SOURCE TO DESTINATION COMMUNICATION IN THE PRESENCE OF FAULTS

by

O. Goldreich, A. Herzberg and Y. Mansour

Technical Report #563

June 1989
Source to Destination Communication in the Presence of Faults

Preliminary Version

Oded Goldreich
Department of Computer Science
Technion, Haifa, Israel

Amir Herzberg
Department of Computer Science
Technion, Haifa, Israel

Yishay Mansour
MIT - Laboratory for Computer Science
345 Technology Square, Cambridge, MA 02139

June 12, 1989

Abstract

We present a protocol for reliable communication between two processors via an unreliable, and possibly even malicious, communication media. Reliable communication means that all messages are accepted in the same order as sent, with no modifications, omissions, insertions or duplications. Our protocol is resilient to processor crashes (in which the entire memory of the processor is erased), and to duplication and reordering of messages on the link. The protocol is practical, with applications in both the data link layer and the data transport layer of the ISO model.

Our approach is probabilistic. Namely, for the worst case behavior of the underlying unreliable channel, our protocol guarantees reliable communication with very high probability. The probabilistic approach is justified in light of the impossibility result recently shown [LMF88] for protocols resilient to processor crashes.

*This exposition will be included in the proceedings of the 8th PODC.
† Partially supported by grant No. 86-00301 from the United States - Israel Binational Science Foundation (BSF).
‡ Partially supported by an ISEF fellowship and NSF contract 8657527-CCR
1 Introduction

We present a protocol for reliable transfer of messages from the higher layer in a source processor to the higher layer in a destination processor. The protocol transfers data between the source and the destination by sending packets via a semi-reliable lower layer. The reliability provided to the higher layer is very strong; intuitively, the messages are delivered exactly as sent, with no modifications, omissions, duplications or reordering. The semi-reliable lower layer is reliable in a very weak sense. Namely, it may lose packets, transfer packets in an order different from the order in which they were sent, and duplicate the same packet any number of times. However, the contents of packets are not modified. The protocol is resilient to processor crashes, in which the entire memory of the processor is erased.

The ability to communicate reliably over relatively unreliable media is of theoretical and practical importance. In particular, our results applies both to the data link layer and to the data transport layer of the ISO model [Tan81,Zim80]. Both layers are basic components of every communication network. In the data link layer, the semi-reliable media is the channel or the "physical link", and a reliable protocol is referred to as a "data link control protocol". In most of the paper, we adopt the terminology of the data link, referring to the lower layer as the channel.

The protocol is applicable also to the transport layer, where our protocol is run in the source and destination processors, in conjunction with a semi-reliable protocol run by the processors connecting them in the network. A trivial implementation for such a semi-reliable protocol is by flooding each packet; a more efficient method (in actual use) is to try to find a reliable path between the two processors and send all messages over that path, replacing the path only when an error is detected [HK89]. If there are no errors while the present message is sent, this conjunction achieves optimal communication cost. Note, however, that the communication complexity increases linearly with the number of errors while sending the message. Other works presented reliable protocols for the transport layer that operate in all processors in the network, assuming reliable communication over the links [CR87,Fin87,AGS89,AGH89]. Some of these [AG88,AMS89,AGH89] ensure bounded communication cost regardless of the number of link failures, assuming that processors never crash. However, the communication complexity per message is at least $O(|E|)$ (in [AGH89]). Hence, in many applications, the use of our protocol in conjunction with an efficient semi-reliable protocol is more efficient.

Previous theoretical works considered only deterministic protocols, and therefore did not allow any probability of error. In [LMF88] it was shown that no deterministic protocol can tolerate host crashes, even when the channel guarantees FIFO and no duplication. We redefine the task in a probabilistic manner, allowing a bounded (negligible) probability of error per message (and thus the impossibility result does not hold). Note that in reality, it is impossible to prevent all errors even if the processors do not fail [AUWY82]; this is neglected in most works (including the present), by assuming that the lower layer never modifies packets (i.e. is semi-reliable).

We present a simple and efficient randomized protocol, that ensures transfer of messages from source to destination, with predetermined (negligible) probability of error per message. Loosely speaking, regardless of the operation of the channel so far, a reliable transfer of the next message is guaranteed to succeed with high probability, even for malicious behavior of the channel. The only restriction on the behavior of the adversary is that it does not depend on the contents of the packets. Namely, packets of the same size are indistinguishable to the adversary. In many cases,
this is a reasonable assumption directly; in other cases, it could be achieved by encryption.

Our protocol uses unbounded storage, but of a different flavor than the unbounded counters used in most other protocols. The counters that are used in the protocol are reset after each successful transmission of a message. Therefore, the size of the storage depends only on the number of errors during the transmission of the present message (and not on the number of messages exchanged or number of errors so far). In addition, our counters may be reset whenever the processor crashes. This eliminates one of the main practical drawbacks of unbounded counters as used by other protocols, i.e. that their value should never be reset (i.e., the protocols are not resilient to crashes). Thus, this use of unbounded counters is not of practical concern, although its removal is a natural (theoretical) challenge.

Other Solutions

For FIFO channels, many protocols are known [Zim80,Tan81]. However, [BS88] show that these protocols are not resilient to processor crashes, and show a protocol that can tolerate processor crashes by having access to a single nonvolatile bit. Recently, [AB89] presented a self stabilizing randomized protocol (and thus can tolerate processor crashes) for FIFO channels.

For non-FIFO channels, without duplication of packets or processor crashes, [AFWZ89] present a protocol that uses fixed size packets. This protocol, although very intriguing from a theoretical point of view, is not practical, since the number of packets transmitted per message is at least the number of packets lost. Many of the difficulties that arise in our model are due to the combination of a channel that is not only non-FIFO but can also duplicate packets and an environment were processors can crash.

Another subject of research, is the tradeoff between the packet size and the number of packets required to deliver a message, for non-FIFO channels (without duplication or processor crashes). Various tradeoffs can be found in [LMF88,WZ89,MS89]. This issue is somewhat unrelated to the one in this work, since, strictly speaking, our protocol uses unbounded packets.

Organization

The paper is organized as follows. The model is described in Section 2. In Section 3 we give an overview of the protocol, while in Appendix A the formal code of the protocol is found. The analysis of the protocol is in Section 4.

2 The Model

The model is illustrated in Figure 1. In the following subsections we describe each component; in subsection 2.5 we discuss the motivations of the assumptions used in the model, and in subsection 2.6 we define correctness of a data link.

For simplicity, we assume that the operations of each module are atomic, in the sense that there is no event between the input event to a module and the resulting output actions of that module. This assumption does not effect our results.

