Yet Yet on the Bounded Retransmission Protocol

Thérèse Hardin and Brahim Mammass

Laboratoire d'Informatique de Paris 6
Tour 45-55, 4 Place Jussieu 75252 Paris Cedex 05, France
e-mail: Brahim.Mammass@lip6.fr, Therese.Hardin@lip6.fr

Abstract. The aim of this paper is twofold. We first present a correctness proof of the Bounded Retransmission Protocol (BRP), which is easily done by bisimulation in the π-calculus. Then, we compare several works on this protocol, focusing on how the used formalism influences implementation choices and proof techniques.

1 Introduction

The development of communication networks needs more and more sophisticated communication protocols which must be reliable. Traditional verification methods use model checking techniques, but they cannot deal with infinite state systems and more generally with mobility.

Our aim is to elaborate some methodologic guides for designing and proving communication protocols using theorem provers. We choose the BRP as our case study because it is simple but, as it is parameterized, model checking cannot be directly applied. The BRP is a communication protocol, developed at Philips Research Laboratory, that transfers messages from a producer to a consumer over an unreliable physical medium that can lose messages.

The π-calculus [23, 24] is an extension of the process algebra CCS [21] with mobility keeping its algebraic properties. It is more expressive than CCS because it provides possibilities for coding data types, λ-calculus and higher order processes. Moreover, it offers the possibility of introducing new channels and passing channel names between processes. So, on one hand, we present a proof of the BRP using π-calculus bisimulations.

On the other hand, we study some related works in order to compare different approaches. They are the following. Helmink, Sellink and Vaandrager [14] analyze the BRP in the setting of I/O automata [17]. Havelund and Shankar [13] combine model checking and theorem proving techniques to prove the correctness of the BRP. They use PVS [26] as a theorem prover, and SMV [19], Murϕ [20], and an extension of PVS with the modal μ-calculus [15] as model checkers. Groote and Van De Pol [9] use as a formal support μCRL [10], a combination of process algebra and abstract data types, to prove the correctness of the BRP. Abrial [1] designs the BRP by successive refinements in the proof-assistant B [2]. Finally, D'Argenio, Katoen, Ruys and Tretmans [8] analyze the BRP in the setting of timed automata [3]. The correctness of the protocol is checked using UPPAAL [5].

The comparison between these approaches tackles the following questions. Do these works start from the same description ? How are the protocol entities modeled ? What is exactly proved ? What are the difficulties encountered in doing the proofs ? Are they due to the used formalism or to implementation choices ? Is the π-calculus a well-suited framework ?

The paper is structured as follows: section 2 presents the initial informal description of the BRP. In section 3, we complete this description by expliciting some points and we give the point of view of the mentioned papers. The section 4 gives an abstract view of the BRP in the π-calculus and compares it to the others. In section 5, we implement the protocol in the π-calculus. The section 6 presents our correctness proof which proceeds by bisimulation. Finally, in section 7, we present the other implementations and proofs of the BRP and we compare them to ours.

2 The Bounded Retransmission Protocol

Apart from some little and irrelevant variations, all the papers start from the same description which is the following. The BRP communicates messages from a producer to a consumer over an unreliable physical medium that can lose messages. It is a nontrivial extension of the alternating bit protocol [6] that uses timeouts and aborts transmission following a bounded number of transmission attempts. The environment of the protocol consists of the producer and the consumer. The black box view of the system is that it accepts requests $Req(f)$ from the producer to transmit the file f. When transmission of f has been either completed or aborted, the producer receives a confirmation $Conf(c)$, where c is either OK, NOTOK, or DTKW (don't know), respectively indicating that the file was successfully transmitted, aborted, or that the last message in the file was not acknowledged but might have been received by the consumer. The consumer either receives an *Inderr* signal indicating that the file transmission was aborted, or an $Ind(m, i)$ signal where m is the message and i is either FIRST, LAST or INC (incomplete) corresponding to the first, last, or an intermediate message in the file.

Opening the black box, the protocol (figure 1) consists of a sender program at the producer side, a receiver program at the consumer side, and two channels (one-place buffers): a message channel K and an acknowledgment channel L. Both channels are unreliable in that they can lose messages or acknowledgments; but, messages are neither garbled, nor received out of order. Two timers are used. A timer has a fixed period T of time associated. When it is set, a timeout occurs T time units later.

The sender sends each message over the channel K, sets $timer_1$, and then waits for an acknowledgment over the channel L. The $timer_1$ is used to detect the loss of a message or an acknowledgment. The time associated with this timer exceeds the time required to send a message over K and to get the acknowledgment over L. If an acknowledgment comes back within this time, the timer is cleared, and the next message is sent. If the transmission has been completed, the sender confirms *Conf*(OK) to signal a successful transmission to the producer. If there is no acknowledgment, a timeout occurs whereupon the message is retransmitted, and the timer set again. There is a fixed upper bound on the number of such retransmissions (MAX). When this retransmission bound has been reached, the sender aborts transmission and confirms that the transmission failed. Either it confirms *Conf*(NOTOK) or it confirms *Conf*(DTKW).

