of the conditions of Σ^n_{WURB} — a contradiction to the fact that $\mathcal{S}^t_{ER}(P)$ solves Σ^n_{WURB}.

Construction of H_1: Let *init*$(\overline{H}_1) =< s^1_1, s_2, \cdots, s_n >$. Starting from these initial states, we schedule all processes in P_1 to execute actions, in a round-robin order, forever: each time a process $p_i \in P_1$ is scheduled, it executes one action according to its transition relation $\mathcal{T}_i$ and changes state according to its state transition function δ_i. If p_i is in a state from which it can execute an action to receive a message, then it executes *receive*(m) where m is the first[6] message sent to p_i that p_i did not previously receive, if any exists, and executes *receive*$(\perp)$ otherwise. Note that for every $p_i \in P$, and every state $s \in \mathcal{Q}_i$, there is at least one action $a \in \mathcal{A}_i$ that such that $(s, a) \in \mathcal{T}_i$, so our construction of history H_1 never blocks. Thus, each $p_i \in P_1$ executes infinitely many actions, and $H_1[i]$ is infinite. No process in P_2 executes any action. So, *correct*$(H_1) = P_1$, and at most t processes (those in P_2) crash in H_1.

From the way the construction of H_1 selects which message a process receives it is easy to see that (a) the relation $\prec_{H_1}$ is acyclic, and thus a strict partial order, and (b) H_1 satisfies the three properties of reliable links, namely **L1** (no creation), **L2** (no duplication), and **L3** (no loss). From (a), H_1 is a history of $P \in \mathcal{H}(P)$. From (b) and the fact that at most t processes crash, $H_1 \in \mathcal{S}^t_R(P) \subseteq \mathcal{S}^t_{ER}(P)$.

Since $\mathcal{S}^t_{ER}(P)$ solves Σ^n_{WURB}, $\overline{H}_1$ satisfies conditions (1.e) of Σ^n_{WURB}: in other words, since s^1_1(*message*) $=$ 1 and $\overline{H}_1[1]$ is infinite, $delivery_1 = 1$ in some local state in $\overline{H}_1[1]$. Suppose the first such state occurs after the l-th event of p_1 in H_1. For each $p_i \in P_1$, let $H^l_1[i]$ be the prefix of $H_1[i]$ such that p_i executes exactly l actions. Note that the history $< H^l_1[1], \cdots, H^l_1[k], H_1[k+1], \cdots, H_1[n] >$ is a down-set of H_1 that also satisfies properties **L1**, **L2**, and **L3** of reliable links.

Construction of H_2: Let *init*$(\overline{H}_2) =< s^0_1, s_2, \cdots, s_n >$. Starting from these local initial states, we schedule all processes in P_2 to execute actions, in a round-robin order, forever. The message receipt policy is as in the construction of H_1. No process in P_1 executes any action. This scheduling constructs a history $H_2 \in \mathcal{H}(P)$ such that *correct*$(H_2) = P_2$, at most t processes crash (all those in P_1), and H_2 satisfies properties **L1**, **L2**, and **L3** of reliable links. Thus, $H_2 \in \mathcal{S}^t_R(P) \subseteq \mathcal{S}^t_{ER}(P)$.

[6] In the round-robin scheduling.

Since $\mathcal{S}^t_{ER}(P)$ solves Σ^n_{WURB}, $\overline{H}_2$ satisfies conditions (1.g) of Σ^n_{WURB}: in other words, since $s^0_1(message) = 0$, there is no $p_i \in P_2$ that has $delivery_i = 1$ in some state in $\overline{H}_2[i]$.

Construction of H_3: Define H_3 as follows:

$$H_3[i] = \begin{cases} H^l_1[i] & \text{if } p_i \in P_1 \\ H_2[i] & \text{if } p_i \in P_2 \end{cases}$$

History H_3 is one in which (a) processes in P_1 execute exactly as in H_1 for their first l actions, and then crash, and (b) processes in P_2 execute exactly as in H_2. So $correct(H_3) = P_2$.

We claim that $H_3 \in \mathcal{S}^t_{ER}(P)$. It is clear that $H_3 \in \mathcal{H}(P)$. Moreover, at most t processes (those in P_1) crash in H_3. Since H_1 and H_2 satisfy **L1** (no creation) and **L2** (no duplication), it is easy to see that H_3 also satisfy those properties. It remains to show that H_3 satisfies **L4** (finite loss): for any $p_i, p_j \in P$, if p_j executes receive actions infinitely often, then the number of messages sent by p_i to p_j that are not received by p_j is finite. There are three possible cases:

1. $p_j \in P_1$: H_3 satisfies **L4** because p_j crashes in H_3 and does not execute receive actions infinitely often.
2. $p_i \in P_1$: H_3 satisfies **L4** because p_i can send at most l messages to p_j before it crashes in H_3.
3. $p_i, p_j \in P_2$: In this case, $H_3[i] = H_2[i]$ and $H_3[j] = H_2[j]$. So the set of messages sent by p_i to p_j that are not received by p_j is the same in both H_3 and H_2. Moreover, p_j executes receive actions infinitely often in H_3 iff it does so in H_2. Since H_2 satisfies **L2** (no loss), H_3 satisfies **L4**.

Thus, $H_3 \in \mathcal{S}^t_{ER}(P)$. However, $\overline{H}_3$ violates condition (1.f) of Σ^n_{WURB}: in fact, $delivery_1 = 1$ in some state of $\overline{H}_3[1]$ (because $H_3[1] = H^l_1[1]$), but for every $p_i \in P_2$, $\overline{H}_3[i]$ is infinite and no state in $\overline{H}_3[i]$ has $delivery_i = 1$ (because $H_3[i] = H_2[i]$). $\Box_{Theorem\ A.1}$

A.2 Translation $WR \overset{n,t}{\rightarrow} FL$

Typically, a process p'_i simulates the execution of an action $a \in \mathcal{A}_i$ of a process p_i by executing a sequence of actions in $\mathcal{A}'_i$. If p'_i crashes before completing this sequence, the simulation of a is interrupted and does not complete. In other words, simulated actions are not necessarily "atomic". To model this, we now allow a finite local history (i.e., the history of a process that crashes) to end with an event that is not followed by a state — this event corresponds to an action interrupted by a crash.