In general, we use the terminology used in [LMF88], based on the I/O automata model presented in [LT87,Lyn87]. The sequence of events occurring in the system is called an execution.
Figure 1: The data link layer components

2.1 The Transmitting Module (TM)

The transmitting module TM describes the behavior of the transmission station. It has an interface to two communication channels, one (denoted \(C^{T\rightarrow R}\)) directed from the transmitting station to the receiving station, and the other (denoted \(C^{R\rightarrow T}\)) directed from the receiving station to the transmitting station. Let \(M\) denote the set of all possible messages.

Input actions:
- \(\text{send\_msg}(m), m \in M\)
- \(\text{receive\_pkt}^{R\rightarrow T}(p), p \in P\)
- \(\text{crash}^T\)

Output actions:
- \(\text{OK}\)
- \(\text{send\_pkt}^{T\rightarrow R}(p), p \in P\)

The \(\text{send\_msg}(m)\) action models a request by a higher layer in the transmitting station to send the message \(m\) on the data link. The \(\text{OK}\) action notifies the higher layers that the last message sent was delivered. The action \(\text{receive\_pkt}^{R\rightarrow T}\) is a receipt of a packet from the receiving station, and \(\text{send\_pkt}^{T\rightarrow R}\) is sending of a packet from the transmitting station to the receiving station. The \(\text{crash}^T\) action causes the memory of the TM to be erased and reset to some initial value.

The following input restriction guarantees that the data link does not need to buffer messages. These messages are buffered instead in the higher layer.

**Axiom 1** Let \(\beta\) be an execution of a TM. Between every two consecutive \(\text{send\_msg}\) actions in \(\beta\) there is either an \(\text{OK}\) or \(\text{crash}^T\) action.

The protocol presented in this work is oblivious of the values of the messages sent through it.
Thus, for the sake of simplicity, we assume (in Axiom 2) uniqueness of the messages delivered by the protocol. This assumption simplifies some of the definitions but is unessential to the results.

**Axiom 2** *Let β be an execution of a TM. For every message m, there is at most one send_msg(m) event in β.*

### 2.2 The Receiving Module (RM)

The receiving module RM describes the behavior of the receiving station. The RM has an interface to $CT \rightarrow R$ as the receiver, and $CT \rightarrow R$ as the transmitter.

**Input actions:**
- `receive_pkt$T \rightarrow R(p)$, $p \in P$
- `crash$R$

**Output actions:**
- `send_pkt$R \rightarrow T(p)$, $p \in P$
- `receive_msg(m), $m \in M$

The output action `receive_msg(m)` models the delivery of message m to the higher layer of the receiving station. The `crash$R$` action causes the memory of the RM to be erased and reset to some initial value.

### 2.3 Communication channel (CC)

Two unidirectional communication channels represent the capability of the processors to exchange packets. One channel, denoted $CT \rightarrow R$, is directed from the transmitting station to the receiving station. The other channel, denoted $CR \rightarrow T$, is directed from the receiving station to the transmitting station. The actions of the two communication channels are tagged by superscripts $T \rightarrow R$ and $R \rightarrow T$ respectively.

The behavior of the communication channel is completely specified in this section. The indeterminism in the scheduling of messages, as well as the faults, are modeled by the “adversary”, described in subsection 2.4. Properties such as fairness and causality are treated as restrictions on the behavior of the adversary (not of the communication channel).

Let $P \subseteq \{0,1\}^*$ be the set of all packets. The function `length` returns the length of a packet $p \in P$, i.e. the number of bits in a packet.

**Input actions:**
- `send_pkt(p), p \in P$
- `deliver_pkt(k), k \text{ is an integer}$

**Output actions:**
- `receive_pkt(p), p \in P$
- `new_pkt(k, l), k \text{ and } l \text{ are an integers}$

The `send_pkt(p)` action models the placing of packet p on the communication channel by the transmitter. The `receive_pkt(p)` represents the receipt of packet p by the receiver. The actions
new_pkt and deliver_pkt are used to pass information to and from the adversary. The next paragraph explains the meaning of those actions.

For every \( \sigma = \text{send_pkt}(p) \) action, the communication channel computes an unique identifier \( id_{\sigma} \) and generates \( \text{new_pkt}(id_{\sigma}, \text{length}(p)) \). The adversary causes the delivery of the packet \( id_{\sigma} \) by generating a \( \text{deliver_pkt}(id_{\sigma}) \) action. The communication channel responds to a \( \text{deliver_pkt}(id_{\sigma}) \) by generating a \( \text{receive_pkt}(p) \) action, such that \( p \) satisfies \( \sigma = \text{send_pkt}(p) \).

### 2.4 Adversary (ADV)

The adversary models a worst case or an even malicious scheduling of packet deliveries, packet losses and processor crashes. As such it can be thought as “trying to defeat the communication protocol”. We apologize for this suggestive use of language.

**Input actions:**
- \( \text{new_pkt}^{T \rightarrow R}(i, l) \), \( i \) and \( l \) are integers
- \( \text{new_pkt}^{R \rightarrow T}(i, l) \), \( i \) and \( l \) are integers

**Output actions:**
- \( \text{crash}^T \)
- \( \text{crash}^R \)
- \( \text{deliver_pkt}^{T \rightarrow R}(i) \), \( i \) is an integer
- \( \text{deliver_pkt}^{R \rightarrow T}(i) \), \( i \) is an integer

The action \( \text{new_pkt}^{T \rightarrow R}(i, l) \) notifies the adversary that a \( \text{send_pkt}^{T \rightarrow R} \) action occurred with identifier \( i \) and packet of length \( l \). The effect of \( \text{deliver_pkt}^{T \rightarrow R}(id(\sigma)) \) is \( \text{receive_pkt}^{T \rightarrow R}(p) \), where \( \sigma = \text{send_pkt}^{T \rightarrow R}(p) \). This enables the adversary to decide which packets are delivered and at what time are they delivered. A packet that was sent can be delivered any number of times (including none). The \( \text{crash}^T \) action causes a host crash in the transmitting station, which means that all the memory of the transmitting station is reset to an initial value. The action for \( CR^\rightarrow T \) and the receiving station have an analogue meaning to those of \( CT^\rightarrow R \) and the transmitting station.

An arbitrary adversary can disconnect the transmitting station from the receiving station. In this case, we cannot hope to pass any messages. In order to enable progress in the system we require that starting at any time, if infinitely many packets are sent then eventually one of them is delivered. An adversary that has this property is called a fair adversary. This is formalized as follows for \( CT^\rightarrow R \). An identical requirement is imposed on the communication channel \( CR^\rightarrow T \).

**Axiom 3** For any execution \( \alpha \) of ADV, if infinitely many \( \text{new_pkt}^{T \rightarrow R}(i_j, l_j) \) actions occur after \( \alpha \), then eventually a \( \text{deliver_pkt}^{T \rightarrow R}(k) \) occurs, with \( k = i_j \) for some \( j \).