The receiver waits for messages over the channel K. If the alternating bit of the received message is equal to that of the previous one, the receiver only retransmits an acknowledgment over the channel L. Otherwise, it delivers the message to the consumer, stores its alternating bit, sets $timer_2$, and sends an acknowledgment over the channel L. In both cases, it waits for the subsequent message over the channel K. The time associated with $timer_2$ must exceed the required time to transmit MAX times a message (i.e, $timer_2 \succ \text{MAX} * timer_1$). If $timer_2$ expires, i.e., no new message is received, the receiver sends an *Inderr* signal to the consumer to indicate the transmission abort.

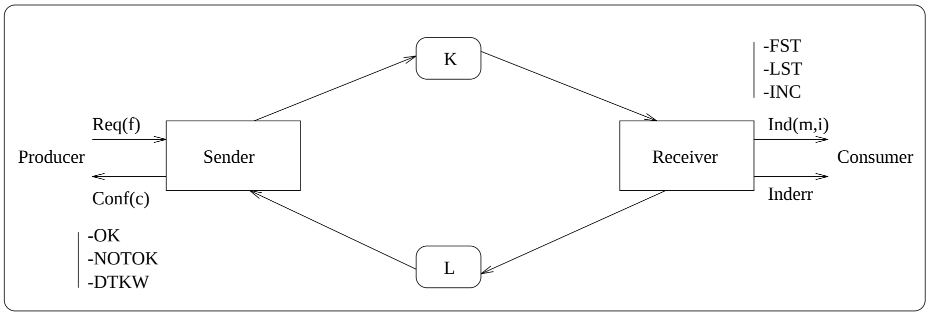

Fig. 1. The BRP protocol

We note that in [8, 9, 14], only one channel is used for indication signals from the receiver to the producer. So, if i equals NOTOK, the notification $Ind(m, i)$ indicates that an abort occurred and m is dummy. This difference seems irrelevant but perhaps may induce some typing difficulties if the protocol is extended by other treatments of m.

3 The complete informal specification

In this section, we give our interpretation of the BRP description, then we compare it to the ones adopted in the other papers.

3.1 Our interpretation

The previous section describes the protocol as a parallel composition of two entities: the sender and the receiver, that run sequentially. Before formalizing the protocol, we must complete the BRP description by expliciting some points. What can be the content of the file ? How can the synchronization between the sender and the receiver be realized ? How does the alternating bit work ? What happens if the first message never arrives at the receiver side ? In which cases must the receiver signal the abort to the consumer ?

We assume that the file may contain zero, one or more messages. If the file is empty, the sender must confirm immediately the transfer with a confirmation OK to the producer. If the file contains one message, this message must be considered as the last message in the file. Moreover, if the transfer is aborted during the transmission of this message, a confirmation DTKW must be sent to the producer.

The synchronization between the sender and the receiver after a transmission abort is achieved by means of $timer_2$. The description gives no indication about its activation. So, we assume that this timer is not enabled when the system starts. Otherwise, $timer_2$ could expire and should be restarted infinitely often leading to an active waiting of the receiver. After getting the first message, the receiver enables $timer_2$.

As $timer_2 \succ \text{MAX} * timer_1$, the sender is the first to detect a transmission abort. The sender may then receive a request to transfer a new file and send its first message while the receiver does not yet detect the abort (i.e., $timer_2$ has not expired). The receiver may consider this message as the next message (or as a duplication of the current message) of the previous file, that is wrong. So, the sender must wait until the receiver detects the abort. Moreover, the two must reinitialize their alternating bit before the beginning of the next transfer. For uniformity, we decide that they reinitialize their alternating bit also when the transfer has been completed.

After the correct termination of a file transmission, two situations are possible. Either the sender receives no new request before expiration of $timer_2$, so the receiver may send a misleading *Inderr* signal to the consumer. Or it receives a new request and sends the first message. If $timer_2$ has not yet expired, the receiver may consider this first message as a duplication of the last message of the previous file because it cannot know the new alternating bit value. To solve these synchronization problems, the sender must signal the end of the current transfer to the receiver before it begins the next one. The receiver can then anticipate the expiration of $timer_2$, then the two reinitialize their alternating bit.

If the first message in the file never arrives at the receiver side, as $timer_2$ is not yet set, the receiver cannot be informed about this abort. But, as decided above, the sender will wait the expiration of $timer_2$. This will lead to a deadlock situation. In a distributed setting, there is no solution to this problem because the abort information has to be transported on an unreliable channel. So, we assume that the first message arrives at least one time at the receiver side.

When an abort occurs during the transmission of an intermediate message, the sender must send a confirmation NOTOK to the producer and the receiver must send an *Inderr* signal to the consumer. If the abort occurs during the transmission of the last message, a confirmation DTKW is sent by the sender. But, at the receiver side, either this message was received, so no *Inderr* signal is sent. Or the message was always lost, an *Inderr* signal must be sent to the consumer. This point is not fully stated in the BRP description.

3.2 Other interpretations

Now, we only point out the differences between the other interpretations and ours. The consequences of their different choices will be discussed in section 7.