We now describe a translation $\mathcal{T} = WR \overset{n,t}{\rightarrow} FL$ for any n and t, $0 \leq t < n$. We do so by showing how to map any set of processes $P = \{p_1, p_2, \cdots, p_n\}$ into a set of processes $P' = \{p'_1, p'_2, \cdots, p'_n\}$ such that $\mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S}^t_{WR}(P)$. Figure 4 explains how to map each p_i into a process p'_i (for every i, $1 \leq i \leq n$). Recall that p_i is defined by $\mathcal{Q}_i$, $\mathcal{Q}^0_i$, $\mathcal{A}_i$, δ_i and $\mathcal{T}_i$ a set of states, a set of initial states, a set of actions, a state transition function and a transition relation, respectively. Figure 4 shows the algorithm that p'_i executes to simulate p_i. From this algorithm it is straightforward to derive the sets $\mathcal{Q}'_i$, $\mathcal{Q}'^0_i$, $\mathcal{A}'_i$, the function δ'_i, and the relation $\mathcal{T}'_i$ that formally define process p'_i. The formal definition of p'_i is tedious and is omitted here.

Consider any history $H' \in \mathcal{S}^t_{FL}(P')$, and let $p'_i \in P'$. An *invocation* event of p'_i in $H'[i]$ is one that corresponds to the execution of an action annotated *inv*$(-)$ in Figure 4. Similarly, a *return* event in $H'[i]$ is one that corresponds to an action of p'_i annotated *ret*$(-)$ in Figure 4. Intuitively, invocation and return events of p'_i in $H'[i]$ denote the beginning and end of the simulation of an event of p_i in $H[i]$. Note that p'_i updates the simulated state s of p_i immediately after each return event, and only then. The simulated state remains unchanged when a non-return event occurs. Furthermore, p'_i broadcasts its queue *Prev_Sends*$_i$ after each return event.

simulation variables
Prev_Sends$_i$: a queue of messages

simulated variables
$v_1, \cdots, v_k$: all the variables of p_i $\{V_i = \{v_1, v_2, \cdots, v_k\}\}$

{In this algorithm, "variable" s represents the simulated state of p_i, }
{i.e., s is a shorthand for variables $v_1, \cdots, v_k$ with their current values. }
{Assignment $s := s'$ is a shorthand for multiple assignment $v_1, \cdots, v_k := s'(v_1), \cdots, s'(v_k)$ }

initial state {initial values of variables}
s is some initial state $s_i^0 \in \mathcal{Q}_i^0$ {select any initial state of p_i}
Prev_Sends$_i$ is the empty queue

do forever
a := an action such that $(s, a) \in \mathcal{T}_i$ {select any action that p_i can execute in state s}

case(a) {p_i' simulates action a of p_i which can be internal, or a send, or a receive}

□ a is an internal action: {p_i' simulates internal action a of p_i}
$s := \delta_i(s, a)$ {*inv*(a) and *ret*(a)}

□ a is a *send*(m, p_j) action: {p_i' simulates a send action of p_i}
append m to *Prev_Sends$_i$* {*inv*(*send*(m, p_j))}
$s := \delta_i(s, \textit{send}(m, p_j))$ {*ret*(*send*(m, p_j))}

□ a is a *receive* action: {p_i' simulates a receive action of p_i}
receive(*Prev_Sends*) {*inv*(*receive*(m))}
if *Prev_Sends* $\neq \perp$ **then** append *Prev_Sends* to *Prev_Sends$_i$*
if *Prev_Sends$_i$* has a message m' such that
dest$(m') = p_i$ and p_i' did not previously simulate *receive*(m') by p_i
then m := first such message in *Prev_Sends$_i$*
else $m := \perp$
$s := \delta_i(s, \textit{receive}(m))$ {*ret*(*receive*(m))}

for $j = 1, \cdots, n$ **do** *send*(*Prev_Sends$_i$*, p_j') {broadcast of *Prev_Sends$_i$*}

Figure 4: Process p_i' simulating process p_i

We have to show that $\mathcal{S}' = \mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S} = \mathcal{S}^t_{WR}(P)$, i.e.,

$$\exists \mathcal{S}_0 \subseteq \mathcal{S} : \mathit{init}(\overline{\mathcal{S}_0}) = \mathit{init}(\overline{\mathcal{S}}) \text{ and } \overline{\mathcal{S}'} \geq_{sa} \overline{\mathcal{S}_0}.$$

We do so by constructing a mapping $\gamma : \mathcal{S}' \to \mathcal{S}$ with the following properties:

- **[S1]** Let $\mathcal{S}_0 = \gamma(\mathcal{S}')$ be the image of $\mathcal{S}'$ under γ. Then $\mathit{init}(\overline{\mathcal{S}_0}) = \mathit{init}(\overline{\mathcal{S}})$.
- **[S2]** For all $H' \in \mathcal{S}'$, history $H = \gamma(H')$ is such that $\overline{H'} \geq_{sa} \overline{H}$.

Consider any history $H' \in \mathcal{S}'$. Let p'_i be any process in P'. From Figure 4, any state s' of p'_i in $H'[i]$ is such that $s' = (s, r)$ where s is a state of p_i, and we define $\mathit{sim}(s') = s$.