### 2.5 Discussion of the Assumptions

In the specification of the communication channel, we implicitly assumed causality of packets. Namely, every packet received has been previously sent. This assumption is a simplification of reality, in which no protocol can guarantee absolute causality in the presence of noise on the link [AUWY82]. In practice, the lower layers can only guarantee certain probability of causality.

We assumed that the adversary accepts only the length and the identifier of packets. Thus, the adversary behavior is oblivious to the values of all packets of the same length. This ensures that
the adversary behavior is independent of the coin tosses of the protocol. The motivation for this restriction can be motivated in two ways, depending on the implementation of the communication channel. If the adversary is not malicious, then the assumption is natural. If the adversary is malicious, this assumption may be approximated by encrypting the packets. Note this requires that it will be impossible to identify two encryptions of the same packet.

In Axiom 2 we assumed uniqueness of the messages delivered by the protocol. The uniqueness of the messages helps us to define errors. For example, if the transmitter sends $a$ and the receiver delivers $b$ there is clearly an error. But if the transmitter sends the sequence of messages $ababa$ and the receiver accepts $bababa$, it is not clear how many errors occurred. One can argue that six (by comparing the $i$-th send to the $i$-th receive) or only two (by looking for the greatest common substring). By assuming that all the messages are unique, we avoid such a confusion. This assumption is unessential to the results.

2.6 Correctness of a Data Link

This subsection we define a reliable (randomized) data link protocol. A reliable data link protocol should satisfy both safety and liveness conditions. The safety conditions essentially require a one to one and onto monotonic mapping of messages delivered to messages sent. If the processors may crash, i.e. lose their entire memory, then such a mapping cannot exist. For example, an execution that terminates in a receive_msg followed by a crash$^R$ will have exactly the same extensions as an execution in which the receive_msg did not occur. Therefore, we approximate the mapping requirement by defining three complementary conditions. Intuitively, those conditions are:

**function:** For every receive_msg($m$) there is an unique send_msg($m$).

**one to one:** There is no more than one receive_msg($m$) (excluding those which follow a crash$^R$ event).

**monotonic and unto:** If send_msg($m$) is followed by OK, then a receive_msg($m$) occurs between them. This ensures monotonous operation (FIFO).

Formally, a pair of randomized algorithms $A = (A^t, A^r)$, is a data link protocol if the external actions of $A^t$ are the same as those of TM and the external actions of $A^r$ are the same as RM. This means that $A$ is the combination of a TM and a RM. Let $D(A, ADV)$ denote the composition of $A^t$, $A^r$, $ADV$, $CT^R$ and $CR^T$ (as illustrated in Figure 1). Before giving the definitions of the correctness conditions, we define a few types of extensions of an execution $\alpha$ of $D(A, ADV)$.

An extension of $\alpha$ is a sequence of actions $\beta$ s.t. $\alpha \beta$ is an execution of $D(A, ADV)$.

A receive-extension of $\alpha$ is an extension $\beta = \gamma \sigma$, such that $\sigma$ is a receive_msg event and $\gamma$ does not include any receive_msg or crash$^R$ events.

An OK-extension of $\alpha$ is an extension $\beta = \gamma \sigma$, such that $\sigma$ is an OK event and $\gamma$ does not include any OK, send_msg or crash$^T$ events.
Using the definitions above we define probabilistic correctness conditions for a data link protocol $A$. The conditions are stated with respect to a security parameter $\epsilon$ bounding the "error probability". The probability distribution is taken over the internal coin tosses of the adversary (in case it is probabilistic) and the internal coin tosses of the protocol $A$. The conditions are specified for any fixed adversary $ADV$.

**causality:** Let $\alpha$ be an execution of $D(A, ADV)$. For every receive extension of $\alpha$, $\beta = \gamma receive\_msg(m)$, there is a unique $send\_msg(m)$ in $\alpha \beta$.

**order:** Let $\alpha$ be an execution of $D(A, ADV)$ that terminates in $send\_msg(m)$. Let $\Omega$ be a random variable assuming as values $OK$ extensions of $\alpha$. The probability that $\Omega$ contains a $receive\_msg(m)$ event is at least $1 - \epsilon$.

**uniqueness:** There are two complementing conditions:

- **no duplications:** let $\alpha$ be an execution of $D(A, ADV)$ that terminates in $send\_msg(m)$. Let $\Omega$ be a random variable assuming as values $OK$ extensions of $\alpha$ that does not contain $crash^R$ events. The probability that $\Omega$ contains more than one $receive\_msg(m)$ event is at most $\epsilon$.

- **no replay:** Let $\alpha$ be an execution of $D(A, ADV)$ that terminates in $receive\_msg(m)$ or $crash^R$. Let $M_\alpha$ be the set of messages $m$ such that $send\_msg(m)$ occurred in $\alpha$ and is followed by an $OK$ or $crash^T$ event in $\alpha$. Let $\Omega$ be a random variable assuming as values receive extensions of $\alpha$. With probability at least $1 - \epsilon$ extension $\Omega$ terminates in $receive\_msg(m)$ s.t. $m \notin M_\alpha$.

The protocol must also ensure **liveness**, which are sufficient conditions under which the protocol makes progress. Intuitively this means that, if packets for infinitely many $send\_pkt$ are delivered, then also infinitely many messages are be delivered.

**Liveness:** Let $\alpha$ be an execution of $D(A, ADV)$ such that after the last $send\_msg$ event in $\alpha$ there is no $crash^T$ or $OK$. If the adversary is a fair adversary then eventually one of the following actions occurs: $crash^I$, $crash^*$, $OK$ or $receive\_msg$.

Note that the liveness condition is not probabilistic. We are guaranteed that the system will always make progress. Also note, if there is a message waiting to be delivered, and the adversary is a fair adversary and does not generate any $crash^T$ or $crash^R$ actions, then with probability one there will occur an $OK$ action.

A pair of randomized algorithms $A = (A^*, A^I)$ is an $1 - \epsilon$ implementation of a data link if they satisfy the following two conditions.

1. If axioms 1 to 2 are satisfied then the causality, order and uniqueness conditions are satisfied with security parameter $\epsilon$.
2. If axioms 1 to 3 are satisfied then the liveness condition is satisfied.

Any protocol that is a $1$ implementation of a data link is a protocol that satisfies the conditions given in [LMF88] for a data link protocol.
3 The Protocol: Overview

First we describe the protocol when there are no faults. Then incrementally we show how this protocol can be modified to tolerate packets loss, host crashes, packets duplication, and delivery out of FIFO order.