In [8, 9], the authors decide that when an abort occurs before the delivery of the first message, the consumer does not need an indication error. Moreover, if the transmission of the next file starts before $timer_2$ expires, the alternating bit scheme is simply continued. This scheme is only interrupted after a failure.

In [14], the acknowledgment consists of three control bits, but is considered as a simple signal in the other presentations.

In [1], the complete informal specification is written in a pseudo-code style. The author decide that $timer_2$ is only started when the received message is not the last one in the file.

4 The abstract view of the BRP

In this section, we consider the system as a black box. Its abstract view is the observable behavior on the external channels *Req*, *Ind*, *Inderr*, and *Conf*, abstracting the communications over the internal channels K and L. We first introduce the polyadic π-calculus [22] which is our formal framework. Then, we give our formal description and compare all the proposed abstract views.

4.1 Syntax and informal semantics of the polyadic π-calculus

Let $x, y, z, u, v, \ldots$ range over $\mathcal{N}$, a set of channel names. Let $A, B, \ldots$ range over a set of agent identifiers; each identifier has a nonnegative arity. We note by $\tilde{x}$ the tuple $x_1, x_2, \ldots, x_n$. Let $P, Q, \ldots$ range over agents (i.e. processes) which are defined as follows:

- 0, an agent which can do nothing.
- $\overline{y}\widetilde{x}.P$, an agent which outputs the tuple $\widetilde{x}$ on channel y; thereafter it behaves as P. In this action, y is the *subject*, $\widetilde{x}$ is the *object*, and both $\widetilde{x}$ and y are *free*.
- $y(\widetilde{x}).P$, an agent which receives a tuple on channel y; thereafter it behaves as P but with the newly received names in place of x_i. In this action, y is the *subject*, $\widetilde{x}$ is *bound*, and y is free.
- $\tau.P$, an agent which performs the silent action τ; thereafter it behaves as P.
- $P + Q$, an agent which behaves like either P or Q.

- $P \mid Q$, an agent representing the parallel composition of P and Q. This agent can do anything that P or Q can do, and moreover if $P = \overline{y}\widetilde{u}.P'$ and $Q = y(\widetilde{x}).Q'$, then $P \mid Q \xrightarrow{\tau} .(P' \mid Q'\{\widetilde{u}/\widetilde{x}\})$ where $Q'\{\widetilde{u}/\widetilde{x}\}$ is the substitution of each occurrence of x_i by u_i in Q'.
- $(\nu\ x)P$, an agent which behaves like P where the name x is local but P can export x.
- $[x = y]P$, an agent which behaves like P if x and y are the same name; otherwise it does nothing.
- $A(y_1, \ldots, y_n)$ is an agent if A is an identifier of arity n; for any such identifier there is a defining equation written $A(x_1, \ldots, x_n) \overset{def}{=} P$, where the names x_1, ..., x_n are distinct and are the only names which may occur free in P. The agent $A(y_1, \ldots, y_n)$ behaves like P where y_i is substituted for x_i for all $i = 1, \ldots, n$. Agent identifiers provide recursion since the defining equation of A may contain A itself.

The formal operational semantics of agents is defined and explained in [24]. In the sequel, we note $(\nu\ x_1 \ldots x_n)P$ instead of $(\nu\ x_1) \ldots (\nu\ x_n)P$.

4.2 The abstract view of the BRP in the π-calculus

The abstract view is pictured in figure 2 and is expressed by three recursive equations. The file is modeled by a list of messages and we use the usual functions *cons*, *hd* and *tl* on lists.

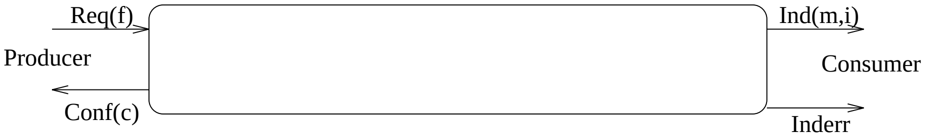

Fig. 2. The abstract view of the BRP

In the initial state S_0, the system waits for a file f on the channel *Req*. If f is empty it returns to S_0, else it processes the first message in the state S_1.

$$S_0 \overset{def}{=} \mathtt{Req}(f).([f = Nil].\overline{\mathtt{Conf}}\ \ \mathtt{OK}.S_0 \ + \ [f = cons(h,t)].S_1(f)$$

In S_1 and S_2, the τ actions indicate that the choice between the delivery or loss of a message or an acknowledgment is decided by the internal actions.

$$\begin{aligned} S_1(f) \overset{def}{=}\ & \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{LAST}.\overline{\mathtt{Conf}}\ \ \mathtt{OK}.S_0 \\ &+\ \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{FIRST}.S_2(tl(f)) \\ &+\ \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{LAST}.\overline{\mathtt{Conf}}\ \mathtt{DTKW}.S_0 \\ &+\ \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{FIRST}.\overline{\mathtt{Conf}}\ \mathtt{NOTOK}.\overline{\mathtt{Inderr}}.S_0 \end{aligned}$$