We now describe a construction that takes $H' \in \mathcal{S}'$ and produces a history H with an associated relation $\prec_H$, and then show that this construction is a mapping γ from $\mathcal{S}'$ to $\mathcal{S}$ that satisfies properties **[S1]** and **[S2]**. For any local history $H'[i]$ construct $H[i]$ as follows:

1. History $H[i]$ starts with $s_i^0 = \mathit{sim}(s_i'^0)$ where $s_i'^0$ is the initial state of $H'[i]$.
2. Extract from $H'[i]$ the sequence σ consisting of all the *return events* and the states that immediately follow them. If $\mathit{ret}(a)$ is the k-th event in σ, then event (p_i, a, k) is the k-th event of $H[i]$. If $\mathit{ret}(a)$ is immediately followed by state s'_a in σ, then (p_i, a, k) is immediately followed by state $s_a = \mathit{sim}(s'_a)$ in $H[i]$.
3. If $H'[i]$ has an invocation event of the form $\mathit{inv}(\mathit{send}(m, p_j))$ but has no event of the form $\mathit{ret}(\mathit{send}(m, p_j))$,[7] and $\mathit{ret}(\mathit{receive}(m))$ is in $H'[j]$, then event $(p_i, \mathit{send}(m, p_j), k)$ is the last element in $H[i]$.[8]

Note that $H[i]$ is finite if and only if $H'[i]$ is finite.

We define the relations $\prec_H$ and $\preceq_H$ over events in H exactly as in Section 2.4. This completes the construction of H and its associated $\prec_H$.

For brevity, from now on, we denote an event $(p_i, \mathit{send}(m, p_j), k)$ of $H[i]$ simply as $\mathit{send}(m, p_j)$. Similarly, $(p_i, \mathit{receive}(m), k)$ is denoted $\mathit{receive}(m)$. This notation preserves the uniqueness of events because in each history, messages are unique.

Consider any history $H' \in \mathcal{S}^t_{FL}(P')$ and the history H obtained from H' by the above construction.

Lemma A.2 $\overline{H'} \geq_{sa} \overline{H}$.

Proof: (Sketch) We must show that for every $p_i \in P$, $\overline{H'}[i] \geq_{sa} \overline{H}[i]$. From Figure 4, process p'_i changes the (simulated) state s of p_i only after each return event in $H'[i]$. In other words, between every two consecutive return events of p'_i in $H'[i]$, the simulated state s of p_i remains the same (i.e., stutters). The result now follows from the construction of H. $\Box_{Lemma\ A.2}$

We now prove that H is in $\mathcal{S} = \mathcal{S}^t_{WR}(P)$. This implies that the construction is indeed a mapping from $\mathcal{S}'$ to $\mathcal{S}$. To show $H \in \mathcal{S}^t_{WR}(P)$, we must prove that (a) H is a history of P, (b) the relation $\prec_H$ is a strict partial order (and so $H \in \mathcal{H}(P)$), (c) at most t processes crash in H, and (d) H satisfies properties **L1**, **L2**, and **L6** of weakly reliable links.

[7] This can occur only if the history $H'[i]$ is finite, i.e., if p'_i crashes.

[8] Index k denotes the total number of events in $H[i]$.

Lemma A.3 *H is a history of P.*

Proof: This is immediate from the algorithm in Figure 4 and the construction of H. $\Box_{Lemma\ A.3}$

We now show that the relation $\prec_H$ is a strict partial order, so $H \in \mathcal{H}(P)$. To do so, we first prove some technical lemmas which require the following definition: for any process p'_j and any event g in $H'[j]$, let s_g^- and s_g^+ be the local states of p'_j immediately before and after event g in $H'[j]$ (if g is not followed by a local state in $H'[j]$, s_g^+ is defined as the local state that would have occurred after g if p'_j did not crash, i.e., $s_g^+ = \delta'_j(s_g^-, a)$ where a is the action of event g). We denote by $PS^-(g)$ and $PS^+(g)$ the values of the queue *Prev_Sends*$_j$ in s_g^- and s_g^+, respectively.

Lemma A.4 *Suppose* receive(m) *is in H, and let* sender$(m) = p_j$ *and* dest$(m) = p_i$.

1. inv(send(m, p_i)) *is in $H'[j]$ and* inv(receive(m)) *is in $H'[i]$, and*
2. inv(send(m, p_i)) $\prec_{H'}$ inv(receive(m)).

Proof: From Figure 4 and the construction of H, since *dest*$(m) = p_i$, *receive*(m) is in $H[i]$. Moreover, $H'[i]$ has two corresponding events: *inv*(*receive*(m)) and *ret*(*receive*(m)). From Figure 4, it is clear that m is in PS^+(*ret*(*receive*(m))). Consider the set $\Lambda = \{g \mid g$ is an event of H' and $m \in PS^+(g)\}$. This set is non-empty since *ret*(*receive*(m)) $\in \Lambda$. Let e be an event in Λ such that no event x in Λ occurs before e according to $\prec_{H'}$.

Claim A.5 *Event e is unique, $e =$* inv(send(m, p_i)), *and, for all events g in Λ, $e \preceq_{H'} g$.*

Proof: Assume e occurs in $H'[k]$ for some k. Since *Prev_Sends*$_k$ is initially empty, from the definition of e we have $m \notin PS^-(e)$ and $m \in PS^+(e)$. Thus, e corresponds to an action that updates *Prev_Sends*$_k$ (and results in the insertion of m). From Figure 4, it is now clear that e corresponds to either (a) the action that appends m to *Prev_Sends*$_k$ when p'_k simulates the action *send*(m, p_i) of p_k, or (b) the action that appends a queue *Prev_Sends* (that contains m) to *Prev_Sends*$_k$.

In case (a), it must be that $e =$ *inv*(*send*(m, p_i)), and that *sender*$(m) = p_k$ and so $k = j$.

In case (b), there is an event r corresponding to the action *receive*(*Prev_Sends*) and $r \preceq_{H'} e$. By property **L1** of H', event x corresponding to the action *send*(*Prev_Sends*$, p'_k$) occurs in H'. Since $m \in$ *Prev_Sends*, $m \in PS^+(x)$, and so $x \in \Lambda$. Moreover, $x \prec_{H'} r$, and by transitivity $x \prec_{H'} e$. This contradicts the definition of e, and so case (b) cannot occur.