The basic protocol is based on exchanging three packets. Assume that all the packets are delivered in order, without duplications or omissions. The receiver initiates the process by sending a random string $\rho_R^1$, of length $l_0$, to the transmitter. When the transmitter receives a random string $\rho$, it replies with a packet $(m_1, \rho, \tau_T^1)$ where $m_1$ is the message and $\tau_T^1$ is a random string of length $l_0$. The receiving station, when receiving a packet $(m^*, \rho^*, \tau^*)$, checks if the first random string $\rho^*$ is equal to $\rho_R^1$. If they are equal, it delivers the message $m^*$ to the higher layer (i.e. generates $\text{receive}.msg(m^*)$) and replies with a packet $(\tau^*, \rho_R^1)$, where $\rho_R^1$ is a new random string. When the transmitter receives a packet $(\tau^*, \rho^*)$, it checks that $\tau^* = \tau_T^1$. If so, then the transmitter generates $\text{OK}$ and waits for $\text{send}.msg(m_2)$, remembering $\rho^*$. When $\text{send}.msg(m_2)$ occurs the transmitting station sends $(m_2, \rho^*, \tau_T^1)$, where $\tau_T^1$ is a new random string, and so on.

The first modification is in order to overcome packets loss (i.e. some packets are sent, but never delivered). We still assume that packets are delivered in order and are not duplicated. We add an internal action $\text{RETRY}$ to the receiving module, and make the assumption that it occurs an infinite number of times. The modification in the protocol is that the receiver, in response to a $\text{RETRY}$ event, retransmits the last packet until it generates a $\text{receive}.msg(m^*)$. As before, when the packets are delivered in order, it is sufficient that both the transmitter and the receiver choose a single random string for a specific message. (Later, in order to make the protocol robust against packet duplication and delivery out of order, this would be modified.) Progress is been made due to the fact that each station continues to transmit the last packet it sent until it gets a reply. If the adversary is fair, then eventually one of the packets of that the station sent is delivered.

A second modification is required when the adversary can also duplicate old packets and delivery packets out of order. First we show why the previous solution fails. Suppose the following scenario: the system was running for a long time (even without any error) and the number of messages exchanged is much larger then $2^{l_0}$. At this time the adversary tries to make the receiver deliver an old message (i.e. violate the no replay condition). The adversary generates a $\text{crash}^T$ event followed by a $\text{crash}^R$ event. Then the adversary starts sending old packets $(m^*, \rho^*, \tau^*)$. There is no limit on the number of packets that the adversary can duplicate. Since much more than $2^{l_0}$ packets were sent, with high probability, the receiver's current random string was already used in the past, and is in one of the old packets. By repeatedly duplicating old packets the probability that one of the old packet will contain the receiver's current random string increases. Eventually, the receiver delivers an old message, violating the no replay condition.

One of the reasons of the above scenario is that the receiver used the same random string although it has received many incorrect packets. The main idea in the modification is that the random string is changed if too many incorrect packets are received. The value that is considered to be "too many" depends on the permitted probability of error, $\epsilon$, and the size of the current random string. If this bound is exceeded, the receiver chooses a new random string, and appends it to its old random string.

We now extend the protocol to tolerate host crashes. After a $\text{crash}^R(\text{crash}^T)$ event, the
receiving station (respectively transmitting station) generates a new random string $\rho^R$ (respectively $\tau^T$). Therefore, the probability that an old packet contains $\rho$ (respectively $\tau$) equal to $\rho^R$ (respectively $\tau^T$) is low.

We now explain how the liveness condition is satisfied. As long as there is no crash$^T$ or OK event, the length of $\tau^T$ is only increasing. Eventually, the length of $\tau^T$ is longer than any packet sent so far. Hence, old packets always have shorter $\tau$ than $\tau^T$. The transmitting station does not consider such packets as errors. Therefore the value of $\tau^T$ is fixed until a crash$^T$ or OK event occurs. Denote this value as $\tau_f$. A similar argument holds for $\rho^R$, if no crash$^R$ or receive_msg occurs. Intuitively, if the values of $\rho^R$ and $\tau^R$ are longer than any other value then the stations will ignore any old packet. Furthermore, eventually they will be able to complete the handshake, if the adversary is a fair adversary. To ensure that the transmitting station eventually sets $\tau^R$ to $\tau_f$, we add a counter $i^R$ of that counts the number of RETRY events since the last receive_msg or crash$^R$ event. The value of $i^R$ is added to the packets sent from the receiving station to the transmitting station.

More details can be found in the code of the protocol in Appendix A.

4 Analysis of the Protocol

In this section we show that the protocol presented is an $1 - \epsilon$ implementation of a data link, as defined in subsection 2.6. We prove the causality condition in subsection 4.1 and the order condition in subsection 4.3. As stated in subsection 2.6, the uniqueness condition holds if both the no duplications and the no replay conditions hold. Those conditions are presented as Theorems 8,7. The liveness condition is presented as Theorem 9.

Let $ADV$ be any adversary and $\alpha$ be any execution of $D(A, ADV)$. Probabilities are always taken over uniform coin tosses of the transmitting station, receiving station and $ADV$, unless stated otherwise. The probability of event $x$ is denoted $P(x)$. We use the convention that if $S$ is a sequence (e.g. an execution), then $S[i]$ is the $i$th element of $S$.

4.1 The causality condition

Theorem 1 The causality condition is satisfied.

Proof: Consider a receive_msg$(m)$ event. From the protocol of the receiving station, there was a receive_pkt$^{T-R}(m, \rho, \tau)$ before. The communication channel delivers only packets that were sent previously, hence a send_pkt$^{T-R}(m, \rho, \tau)$ action has occurred before. From the protocol of the transmitting station, before a receive_pkt$^{T-R}(m, \rho, \tau)$ happened send_msg$(m)$. The uniqueness of the send_msg$(m)$ event follows from the uniqueness of messages.

4.2 Preliminary Observations

The proofs of both the order and the uniqueness conditions are based on a common observation discussed in this subsection. This observation applies both to the transmitter and the receiver; we discuss and prove in Lemma 2 the observation in the receiving station. We omit the statement and the proof of Lemma 2T, which shows the dual observation for the transmitting station.
We need the following definition, which is based on the specification of the communication channel in subsection 2.3. Let $\sigma$ be a receivepkt$^{R-T}$ (receivepkt$^{T-R}$) event. This event immediately follows a deliverpkt$^{R-T}$ (corresponding deliverpkt$^{T-R}$) event. There is an unique preceding newpkt$^{R-T}$ (corresponding newpkt$^{T-R}$). Immediately before that event was a sendpkt$^{R-T}$ (corresponding sendpkt$^{T-R}$); denote it by $\delta$. Then we say that $\sigma$ was caused by $\delta$, or $\delta = cause(\sigma)$. If $\sigma$ was immediately followed by a OK (corresponding receivemsg) event denoted $\sigma^*$, we also say that $\sigma^*$ was caused by $\delta$.

