Task Analysis

This type of communication problem occurs at the so-called data-link layer of the OSI reference model. See Chapter 4, Sections 3 and 4, in *Computer Networks* by Tanenbaum, Second Edition, Prentice-Hall, 1988.

The programs have been written in the programming language Tangram developed at Philips Research for the design of VLSI circuits. Among other things, this task tests the competitor's ability to read and write an unfamiliar programming language given some informal explanation. The use of a 'standard' language, say Pascal, has the drawback that this will be perceived as giving an unfair advantage to those competitors familiar with that language.

The three questions are of increasing difficulty. Subtask A is relatively simple, Subtask B is already tricky, and Subtask C can be considered difficult (even though the required protocol changes are minimal). Subtask A tests the competitors' understanding of the task description and their ability to read Tangram. Subtask B requires a more careful analysis of the protocol and more imagination of what can go wrong (with respect to the given specification). Subtask C requires an intimate understanding of the communication problem and, to a smaller extent, tests the ability to write Tangram.

Subtask A

The retransmission protocol solves the problem of *losing* messages through damage by the *forward* link. However, the newly introduced backward link can also damage messages, namely the acknowledge bits. This is where the problem lies.

Table 1 shows how messages can get *duplicated*. The situation arises when $p_0 = 0$, $p_1 = 1$, and the 'good reception' bit for the first transmission gets damaged. This triggers a retransmission by *S*, whereas *R* expects a new message to arrive. So we can get $c_0 = c_1 = 0$.

T	*x*	*sf*	*fr*	*rb*	*bs*	*y*
0	0					
1		0				
2			0			
3						0
4				1		
5					*E*	
6		0				
7			0			
8						0
9				1		
10					1	
11	1					

Table 1: Transmission of $p_0 = 0$ and $p_1 = 1$ with damaged acknowledge

Subtask B

The improved protocol solves the problem of *duplicating* messages. Let us first illustrate how it deals with damage on the backward link. Table 2 shows the situation corresponding to that of Table 1 above. The columns labeled *t* and *u* indicate the tag in *S* and *R* respectively. A message on channels *sf* and *fr* consisting of value *v* and tag *t* is denoted by *v*,*t*. The improved protocol ignores the retransmission started in step 8, because the transmitted tag differs from the expected tag.

That the improved protocol does not work as desired is quite subtle. It will neither lose messages, nor duplicate them (these are *safety* properties). However, it may fail to deliver a message because of repeated errors on the forward and backward links (of course, these errors may not be such that from a certain moment onward all messages are damaged). Hence, it fails a *liveness* property. This can only be exhibited by an *infinite* scenario.

Table 3 shows how transmission can be blocked when transmitting $p_0 = 0$ and $p_1 = 1$. The pattern is that alternatingly the forward link succeeds but the backward link not, and then the forward fails but the backward link

T	*x*	*t*	*sf*	*fr*	*rb*	*bs*	*u*	*y*
0		0					0	
1	0	0					0	
2		0	0,0				0	
3		0		0,0			0	
4		0					0	0
5		0					1	
6		0			1		1	
7		0				*E*	1	
8		0	0,0				1	
9		0		0,0			1	
10		0			1		1	
11		0				1	1	
12		1					1	
13	1	1					1	
14		1	1,1				1	
15		1		1,1			1	
16		1					1	1
16		1					0	

Table 2: Transmission of $p_0 = 0$ and $p_1 = 1$ with damaged acknowledge

T	*x*	*t*	*sf*	*fr*	*rb*	*bs*	*u*	*y*
0		0					0	
1	0	0					0	
2		0	0,0				0	
3		0		0,0			0	
4		0					0	0
5		0					1	
6		0			1		1	
7		0				*E*	1	
8		0	0,0				1	
9		0		*E*			1	
10		0			0		1	
11		0				0	1	
12		0					1	
14		0	0,0				1	
15		0		0,0			1	

Table 3: Transmission of $p_0 = 0$ with alternating damages

not. So both links succeed half of the time. Time steps 6 through 15 can be repeated indefinitely. Thus, p_1 is never even retrieved from *P*; let alone delivered to *C*.

This problem is even better illustrated with a *state graph* that shows the states and state transitions of the combined system. *Not yet included.*

Subtask C

The correct protocol is also known as the *alternating-bit protocol*. It belongs to the family of sliding-window protocols, namely with window size equal to one. A recent formal treatment of these protocols is "The Sliding-Window Protocol Revisted" by Jan L. A. van de Snepscheut in *Formal Aspects of Computing* (1995) 7:3–17.

The improved protocol of Subtask B cycles through four phases. In the first phase, the message is "forced" through, by having *S* repeat it as often as is necessary. In the second phase, the acknowledge of good receiption is "forced" through, by having *R* repeat it as often as is necessary. The third and fourth phases are like the first and second phases but with opposite tags.

The first and third phases are all right. The second and fourth phases are just a little too weak. The idea behind the solution is to use a similar approach to the second and fourth phase as for the first and third phase. To that end the receiver alternates the meaning of the 0 and 1 acknowledge bits as well. Initially, 0 is a positive acknowledge and 1 negative. The sender repeats messages until an acknowledge with the expected bit *m:t* arrives; the receiver repeats acknowledges until a message with the expected bit *u* arrives.

The correctness can be seen from the state graphs (not yet included). The programs of the sender and receiver are shown in Figures 1 and 2 respectively.

```
S = proc (x?Msg & sf!TMsg & bs?Msg).
begin m: var TMsg & a: var Msg
| m.t := 0 ; x?m.v
; forever
  do sf!m ; br?a
   ; if a = m.t
     then m.t := 1-m.t ; x?m.v
     fi
  od
end
```

Figure 1: Program 3 for sender

```
R = proc (fr?TMsg & rb!Msg & y!Msg).
begin m: var TMsg & u: var Nat
| u := 0
; forever
  do fr?m
   ; if m = E then rb!(1-u)
     else if m.t = u
          then y!m.v ; u := 1-u
          fi
        ; rb!(1-u)
     fi
  od
end
```

Figure 2: Program 3 for receiver

Variations

The difficulty of the task can be varied somewhat by giving hints. For instance, in Subtask A we could specifically ask for a scenario in which messages are duplicated. In Subtask B, we could ask for an infinite scenario delivering only one message. In Subtask C, we could point at the idea of alternating the role of the acknowledge bits.

There are also some other (incorrect) versions of these protocols. For instance, a protocol that attempts to solve the problem of duplicating messages by indicating, with an additional bit, whether a transmission is a first transmission or a retransmission.

Only a limited subset of Tangram statements and expressions is introduced. This slightly complicates subtask C, but keeps the language simple. For instance, `forever do ... od` is the only repeating statement given; the notation for equality testing is given but not for inequality; no notation is given for boolean expressions (and, or, not). Expressing your solution in this limited language is part of the task.

Tom Verhoeff