Thus, $e =$ *inv*(*send*(m, p_i)) and e occurs in $H'[j]$. Since event *inv*(*send*(m, p_i)) cannot occur twice in H',[9] event e is unique. From the definition of e, we conclude that for all events g in Λ, $e \preceq_{H'} g$. $\Box_{Claim\ A.5}$

From the above, *inv*(*send*(m, p_i)) is in $H'[j]$ and *inv*(*receive*(m)) is in $H'[i]$ — concluding the proof of part (1) of the lemma. We now show part (2).

Claim A.6 *There is an event g in Λ such that $g \prec_{H'}$* inv(receive(m)).

Proof: Let $r =$ *inv*(*receive*(m)). Recall that r occurs in $H'[i]$. There are two cases:

1. $m \in PS^-(r)$. Since *Prev_Sends*$_i$ is initially empty, there is an event g in $H'[i]$ such that $g \prec_{H'} r$ and $m \in PS^+(g)$ (and so $g \in \Lambda$).

[9] Recall that δ_i and $\mathcal{T}_i$ ensure that each message m in H is uniquely tagged.

2. $m \notin PS^-(r)$. Since $m \in PS^-(\mathit{ret}(\mathit{receive}(m)))$, m is inserted in *Prev_Sends*$_i$ between events r and $\mathit{ret}(\mathit{receive}(m))$. From Figure 4, this must occur when p'_i appends a queue *Prev_Sends* containing m to *Prev_Sends*$_i$. Note that event r corresponds to the *receive*(*Prev_Sends*) action (that is executed just before p'_i appends *Prev_Sends* to *Prev_Sends*$_i$). By property **L1** of H', history H' must have an event g corresponding to the action *send*(*Prev_Sends*, p'_i). Since $m \in$ *Prev_Sends*, $m \in PS^+(g)$, and so $g \in \Lambda$. Moreover, $g \prec_{H'} r$. $\Box_{Claim\ A.6}$

By Claims A.5 and A.6, there is an event g in Λ such that $e \preceq_{H'} g \prec_{H'} \mathit{inv}(\mathit{receive}(m))$, where $e = \mathit{inv}(\mathit{send}(m, p_i))$. This concludes the proof of part (2) of the lemma. $\Box_{Lemma\ A.4}$

Lemma A.7 *The relation $\prec_H$ is a strict partial order.*

Proof: By construction, $\prec_H$ is transitive. It remains to show that it is acyclic.

Claim A.8 *For any pair of events e_1, e_2 in H the following holds:*

$$e_1 \prec_H e_2 \Rightarrow \mathrm{inv}(e_1) \prec_{H'} \mathrm{inv}(e_2) \tag{1}$$

Proof: If e_1 and e_2 are in the same local history, (1) follows directly from Figure 4, our construction of H, and the definition of $\prec_H$. If $e_1 = \mathit{send}(m, p_j)$ and $e_2 = \mathit{receive}(m)$, then $\mathit{dest}(m) = p_j$, and (1) follows from Lemma A.4. The claim now follows from the transitivity of $\prec_H$ and $\prec_{H'}$. $\Box_{Claim\ A.8}$

Assume, for contradiction, that $\prec_H$ has a cycle. Since $\prec_H$ is transitive, there must exist some event e_1 in H such that $e_1 \prec_H e_1$. By Claim A.8, $\mathit{inv}(e_1) \prec_{H'} \mathit{inv}(e_1)$ — a contradiction to the fact that $\prec_{H'}$ is acyclic. $\Box_{Lemma\ A.7}$

By Lemmas A.3 and A.7, H is a history of P and $\prec_H$ is a strict partial order. Thus, $H \in \mathcal{H}(P)$.

Lemma A.9 *At most t processes crash in H.*

Proof: Immediate from the following facts: (a) at most t processes crash in H' (because $H' \in \mathcal{S}^t_{FL}(P')$, and (b) for every i, $1 \le i \le n$, $H[i]$ is finite if and only if $H'[i]$ is finite, thus p_i is correct in $H[i]$ if and only if p'_i is correct in $H'[i]$. $\Box_{Lemma\ A.9}$

To prove that $H \in \mathcal{S}^t_{WR}(P)$ it remains to show that H satisfies the properties of weakly reliable links.

Lemma A.10 *For every process $p'_i \in P'$, the queue* Prev_Sends$_i$ *is non-decreasing in $H'[i]$.*

Proof: Obvious from the way p'_i maintains *Prev_Sends*$_i$ in Figure 4. $\Box_{Lemma\ A.10}$

Lemma A.11 *Suppose $e = \mathrm{inv}(\mathrm{send}(m, \mathrm{dest}(m)))$ is in H'. For all events g in H', if $PS^+(g)$ contains m then $PS^+(g)$ also contains $PS^+(e)$.*

Proof: Let $\Lambda' = \{g \mid g$ is an event in H' and $PS^+(g)$ contains m but does not contain $PS^+(e)\}$. For contradiction, suppose that Λ' is not empty. Let e' be an event in Λ' such that no event x in Λ' occurs before e' according to $\prec_{H'}$. Suppose e' occurs in $H'[j]$ for some process p'_j.

By the definition of e', the monotonicity of *Prev_Sends*$_j$ (Lemma A.10), and the fact that *Prev_Sends*$_j$ was initially empty, we must have $m \notin PS^-(e')$ and $m \in PS^+(e')$. Thus, e' corresponds to an action that updates *Prev_Sends*$_j$ (and results in the insertion of m). Since $e \notin \Lambda'$ (by definition of Λ'), $e' \neq e$. Therefore,

e' corresponds to the action that appends a queue *Prev_Sends* to *Prev_Sends*$_j$ during the simulation of some receive action. Note that *Prev_Sends* contains m but does not contain $PS^+(e)$.