Whenever the receiving station changes the value of $\rho$, it selects the new value randomly, independent on the past history. The claim intuitively means that for “almost every” $\rho$ the execution would be “indistinguishable” by the adversary. This implies that if the probability that the receiving station will accept a packet with the correct value of $\rho$, which has been sent before $\rho$ was chosen, is small.

**Lemma 2** Let $\alpha$ be an execution terminating in a receivepkt$^{T-R}$ event in which $t^R$ is incremented to $t$ (and num$^R$ is reset to 0). Let $\sigma_{t,n}$ denote the first receivepkt$^{T-R}$ event after $\alpha$, where $t^R = t$ and num$^R = n$. Let $\rho'_t$ be a random variable denoting the last size$(t, \epsilon)$ bits of $\rho^R$ generated at $\sigma_{t,0}$. Let $\rho'_{t,n}$ be a random variable denoting the last size$(t, \epsilon)$ bits of $\rho$ in the packet accepted at $\sigma_{t,n}$. For every fixed random tape of the transmitting station and $ADV$, there is a fixed sequence $R^*_t \in \{(0,1)^{\text{size}(t, \epsilon)} \cup \{L\}}^{\text{bound}(t)}$ s.t. in every extension of $\alpha$ in which $\rho'_t \notin R^*_t$, and for every $n$, $0 \leq n \leq \text{bound}(t)$, holds:

- If $R^*_t[n] \neq L$ then $R^*_t[n] = \rho'_{t,n}$.
- If $R^*_t[n] = L$ then $\sigma_{t,n}$ is caused by a sendpkt$^{T-R}$ caused by a receivepkt$^{R-T}$ event occurring after $\sigma_{t,0}$.

**Proof:** We iteratively construct $R^*_t$. For $n = 0$, $R^*_t[0] = \rho'_{t,0}$. For any fixed random tape of the adversary and the transmitting station, i.e. does not depend at all on $\rho'_t$. (This response is a receivemsg($m$) event unless prefix($r, r^R$)). If $R^*_t[n-1] \in \{0,1\}^{\text{size}(t, \epsilon)}$ then in every execution in which $\rho'_t \neq R^*_t[n-1]$, the receiving station rejects the packet accepted at $\sigma_{t,n-1}$.

In both cases, the packet delivered to the receiving station depends on $\rho'_t$ only if the packet was sent after a receive pkt$^{T-R}$ event caused after $\sigma_{t,0}$. In this case, let $R^*_t[n] = L$. If the packet delivered to the receiving station at $\sigma_{t,n-1}$ does not depend on $\rho'_t$, then $\rho$ in this packet has some fixed value $\rho_{t,n-1}$ (for a fixed execution and fixed random tapes of the transmitting station and the adversary). In this case, let $R^*_t[n]$ be the last size$(t, \epsilon)$ bits of $\rho_{t,n-1}$.

### 4.3 The order condition.

We now prove the order condition. The proof uses three lemmi which are presented later.

**Theorem 3** The order condition is satisfied. Namely, let $\Omega$ be a random variable assuming the value of OK extensions of $\alpha$. If $\alpha$ terminates in a sendmsg($m$) event, then $P(\text{receivemsg}(m) \notin \Omega) \leq \epsilon$.

**Proof:** Divide the execution $\alpha\Omega$ into three subsequences:
\( \alpha_{-1} \): all the events until the last receive\_msg in \( \alpha \). If there is no receive\_msg event in \( \alpha \), then \( \alpha_{-1} \) is empty.

\( \alpha_0 \): a random variable denoting the events from the last receive\_msg event in \( \alpha \) to the first receive\_msg event in \( \Omega \). If there is no receive\_msg event in \( \Omega \) (respectively \( \alpha \)), then \( \alpha_0 \) contains all of \( \Omega \) (respectively \( \alpha \)).

\( \alpha_1 \): a random variable denoting the events in \( \Omega \) which are not in \( \alpha_0 \).

Denote by \( \delta \) the send\_pkt\( R \rightarrow T \) event that caused the OK event. Denote by \( \tau_0 \) the value of \( \tau^T \) after \( \alpha \). Denote by \( \tau_f \) the value of \( \tau^T \) immediately before the OK event. By the protocol, the value of \( \tau^T \) in the beginning of an OK extension is a prefix of the value of \( \tau^T \) during the extension. Namely, \( \tau_0 \) is a prefix of \( \tau_f \). Denote by \( \tau_0^R \) the value of \( \tau^R \) at the receive\_msg event which preceded \( \delta \). From the protocol, the value of \( \tau_f \) is not an extension of \( \tau_{\text{crash}} \). Therefore the value of \( \tau_f \) is an extension of \( \tau_0^R \). But the length of \( \tau_0 \) is the shortest string used in the protocol. Therefore, \( \tau_0 \) is a prefix of \( \tau_0^R \) as well. Since \( \tau_0 \) is randomly chosen by the transmitting station, it follows that

\[
P(\delta \in \alpha_0) \leq P(\text{prefix}(\tau_0, \tau_0^R)) \leq \frac{1}{4}
\]

Let \( \xi \) be a random variable which is true when receive\_msg(\( m \)) \( \not\in \Omega \). Let \( n_{RM} \) be a random variable denoting the number of receive\_msg events in \( \Omega \). Let \( \gamma_1 \) and \( \gamma_2 \) denote the first and second receive\_msg event in \( \Omega \), respectively. By lemma 4 proven below, \( P(\xi \land n_{RM} \geq 2) \leq \frac{1}{4} \). Intuitively, this is since after the first receive\_msg event the receiving station chooses a new \( p_R \), and the probability that it will accept a packet from the past with the correct \( p \) is small.

By lemma 5 below, \( P(\xi \land \delta \in \alpha_1 \land n_{RM} < 2) \leq \frac{1}{4} \). Intuitively, this is since after a receive\_msg(\( m' \)) event, such that \( m' \neq m \), the value of \( \tau^R \) is probably not a prefix or an extension of \( \tau^T \).