We can now proceed as in case (b) of Claim A.5 to show that H' has an event x corresponding to the action *send*(*Prev_Sends*, p_j), and $x \prec_{H'} e'$. Since *Prev_Sends* contains m but not $PS^+(e)$, then $PS^+(x)$ contains m but not $PS^+(e)$, and so $x \in \Lambda'$ — contradicting the definition of e'. $\Box_{Lemma\ A.11}$

Corollary A.12 *Suppose events* inv(send(m, dest(m))) *and* ret(receive(m)) *are in* H'. *Then* PS^+(ret(receive(m))) *contains* PS^+(inv(send(m, dest(m)))).

Proof: Let $g = ret(receive(m))$. Note that $m \in PS^+(g)$ and apply Lemma A.11. $\Box_{Corollary\ A.12}$

Lemma A.13 *For every process p_i that executes receive actions infinitely often in H, if m is in* Prev_Sends$_i$ *in H' and* dest(m) $= p_i$*, then p_i receives m in H.*

Proof: Since p_i executes receive actions infinitely often in H, there are an infinite number of *ret*(*receive*($-$)) events of p'_i in $H'[i]$. Thus, p'_i executes the entire sequence of actions that simulates a receive action of p_i infinitely often in H'. Every time p'_i executes such a sequence of actions in H', p_i receives in H the *first* message m' of *Prev_Sends*$_i$ that p_i has not yet received such that *dest*(m') $= p_i$. Since m is in *Prev_Sends*$_i$ and *dest*(m) $= p_i$, from the way queue *Prev_Sends*$_i$ is maintained by p'_i in H', we deduce that p_i eventually receives m in H. $\Box_{Lemma\ A.13}$

Lemma A.14 *If process p_i sends m to p_j in history H before (according to $\prec_H$) some event e of a correct process p_l in H, then m is eventually in* Prev_Sends$_l$ *in history H'.*

Proof: Since p_l is correct in H, process p'_l is correct in H'. Suppose $i = l$. In this case, p'_i appends m to *Prev_Sends*$_i$ in $H'[i]$ during its simulation of the action *send*(m, p_j) of p_i. Now suppose $i \neq l$. By hypothesis, event *send*(m, p_j) is in $H[i]$, and *send*(m, p_j) $\preceq_H e$. By the definition of the $\preceq_H$ relation (given in the construction of H) there must exist some messages $m_0, m_1, \cdots, m_{k-1}$ and processes $p_i = p_{i_0}, p_{i_1}, \cdots, p_{i_{k-1}}, p_{i_k} = p_l$ such that:

1. *send*(m, p_j) $\preceq_H$ *send*(m_0, p_{i_1}), both events are in $H[i_0]$, and
2. for $0 \leq j \leq k-1$, *send*($m_j, p_{i_{j+1}}$) $\prec_H$ *receive*(m_j), these two events occur in $H[i_j]$ and $H[i_{j+1}]$, respectively, and
3. for $1 \leq j \leq k-1$, *receive*(m_{j-1}) $\prec_H$ *send*($m_j, p_{i_{j+1}}$), both events are in $H[i_j]$, and
4. *receive*(m_{k-1}) $\preceq_H e$, both events are in $H[i_k]$.

Since e occurs at a correct process, namely, $p_{i_k} = p_l$, event *ret*(e) exists. From Lemma A.4, the definition of $\preceq_H$, and (1)-(4) above, we have:

1. *inv*(*send*(m, p_j)) $\preceq_{H'}$ *inv*(*send*(m_0, p_{i_1})), both events are in $H'[i_0]$, and
2. for $0 \leq j \leq k-1$, *inv*(*send*($m_j, p_{i_{j+1}}$)) $\prec_{H'}$ *ret*(*receive*(m_j)), these two events occur in $H'[i_j]$ and $H'[i_{j+1}]$, respectively, and
3. for $1 \leq j \leq k-1$, *ret*(*receive*(m_{j-1})) $\prec_{H'}$ *inv*(*send*($m_j, p_{i_{j+1}}$)), both events are in $H'[i_j]$, and
4. *ret*(*receive*(m_{k-1})) $\preceq_{H'}$ *ret*(e), both events are in $H'[i_k]$.

By Corollary A.12, for any message $m \in \mathcal{M}(P)$, if $inv(send(m, dest(m)))$ and $ret(receive(m))$ are in H', then $PS^+(ret(receive(m)))$ contains $PS^+(inv(send(m, dest(m))))$. From this observation, Lemma A.10, and (1)-(4) above, we conclude that:

1. $PS^+(inv(send(m_0, p_{i_1})))$ contains m, and
2. $PS^+(ret(receive(m_j))$ contains $PS^+(inv(send(m_j, p_{i_{j+1}})))$, for $0 \leq j \leq k-1$, and
3. $PS^+(inv(send(m_j, p_{i_{j+1}}))$ contains $PS^+(ret(receive(m_{j-1})))$, for $1 \leq j \leq k-1$, and
4. $PS^+(ret(e))$ contains $PS^+(ret(receive(m_{k-1}))$.

By chaining the above facts, $PS^+(ret(e))$ contains m. Since $ret(e)$ occurs at a correct process, namely p'_l, it is followed by a state. From the definition of $PS^+(ret(e))$, we conclude that m is in the queue *Prev_Sends*$_l$ of process p'_l immediately after *ret*(e) in $H'[l]$. $\Box_{Lemma\ A.14}$

Lemma A.15 *H satisfies the properties of weakly reliable links.*

Proof: We must show that for every i, j, the link from p_i to p_j is weakly reliable in H, i.e., properties **L1**, **L2** and **L6** hold.

L1 (No Creation): Suppose p_j receives m from p_i in H, i.e., event $receive(m)$ with $sender(m) = p_i$ and $dest(m) = p_j$ is in $H[j]$. By Lemma A.4, $inv(send(m, p_j))$ occurs in $H'[i]$. Since $ret(receive(m))$ also occurs in $H'[j]$ (because $receive(m)$ is in $H[j]$), by the construction of H from H', event $send(m, p_j)$ occurs in $H[i]$, i.e., p_i sends m to p_j in H.

L2 (No Duplication): From Figure 4 (and the construction of H from H'), it clear that no process $p_j \in P$ receives a message more than once in H.

L6 (No Visible Loss): Suppose p_i sends m to p_j in H before (according to $\prec_H$) some event e of a correct process p_l in H (note that $dest(m) = p_j$). From Lemma A.14, m is eventually in *Prev_Sends*$_l$ of p'_l in H'. By Lemma A.10, m remains in *Prev_Sends*$_l$ in H' forever. Consider the set $\mathcal{M}_{lj}$ of all the messages sent by p'_l to p'_j in H'. Each such message is sent when p'_l executes a $send(Prev\_Sends, p'_j)$ action, and it consists of p'_l's current value of the queue *Prev_Sends*$_l$. Since p_l is correct in H, p'_l is correct in H'. Thus, p'_l executes $send(Prev\_Sends, p'_j)$ actions infinitely often in H' and so $\mathcal{M}_{lj}$ is infinite. Note that only a finite number of messages in $\mathcal{M}_{lj}$ are *Prev_Sends*$_l$ queues that do *not* contain m.

Now assume that p_j executes receive actions infinitely often in H. By our construction of H from H', p'_j executes receive actions infinitely often in H'. Since H' satisfies **L5** (Fair Loss) and $\mathcal{M}_{lj}$ is infinite, p'_j receives an infinite subset of the messages in $\mathcal{M}_{lj}$. Thus, p'_j eventually receives (in H') a queue *Prev_Sends*$_l$ that contains m. From this receipt onwards, m is in *Prev_Sends*$_j$ in H'. Since $dest(m) = p_j$, by Lemma A.13, p_j eventually receives m in H, as we needed to show. $\Box_{Lemma\ A.15}$

This completes the proof that our construction maps every history H' in $\mathcal{S}' = \mathcal{S}^t_{FL}(P')$ into a history H in $\mathcal{S} = \mathcal{S}^t_{WR}(P)$. Let $\gamma : \mathcal{S}' \to \mathcal{S}$ denote this mapping.

Lemma A.16 *Let $\mathcal{S}_0 = \gamma(\mathcal{S}')$ be the image of $\mathcal{S}'$ under γ. Then* $\text{init}(\overline{\mathcal{S}_0}) = \text{init}(\overline{\mathcal{S}})$.

Proof: Since $\mathcal{S}_0 \subseteq \mathcal{S}$, $init(\overline{\mathcal{S}_0}) \subseteq init(\overline{\mathcal{S}})$. We now show that $init(\overline{\mathcal{S}}) \subseteq init(\overline{\mathcal{S}_0})$. Let $< s^0_1, \cdots, s^0_i, \cdots, s^0_n >$ be any element of $init(\overline{\mathcal{S}})$. Note that, for $1 \leq i \leq n$, $s^0_i \in \mathcal{Q}^0_i$ is an initial state of p_i. From Figure 4, process p'_i can start by simulating any initial state of p_i. In particular, it can start from some state s'^0_i such that $s^0_i = sim(s'^0_i)$. So $< s'^0_1, \cdots, s'^0_i, \cdots, s'^0_n >$ is in $init(\overline{\mathcal{S}'})$. From the definition of γ, $< s^0_1, \cdots, s^0_i, \cdots, s^0_n >$ is in $init(\overline{\mathcal{S}_0})$. $\Box_{Lemma\ A.16}$

Lemma A.17 $\mathcal{S}' = \mathcal{S}^t_{FL}(P')$ *simulates* $\mathcal{S} = \mathcal{S}^t_{WR}(P)$.

Proof: This follows from the existence of the mapping $\gamma : \mathcal{S}' \to \mathcal{S}$ that satisfies properties **[S1]** (Lemma A.2) and **[S2]** (Lemma A.16). $\Box_{Lemma\ A.17}$

Theorem A.18 *Figure 4 defines a translation* $\mathcal{T} = WR \stackrel{n,t}{\to} FL$ *for any* n *and* t, $0 \leq t < n$.

Proof: Figure 4 shows how to map any set of n processes P into a set of n processes P' such that $\mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S}^t_{WR}(P)$ (Lemma A.17). $\Box_{Theorem\ A.18}$

A.3 Translation $R \stackrel{n,t}{\to} FL$ for $n > 2t$

The translation $\mathcal{T} = R \stackrel{n,t}{\to} FL$ is defined by Figure 5 which shows how to map a set of processes $P = \{p_1, p_2, \cdots, p_n\}$ into a set $P' = \{p'_1, p'_2, \cdots, p'_n\}$ such that $\mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S}^t_R(P)$ when $n > 2t$. The proof that $\mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S}^t_R(P)$, i.e, $\mathcal{T}$ is indeed a translation $R \stackrel{n,t}{\to} FL$ is similar to the proof of Lemma A.17, and is outlined below.

To show that $\mathcal{S}' = \mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S} = \mathcal{S}^t_R(P)$, we construct a mapping $\gamma : \mathcal{S}' \to \mathcal{S}$ with the following properties:

- **[S1]** Let $\mathcal{S}_0 = \gamma(\mathcal{S}')$ be the image of $\mathcal{S}'$ under γ. Then *init*$(\overline{\mathcal{S}_0}) =$ *init*$(\overline{\mathcal{S}})$.
- **[S2]** For all $H' \in \mathcal{S}'$, history $H = \gamma(H')$ is such that $\overline{H'} \geq_{sa} \overline{H}$.