By lemma 6 below, \( P(\xi \land \delta \in \alpha \land n_{RM} < 2) \leq \frac{1}{4} \). Intuitively, this is since after \( \alpha \) the transmitting station select a new \( \tau^T \), and the probability that it will accept a packet from the past with \( \tau = \tau^T \) is small. From all of the above follows that:

\[
P(\xi) = P(\xi \land n_{RM} \geq 2) + P(\xi \land n_{RM} < 2) \\
\leq \frac{1}{4} + P(\xi \land \delta \in \alpha_0 \land n_{RM} < 2) \\
\leq \frac{1}{4} + P(\xi \land \delta \in \alpha_1 \land n_{RM} < 2) \\
\leq \frac{1}{4} + P(\xi \land \delta \in \alpha_{-1} \land n_{RM} < 2) \\
\leq \frac{1}{4} + P(\xi \land \delta \in \alpha_{-1} \land n_{RM} < 2) \\
\leq \epsilon
\]

\[\square\]

We now present the three lemmi used in theorem 3. The first lemma shows that it is improbable that an OK extension of an execution terminating in a receive\_msg or crash\( R \) contains a receive\_pkt\( T \rightarrow R \) with the new \( p \) value, which was sent before \( p \) was selected. Intuitively, this shows that a certain case of message duplication is improbable. We use this lemma in other proofs as well.
Lemma 4 Let $\Omega$ be a random variable assuming the value of OK extensions of $\alpha$, that contain a receive\_msg or a crash event, denoted $\gamma_1$. Let $\xi$ be a random variable which is true if $\Omega$ contains a receive\_pkt\_T\_R event denoted $\gamma_2$, such that: (1) $\gamma_2$ follows $\gamma_1$, (2) $\text{cause}(\gamma_2) \in \alpha$, and (3) the value of $\rho$ accepted at $\gamma_2$ equals the value of $\rho^R$ at $\gamma_2$. Then $P(\xi) \leq \frac{\epsilon}{4}$.

Proof: Without loss of generality assume that there is no receive\_msg event between $\gamma_1$ and $\gamma_2$. Let $\sigma_{t,n}$ denote the first receive\_pkt\_T\_R event after $\gamma_1$, where $i_R = t$ and $\text{num}_R = n$. Let $\rho'_t$ be a random variable denoting the last size($t, e$) bits of $\rho^R$ at $\sigma_{t,0}$. Let $\rho'_{t,n}$ be a random variable denoting the last size($t, e$) bits of $\rho$ in the packet accepted at $\sigma_{t,n}$. A necessary condition for $\xi$ to occur is that a receive\_msg($m'$) follows some $\sigma_{t,n}$ whose cause is in $\alpha$. (Since if the cause of $\sigma_{t,n}$ is in $\Omega$, then $m' = m$.) The receiving station does a receive\_msg after $\sigma_{t,n}$ only if $\rho'_t = \rho'_{t,n}$. By definition, the cause of $\gamma_2$ is in $\alpha$. Therefore,

$$P(\xi) \leq \sum_{t=0}^{\infty} \sum_{n=1}^{\text{bound}(t)} P(\rho'_t = \rho'_{t,n} \land \text{cause}(\sigma_{t,n}) \in \alpha)$$ (1)

Until now, the probabilities were taken with respect to the coin tosses of both stations and of the adversary. It is sufficient to consider any fixed random tape for the adversary and the transmitting station, and also to fix the random tape of the receiving station until $\sigma_{t,0}$. Therefore $\rho'_{t,n}$ is a function of the send\_pkt\_R\_T and receive\_msg events after $\sigma_{t,0}$ and before $\sigma_{t,n}$. Lemma 2 shows that there is a fixed set $R^*$ such that $|R^*| \leq \text{bound}(t)$ and if $\rho'_t \notin R^*$ then for $0 \leq n \leq \text{bound}(t)$, either $\rho'_{t,n} \neq \rho'_t$ or the cause of $\sigma_{t,n}$ is in $\Omega$. The probability that a uniformly chosen $\rho'_t$ is in $R^*[n]$ is at most $\frac{1}{2^{\text{bound}(t)}}$. By substituting this value in 1,

$$P(\xi) \leq \sum_{t=0}^{\infty} \text{bound}(t) \cdot \frac{\epsilon}{2^{t+3}} \leq$$

$$\sum_{t=0}^{\infty} \frac{2^t \cdot \epsilon}{8 \cdot 2^t} < \frac{\epsilon}{4}$$

Lemma 5 Let $\Omega$ be a random variable denoting OK extensions of $\alpha$ which contains one receive\_msg event, denoted $\sigma$. Let $\delta$ denote the send\_pkt\_R\_T event that caused the OK event in $\Omega$. If $\alpha$ terminates in a send\_msg($m$) event, then $P(\sigma \neq \text{receive\_msg}(m) \land \delta$ occurs after $\sigma) < \frac{\epsilon}{4}$.

Proof: Let $\tau_0$ denote the value generated for $\tau^T$ at the beginning of $\Omega$. Let $\tau_0^R$ denote the value of $\tau^R$ at the end of $\alpha$. The probability distribution of $\tau_0$ is uniform, therefore the probability that $\tau_0 = \tau_0^R$ is bounded by $\frac{1}{2^{3 \text{bound}(t)}} \leq \frac{\epsilon}{8}$.

Until now, the probabilities were taken with respect to uniform coin tosses of both stations and of the adversary. It is sufficient to consider any fixed random tape for the adversary and the receiving station. If $\tau_0 \neq \tau_0^R$ then in all executions, either $\sigma = \text{receive\_msg}(m)$ or the value assigned to $\tau^R$ at $\sigma$ is some fixed value $\tau^*$. This is since what happens in $\sigma$ is determined by the responses of the receiving station to the receive\_pkt\_T\_R events before $\sigma$. Those responses are the same for all $\tau_0$ which are not a prefix of $\tau_0^R$.

It remains to show that the probability that $\tau^*$ has the same first $l_0$ bits as $\tau_0$ is less than $\frac{\epsilon}{8}$. But this is immediate since $\tau^*$ is a constant and $\tau_0$ is generated randomly. □
We now prove another lemma used by Theorem 3. This lemma deals with an OK extension of an execution which terminates in send.msg(m). In this lemma we show that it is improbable that an event prior to the extension is the cause of the OK event terminating the extension. This lemma is, in a sense, dual to lemma 4.

Lemma 6 Let $\Omega$ be a random variable denoting OK extensions of $\alpha$. Let $\delta$ denote the send_pkt$^{T \leftarrow T}$ event that caused the OK event in $\Omega$. The probability that $\delta \in \alpha$ is at most $\frac{3}{4}$.