We map each history $H' \in \mathcal{S}'$ to a history H with its associated $\prec_H$ relation exactly as we did in Section A.2 (for the translation $WR \stackrel{n,t}{\to} FL$). The proof that this is indeed a mapping γ from $\mathcal{S}'$ to $\mathcal{S}$ that satisfies properties **[S1]** and **[S2]** is given below. We omit the proofs of all the lemmas whose proofs are the same as in Section A.2.

Lemma A.19 *For every process* $p'_i \in P'$*, the queue* Prev_Sends$_i$ *is non-decreasing in* $H'[i]$.

Lemma A.20 *For every* i , $1 \leq i \leq n$, $H[i]$ *is finite if and only if* $H'[i]$ *is finite.*

Proof: From our construction of H from H', it is obvious that for every i , $1 \leq i \leq n$, if $H'[i]$ is finite then $H[i]$ is finite. The proof that for every i, $H[i]$ is finite only if $H'[i]$ is finite is by contradiction. Suppose that for some i, $H[i]$ is finite but $H'[i]$ is infinite. From our construction, this implies that the number of return events in $H'[i]$ is finite. From Figure 5, it is clear that this can happen only if p'_i spins forever in the **while** loop during the simulation of some *send*(m, p_j) action. In this loop, p'_i sends *Prev_Sends*$_i$ (which contains m) to all processes infinitely often. From Figure 5, it is clear that each correct process executes receive actions infinitely often (even if it is itself "blocked" spinning forever in the **while** loop during the simulation of a *send* action). By property **L5** of fair lossy links, each such correct process eventually receives a *Prev_Sends*$_i$ that contains m, and from Figure 5, it appends this *Prev_Sends*$_i$ to its own *Prev_Sends* queue. By Lemma A.19 and Figure 5, each correct process sends a *Prev_Sends* queue that contains m infinitely often to p'_i. Since p'_i spins forever in the **while** loop, it executes receive actions infinitely often. By property **L5** of fair lossy links, p'_i eventually receives a *Prev_Sends* queue containing m from every correct process, i.e. from at least $t+1$ distinct processes (because $n > 2t$). Thus, p'_i does not spin forever in the **while** loop — a contradiction. $\Box_{Lemma\ A.20}$

Lemma A.21 $\overline{H'} \geq_{sa} \overline{H}$.

simulation variables
    Prev_Sends$_i$: a queue of messages
    Proc_Ack$_i$: a set of processes
simulated variables
    $v_1, \cdots, v_k$: all the variables of p_i $\{V_i = \{v_1, v_2, \cdots, v_k\}\}$

initial state {initial values of variables}
    s is some initial state $s_i^0 \in \mathcal{Q}_i^0$ {select any initial state of p_i}
    Prev_Sends$_i$ is the empty queue and *Proc_Ack*$_i$ is the empty set

do forever
    a := an action such that $(s, a) \in \mathcal{T}_i$ {select any action that p_i can execute in state s}

    case(a) $\{p_i'$ simulates action a of p_i which can be internal, or a send, or a receive$\}$

        □ a is an internal action: $\{p_i'$ simulates internal action a of $p_i\}$
            $s := \delta_i(s, a)$ $\{$*inv*(a) and *ret*$(a)\}$

        □ a is a *send*(m, p_j) action: $\{p_i'$ simulates a send action of $p_i\}$
            append m to *Prev_Sends*$_i$ $\{$*inv*$($*send*$(m, p_j))\}$
            Proc_Ack$_i$:= $\emptyset$
            while $|$*Proc_Ack*$_i| < t + 1$ {till m is echoed by at least $t + 1$ *distinct* processes}
                for $j = 1, \cdots, n$ **do** *send*(*Prev_Sends*$_i$, p_j')
                receive(*Prev_Sends*)
                if *Prev_Sends* $\neq \perp$ **then**
                    append *Prev_Sends* to *Prev_Sends*$_i$
                    if m in *Prev_Sends* **then** *Proc_Ack*$_i$:= *Proc_Ack*$_i$ $\cup$ $\{$*sender*(*Prev_Sends*)$\}$
            $s := \delta_i(s,$ *send*$(m, p_j))$ $\{$*ret*$($*send*$(m, p_j))\}$

        □ a is a *receive* action: $\{p_i'$ simulates a receive action of $p_i\}$
            if *Prev_Sends*$_i$ has a message m' such that $\{$*inv*$($*receive*$(m))\}$
                dest$(m') = p_i$ and p_i' did not previously simulate *receive*(m') by p_i
            then m := first such message in *Prev_Sends*$_i$
            else $m := \perp$
            $s := \delta_i(s,$ *receive*$(m))$ $\{$*ret*$($*receive*$(m))\}$

    for $j = 1, \cdots, n$ **do** *send*(*Prev_Sends*$_i$, p_j') {broadcast *Prev_Sends*$_i$}
    receive(*Prev_Sends*) {receive a message}
    if *Prev_Sends* $\neq \perp$ **then** append *Prev_Sends* to *Prev_Sends*$_i$

Figure 5: Process p_i' simulating process p_i

Lemma A.22 *H is a history of P.*

Lemma A.23 *Suppose* receive(m) *is in H, and let* sender$(m) = p_j$ *and* dest$(m) = p_i$.

1. inv(send(m, p_i)) *is in $H'[j]$ and* inv(receive(m)) *is in $H'[i]$, and*
2. inv(send(m, p_i)) $\prec_{H'}$ inv(receive(m)).

Proof: The proof is as in Lemma A.4, except for the proof of the claim below.

Claim A.24 *There is an event g in Λ such that $g \prec_{H'}$* inv(receive(m)).

Proof: Let $r = inv(receive(m))$ and $r' = ret(receive(m))$. Recall that both r and r' occur in $H'[i]$. From Figure 5, it is clear that m is in $PS^+(r')$. Furthermore, the queue *Prev_Sends$_i$* is not modified between events r and r' in $H'[i]$. Thus, $m \in PS^-(r)$. Since *Prev_Sends$_i$* is initially empty, there is an event g in $H'[i]$ such that $g \prec_{H'} r$ and $m \in PS^+(g)$ (and so $g \in \Lambda$). $\Box_{Claim\ A.24}$

Lemma A.25 *The relation $\prec_H$ is a strict partial order.*

By Lemmas A.22 and A.25, $H \in \mathcal{H}(P)$.

Lemma A.26 *At most t processes crash in H.*

To prove that $H \in \mathcal{S}^t_R(P)$ it remains to show that H satisfies the properties of reliable links.

Lemma A.27 *For every process p_i that executes receive actions infinitely often in H, if m is in* Prev_Sends$_i$ *in H' and* dest$(m) = p_i$*, then p_i receives m in H.*

Lemma A.28 *H satisfies the properties of reliable links.*

Proof: We must show that for every i, j, the link from p_i to p_j is weakly reliable in H, i.e., properties **L1**, **L2** and **L3** hold.

L1 (No Creation) and **L2** (No Duplication): The proof is the same as in the proof of Lemma A.15.

L3 (No Loss): Suppose that p_i sends m to p_j and p_j executes receive actions infinitely often in H. We have to show that p_j receives m in H.

Since p_i sends m to p_j in H, from our construction of H from H' there are two possible cases:

1. Event *inv(send(m, p_j))* is in $H'[i]$ and event *ret(receive(m))* is in $H'[j]$. From the construction of H from H', p_j receives m in H.
2. Event *ret(send(m, p_j))* is in $H'[i]$. When this event occurs, $|$*Proc_Ack$_i$*$| \geq t + 1$, and so *Proc_Ack$_i$* contains at least one correct process, say p'_l. From Figure 5, p'_i received from p'_l a queue *Prev_Sends* that contains m in H'. By property **L1** of fair lossy links, m is in *Prev_Sends$_l$*. By Lemma A.19, m remains in *Prev_Sends$_l$* in H' forever. The proof that p_j receives m in H now proceeds exactly as in the proof of property **L6** in Lemma A.15. $\Box_{Lemma\ A.28}$

This completes the proof that our construction maps every history H' in $\mathcal{S}' = \mathcal{S}^t_{FL}(P')$ into a history H in $\mathcal{S} = \mathcal{S}^t_R(P)$. Let $\gamma : \mathcal{S}' \to \mathcal{S}$ denote this mapping.

Lemma A.29 *Let $\mathcal{S}_0 = \gamma(\mathcal{S}')$ be the image of $\mathcal{S}'$ under γ. Then* $\mathrm{init}(\overline{\mathcal{S}_0}) = \mathrm{init}(\overline{\mathcal{S}})$.

Lemma A.30 *$\mathcal{S}' = \mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S} = \mathcal{S}^t_R(P)$.*

Proof: This follows from the existence of the mapping $\gamma : \mathcal{S}' \rightarrow \mathcal{S}$ that satisfies properties **[S1]** (Lemma A.21) and **[S2]** (Lemma A.29). $\Box_{Lemma\ A.30}$

Theorem A.31 *Figure 5 defines a translation $\mathcal{T} = WR \overset{n,t}{\rightarrow} FL$ for any n and t, $0 \leq t < n$.*

Proof: Figure 5 shows how to map any set of n processes P into a set of n processes P' such that $\mathcal{S}^t_{FL}(P')$ simulates $\mathcal{S}^t_R(P)$ (Lemma A.30). $\Box_{Theorem\ A.31}$

Acknowledgments

We are grateful to David Cooper and Vassos Hadzilacos for valuable discussions that helped us to significantly improve the presentation of the results.

References

[AAF+94] Y. Afek, H. Attiya, A. D. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. Zuck. Reliable communication over unreliable channels. *Journal of the ACM*, 41(6):1267–1297, 1994.

[ABND+90] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, 1990.

[AE86] B. Awerbuch and S. Even. Reliable broadcast protocols in unreliable networks. *Networks: An International Journal*, 16, 1986.

[AGH90] B. Awerbuch, O. Goldreich, and A. Herzberg. A quantitative approach to dynamic networks. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pages 189–204, Québec City, Québec, Canada, 1990.

[Bir85] K. Birman. Replication and fault-tolerance in the ISIS system. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 79–86, Orcas Island, WA USA, 1985.

[BN92] R. Bazzi and G. Neiger. Simulating crash failures with many faulty processors. In A. Segal and S. Zaks, editors, *Proceedings of the Sixth International Workshop on Distributed Algorithms*, volume 647 of *Lecture Notes on Computer Science*, pages 166–184. Springer-Verlag, 1992.

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

[Cha90] S. Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pages 311–324, Québec City, Québec, Canada, August 1990.

[DLP+86] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1986.

[FLMS93] A. D. Fekete, N. Lynch, Y. Mansour, and J. Spinelli. The impossibility of implementing reliable communication in the face of crashes. *Journal of the ACM*, 40(5):1087–1107, 1993.

[GA88] E. Gafni and Y. Afek. End-to-end communication in unreliable networks. In *Proceedings of the Seventh ACM Symposium on Principles of Distributed Computing*, pages 131–148, Toronto, Ontario, Canada, August 1988.

[Gop92] A. Gopal. *Fault-Tolerant Broadcasts and Multicasts: The Problem of Inconsistency and Contamination*. PhD thesis, Cornell University, January 1992.

[HT94] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR 94-1425, Cornell University, Dept. of Computer Science, Cornell University, Ithaca, NY 14853, May 1994.

[JV96] M. Jayaram and G. Varghese. Crash failures can drive protocols to arbitrary states. In *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing*, pages 247–256, May 1996.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lam83] L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83: proceedings of the IFIP Ninth World Congress*, pages 657–668. IFIP, North-Holland, September 1983.

[NT90] G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, 1990.

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