Proof: Let $\sigma_{t,n}$ denote the first receive_pkt$^{R \leftarrow T}$ in $\Omega$, where $t^R = t$ and $num^R = n$. Let $\tau'_t$ be a random variable denoting the last size$(t, \epsilon)$ bits of $\tau^T$ at $\sigma_{t,0}$. Let $\tau^*_t,n$ be a random variable denoting the last size$(t, \epsilon)$ bits of $\tau$ in the packet accepted at $\sigma_{t,n}$. The transmitting station does an OK after $\sigma_{t,n}$ only if $\tau'_t = \tau^*_t,n$. By definition, the cause of the OK is in $\alpha$. Therefore,

$$P(\delta \in \alpha) \leq \sum_{t=0}^{\infty} \sum_{n=1}^{\text{bound}(t)} P(\tau'_t = \tau^*_t,n \land \text{cause}(\sigma_{t,n}) \in \alpha)$$

(2)

Until now, the probabilities were taken with respect to uniform coin tosses of both stations and of the adversary. It is sufficient to consider any fixed random tape for the adversary and the receiving station, and also to fix the random tape of the transmitting station until $\sigma_{t,0}$. Lemma 2$^T$ shows that there is a fixed set $T^*$ such that $|T^*| \leq \text{bound}(t)$ and if $\tau'_t \notin T^*$ then for $0 \leq n \leq \text{bound}(t)$, either $\tau^*_t,n \neq \tau'_t$ or the cause of $\sigma_{t,n}$ is in $\Omega$. The probability that a uniformly chosen $\tau'_t$ is in $T^*$ is at most $\frac{1}{2^{\text{bound}(t)} \cdot \text{bound}(t)} \leq 2^{-t^R}$. By substituting this value in 2,

$$P(\xi) \leq \sum_{t=0}^{\infty} \text{bound}(t) \cdot \frac{\epsilon}{2^{t+3}} \leq \sum_{t=0}^{\infty} \frac{2^t \cdot \epsilon}{8 \cdot 2^t} < \frac{\epsilon}{4}$$

(3)

\[\square\]

4.4 The uniqueness condition

Intuitively, the no-replay condition states that it is improbable that the receiving station will output old messages (replays).

Theorem 7 The no-replay condition is satisfied. Namely, Let $\alpha$ be an execution of $D(A,ADV)$ that terminates in receive_msg$(m)$ or crash$^R$. Let $M_\alpha$ be the set of messages $m$ such that send_msg$(m)$ occurred in $\alpha$ and is followed by an OK or crash$^T$ event in $\alpha$. Let $\Omega$ be a random variable assuming as values receive extensions of $\alpha$. Let $\xi$ be a random variable which is true when an extension chosen according to $\Omega$ terminates in receive_msg$(m)$ s.t. $m \in M_\alpha$. Then $P(\xi) < \epsilon$.

Proof Sketch: Let $\sigma_{t,n}$ be the first receive_pkt$^{T \leftarrow R}$ event in $\Omega$ where $t^R = t$ and $num^R = n$. The proof is essentially the same as that of lemma 4, which means we even prove a better bound of $\frac{3}{4}$. The only difference is in the argument why is it sufficient to bound the probability that $\rho'_t = \rho^*_t,n \wedge \text{the cause of } \sigma_{t,n} \text{ is in } \alpha$. The argument here is that if that is not the case, then the transmitting station must be transmitting a message $m'$ s.t. send_msg$(m')$ is not followed by an OK or crash$^T$ in $\alpha$, and therefore $m \notin M_\alpha$. \[\square\]
We now prove the no duplication condition, as defined in section 2.6. Intuitively, this condition states that it is improbable that a message sent is accepted more than once (before the OK occurs), unless the receiving station crashes (in which case it is impossible to avoid duplications).

**Theorem 8** The no duplication condition is satisfied. Namely, let \( \alpha \) be an execution of \( D(A, ADV) \) that terminates in send_msg(m). Let \( \Omega \) be a random variable assuming as values OK extensions of \( \alpha \) that does not contain crash\( R \) events. The probability that \( \Omega \) contains more than one receive_msg(m) event is at most \( \epsilon \).

**Proof:** Let \( \beta \) be an OK extension of \( \alpha \), chosen according to \( \Omega \), which contains two receive_msg(m) events, say \( \sigma_0 \) and \( \sigma_2 \). Denote the value of \( \tau \) assigned to \( \tau^R \) at \( \sigma_0 \) and \( \sigma_2 \) by \( \tau_0 \) and \( \tau_2 \) respectively. Both \( \sigma_0 \) and \( \sigma_2 \) were caused by a send_pkt\( ^R-T \) event in \( \beta \), therefore the value of \( \tau_2 \) is either an extension or a prefix of \( \tau_0 \). The receiving station does receive_msg(m) at \( \sigma_2 \) when \( \tau^R \) is neither an extension nor a prefix of \( \tau_2 \). The only way for this to hold is when after \( \sigma_0 \) there was a receive_pkt event in which the receiving station accepted a packet \((m', \rho', \tau')\) with \( \rho' = \rho^R \) (and \( \tau' \) which is neither a prefix nor an extension of \( \tau_2 \)). In such a case the receiving station generates a \( \sigma_1 = receive_msg(m') \) event. Therefore, the lemma follows from lemma 4. \( \Box \)

### 4.5 The liveness condition

Intuitively, the liveness condition states that if the adversary delivers packets sometimes, then eventually the protocol will manage to transfer a message. Note that this liveness holds for an almost unrestricted adversary, and a more tame adversary may not require all of the mechanism used in the protocol to ensure liveness in the strong sense of the following theorem.

**Theorem 9** The liveness condition is satisfied. Namely, Let \( ADV \) be a fair adversary. Let \( \alpha \) be an execution of \( D(A, ADV) \) that terminates in send_msg event. Then eventually one of the following actions occurs: crash\( T \), crash\( R \), OK or receive_msg.

**Proof Sketch:** We assume that after \( \alpha \), no crash\( T \) or crash\( R \) actions occur, otherwise the Theorem follows. We show that eventually either a receive_msg or an OK event occurs.

Let \( i_{max} \) be the maximum value of \( i^R \) in \( \alpha \), \( k_{max} \) the maximum length of \( \tau^T \) in \( \alpha \) and \( l_{max} \) the maximum length of \( \rho^R \) in \( \alpha \).

The value of \( i^R \) can be decremented only by a receive_msg or a crash\( R \) event occurs. If either events occur, the Theorem follows. Therefore, assume that the value of \( i^R \) is monotonic increasing. By the same argument, the length of \( \rho^R \) and \( \tau^T \) is monotonic increasing.

There are an infinite number of RETRY events. In each of them the value of \( i^R \) is incremented, therefore eventually \( i^R \) is greater than \( i_{max} \). Denote the first event where \( i^R \geq i_{max} \) by \( time_1 \). After \( time_1 \) any packet \((\rho, \tau, i)\) sent by the receiving station has \( i \geq i_{max} \). Since \( ADV \) is a fair adversary, one of those packets eventually reaches the transmitting station. Assuming this does not cause an OK event, then afterwards \( i^T \geq i_{max} \). Denote this event by \( time_2 \).

After the length of \( \rho^R \) exceed \( l_{max} \), the value of \( \rho^R \) is fixed. In order to change \( \rho^R \) one of the following events must happen: receive_msg, crash\( R \), or a receive_pkt\( ^T-R \) event in which \( |\rho| = |\rho^R| \) but \( \rho = \rho^R \). This can not happen since there is only one value that was generated that is of length greater than \( l_{max} \). By a similar argument we show that the length of \( \tau^R \) exceeds \( k_{max} \),
it becomes fixed. This implies that there is an event after which both $p^R$ and $T^T$ are fixed. Denote these values by $\rho_{\text{stable}}$ and $\tau_{\text{stable}}$. Denote by $\text{time}_3$ an event where the values of $\rho^R$, $\tau^T$ stabilized and $i^T \geq i_{\text{max}}$.

After $\text{time}_3$, the receiver sends infinitely many packets with $(\rho_{\text{stable}}, \tau_j, i_j)$. The transmitter replies each time $i_j > i^T$, and since the value of $i^R$ is unbounded, this will happen an infinite number of times. The transmitter will send infinitely many $(m, \rho_{\text{stable}}, \tau_{\text{stable}})$ packets, and will not send any other packet in between. Eventually the receiver receives one of the packets. If $\tau^R$ is not a prefix of $\tau_{\text{stable}}$ then it generate a receive_msg($m$) event, and the Theorem follows. Otherwise, it updates $\tau^R$ to $\tau_{\text{stable}}$, and starts sending packets $(\rho_{\text{stable}}, \tau_{\text{stable}}, i')$. Eventually, one of the packets reach the transmitter which generates an OK.

5 Conclusions and Open Problems

We introduce a model where an explicit probability of error is allowed. Using randomization, we designed a fault-tolerant and reliable source to destination protocol. We believe that the idea of an explicit error rate, with conjunction to a randomized protocol, may be very fruitful in many other applications.

The protocol presented here is simple and practical. The main open problem is to extend the protocol to a model in which allows the communication channel to deliver packets that were not sent. This means that the communication channel does not obey the causality axiom. In such a model, our protocol satisfies all the correctness conditions except liveness (given that the definition of the causality condition is relaxed to be probabilistic).

Other directions for further research are to weaken the assumption that adversary does not depend on the contents of packets, and to modify the protocol for better efficiency (e.g. to select good size, $\text{bound}$, $\text{increment}$ functions).

Acknowledgements

We wish to thank Yehuda Afek, Hagit Attiya, Baruch Awerbuch, Benny Chor, Shay Kutten and Adrian Segall for helpful discussions.

References


A The Protocol

This appendix contains formal description of the protocols of the receiver and the transmitter. We use the convention that static local variables of the receiver (transmitter) are denoted by the superscript \( R \) (\( T \) respectively). Namely, \( t^R \) is variable \( t \) in the receiver. The variables are described in Figure 4. The receiving station protocol is described in Figure 5 and the transmitting station is described in Figure 2.

Figure 3 describes some basic routines and constants used in the protocol. Below we describe some 'technical' details concerning the basic routines.

After a crash\(^R\) event, the receiving station has to select a new value for \( \tau^R \), the string used to check for new messages. But if we select a value that may be equal to or a prefix of \( t^T \), a message may be lost. This is not allowed by the safety properties. To prevent this, the receiving station chooses a special value to \( \tau^R \) upon crash, denoted \( \tau_{\text{crash}}^R \). The transmitting station always selects \( \tau^T \) so that \( \tau_{\text{crash}} \) is not a prefix of \( \tau^T \).

We use the function size for the length of extensions to the random string. This function depends on the maximal probability of error permitted \( \epsilon \) and on the number of times the string was extended so far \( t \). We use the function bound for the maximal number of wrong packets we accept before we extend the random string. The value of bound depends on the number of times the string was extended so far \( t \). The specific pair of bound and size given in Figure 3 is not the only selection that ensures correctness of the protocol.
$\text{size}(t, e) = t + 4 - \lfloor \log(e) \rfloor$

$\text{bound}(t) = \lfloor \frac{2t}{4} \rfloor$

$\text{random}(l)$ returns a string of $l$ random bits.

$\text{concat}(s, r)$ returns the concatenation of $s$ and $r$.

$\text{prefix}(s, r)$ is true iff $s$ is a prefix of $r$.

$\tau_{\text{crash}}$ returns some predefined string, e.g. 0.

$\tau'_{\text{crash}}$ returns a string different from $\tau_{\text{crash}}$, e.g. 1.

$\text{increment}(i)$ adds 1 to $i$.

Figure 3: Basic routines used in the protocol.

$\text{num}$ counts the number of packets with incorrect values received for the current random string.

$t$ is the number of times that the random string was extended during the delivery of the current message.

$p$ is the receiver's random string.

$r$ is the transmitter's random string.

$m_k$ hold the current message to be delivered.

$k$ denotes the number the message (used only for analysis).

$\text{busy}$ signals if the transmitting station is ready to accept a new message.

$i$ a counter of the times a packet is retransmitted.

Figure 4: The variables of the protocol.
\texttt{\texttt{crash}} \texttt{R} \quad \text{Effect: } k = 1; \ t_R = 1; \ \texttt{num}^R = 0; \ \tau_0^R = \tau_{\text{crash}}
\rho_k^R = \text{random}(\text{size}(t_R, \epsilon)); t_R = 1
\texttt{RETRY} \quad \text{Effect: } \text{send\_pkt}_{R-T}(\rho_0^R, \tau_{k-1}^R, i_R^R); \ \text{increment}(i_R^R)
\texttt{receive\_pkt}_{T-R}(m, \rho, \tau) \quad \text{Effect:}
\begin{align*}
\text{IF } \rho = \rho_k^R \text{ THEN} \\
\text{IF } \text{prefix}(\tau_{k-1}^R, \tau) \text{ THEN } \tau_{k-1}^R = \tau \\
\text{ELSE IF NOT } \text{prefix}(\tau, \tau_{k-1}^R) \text{ THEN receive\_msg}(m) \\
\quad \tau_k^R = \tau \ k = k + 1; \ t_R = 1; \ \text{num}^R = 0; \ i_R = 1 \\
\quad \rho_k^R = \text{random}(\text{size}(t_R, \epsilon)) \\
\text{ELSE IF } \text{length}(\rho) = \text{length}(\rho_k^R) \\
\quad \text{AND NOT } \text{prefix}(\rho, \rho_{k-1}^R) \text{ THEN} \\
\quad \text{num}^R = \text{num}^R + 1 \\
\text{IF } \text{num}^R \geq \text{bound}(t_R, \epsilon) \text{ THEN} \\
\quad t_R = t_R + 1; \ \text{num}^R = 0 \\
\quad \rho_k^R = \text{concat}(\rho_k^R, \text{random}(\text{size}(t_R, \epsilon)))
\end{align*}

Figure 5: The protocol in the receiving station.