

OCCAM: l'esempio concreto

di Luciano Macera

Come annunciato sullo scorso numero, questo mese commenteremo un po' la parte del software di rete di ADPnetwork scritta in OCCAM e quindi fatta per girare sul transputer delle schede di rete.

Per meglio comprendere, però, il funzionamento sarebbe opportuno che teniate sotto controllo anche l'articolo precedente a questo, in modo da non perdervi nei meandri dei troppi canali e processi di cui parleremo questo mese

Premessa

Inizialmente ADPnetwork, se ricordate, nasceva per girare su macchine Amiga senza hardware di rete aggiunto. La comunicazione tra macchina e macchina avveniva utilizzando semplicemente l'interfaccia seriale bidirezionale di cui ogni Amiga dispone. Non essendoci, allora, transputer di mezzo, tutto il software di rete era stato scritto in C, naturalmente utilizzando tutti (o quasi...) gli strumenti messi a disposizione da tale linguaggio. In più, per la comunicazione inter process, veniva massivamente utilizzato il tool ADPmttb (multitasking toolbox) scritto anch'esso interamente da ADP.

Con la progettazione e realizzazione, nella seconda fase, da parte del sottoscritto e di Giuseppe Cardinale Ciccotti (con tra i piedi, però, sempre ADP!) della scheda a transputer, parte del software di rete migrava sulla scheda e veniva tradotto ed adattato nel linguaggio OCCAM.

Volevamo inoltre mettere il meno possibile le mani nella parte di software ancora in esecuzione su Amiga, primo per risparmiare il più possibile tempo (lo SMAU 1990, sempre più prossimo, di certo non aspettava...), secondo per non modificare l'interfacciamento software con l'handler e il server di rete (scritti da Marco Ciuchini e Andrea Sautoni) che rimanevano così unici per la rete software e per quella hardware +software (figura 1).

Non disponendo, l'OCCAM, dei tipi di dato strutturati (come i record) la mag-

gior fatica dell'opera di traduzione è stata proprio la manipolazione dei pacchetti non più come record, ma come semplici array in cui andare a zappare con gli indici per prelevare o inserire i vari campi. Array che, una volta recapitati ai processi Amiga, venivano nuovamente visti (e utilizzati) da questi come record. Troverete dunque nel sorgente OCCAM listato in queste pagine spesso costanti il cui nome ha l'estensione «.pos» (punto pos): sono definite in un include a parte ed indicano, appunto, la posizione dei vari campi all'interno del pacchetto (visto da OCCAM come array) in arrivo, in transito o in partenza.

Ad esempio la riga scritta in «C»:

```
cks = frame->CkSum;
```

che copia nella variabile ad 8 bit «cks» il contenuto del campo CkSum, sempre ad 8 bit, della struttura puntata da «frame», in OCCAM diventa:

```
cks := frame[CkSum.pos]
```

in cui «cks» è sempre una variabile, «frame» un array di byte, «CkSum.pos» la posizione di tale campo, espressa in byte, nel record originario. La cosa naturalmente si complica lievemente quando dobbiamo prelevare valori diversi da byte. Ad esempio, per leggere un campo a 16 bit in «C» non cambia nulla:

```
mitt = frame->Mitt;
```

purché la variabile «mitt» sia sufficiente a contenere «frame->Mitt» non abbia-

mo nessun tipo di problema. In OCCAM non è possibile copiare direttamente in una variabile di tipo INT16 (intero a 16 bit) due posizioni contigue di un array di byte ma è necessario fare il trasferimento accedendo due volte all'array:

```
mitt := (frame[Mitt.pos]<<8) + frame[Mitt.pos + 1]
```

L'operazione «<<8» esegue semplicemente uno shift a sinistra di 8 bit per fare posto al secondo elemento che completa così la lettura.

Canali, processi, transputer

In figura 2 è mostrato il «pallogramma» dei processi OCCAM in esecuzione sulla scheda di rete (governata, come detto e stradetto, da un transputer T222) e i relativi canali di comunicazione tra gli stessi. I canali «prev.in» e «next.out» sono mappati su due link fisici del transputer e tramite driver differenziali vengono collegati il primo alla macchina precedente, il secondo alla macchina successiva. I canali «toamiga» e «fromamiga» sono anch'essi mappati su un link fisico e da questo su un Link Adaptor per l'interfacciamento diretto con il bus dell'Amiga.

Due processi qualsiasi in comunicazione tra loro, ad esempio SenderAmiga e AmigaInterface, possono ovviamente chiamare i loro canali d'ingresso e d'uscita con nomi diversi (lo stesso canale è chiamato «amigarequest» dal primo e «strobe» dal secondo). A rimettere le cose in ordine, ovvero a far sì che canali con nomi diversi in processi diversi identifichino lo stesso canale, ci penserà il processo master «NetPuter», presente in fondo al listato, che dichiara i canali necessari e li passa come parametri a tutti i processi che lancia in parallelo (se state guardando il listato, sono proprio le ultime 7 righe capeggiate da un bel «PAR»).

I parametri del processo NetPuter sono invece i link fisici (più l'indirizzo di

rete) specificati in un apposito file di configurazione (non listato) utilizzato durante la fase di generazione eseguibile. Ricordiamo, infatti, che né nel sergente, né in fase di compilazione o di linking è necessario fare alcuna assunzione riguardo il lancio di processi su singolo processore o su processori diversi. Una stessa applicazione multi processo, una volta compilata e linkata, a seconda delle specifiche indicate nel file di configurazione per la generazione dell'eseguibile, potrà ad esempio girare in multitasking su un solo transputer (mappando i processi sul medesimo chip e passando loro come parametri canali logici) oppure su più chip, in parallelismo reale, passando come parametri dei canali i link fisici dei transputer. In fase di debugging della rete, ad esempio, le prime prove sono state fatte su un solo transputer sul quale lanciavamo i processi relativi a quattro nodi distinti (in tutto $5 \times 4 = 20$ processi): dopo aver visto che il funzionamento corretto avveniva, senza ricompilare nulla abbiamo mappato i 4 software di rete su 4 transputer distinti avendo modo così di saggiare anche le effettive performance dei link fisici. Insomma, vi assicuriamo, è stato un bel divertimento!

Commentiamo il codice

Ma torniamo al listato e cominciamo dal primo processo «Dispatcher» che ha il compito di decidere se i pacchetti in arrivo sono per quel nodo (nel qual caso lo inoltra verso Amiga e genera il pacchetto di ACK corrispondente) o per altri nodi ovvero da ributtare sulla rete.

Le prime funzioni o procedure che troviamo nel listato sono utilizzate, come vedremo, dal vero e proprio processo «Dispatcher» il cui codice inizia un po' di righe più avanti e, quando necessario, dagli altri processi. Servono per calcolare il CheckSum sull'header del pacchetto, per risincronizzarsi su un pacchetto valido in caso di errore, per calcolare il codice CRC del corpo del pacchetto inviato o ricevuto. Quest'ultimo è un algoritmo di pubblico dominio come ampiamente dichiarato nella finestra di commento nel sorgente della funzione «crc32».

Subito dopo inizia il processo «Dispatcher» i cui parametri sono l'identificatore di rete per quel nodo e i canali (logici e fisici) passati dal processo master prima menzionato.

Dopo le dichiarazioni iniziali d'obbligo, troviamo una procedura locale che serve per reinoltrare i pacchetti in transito previa marchiatura degli stessi. L'opera-

zione, non necessaria ma fortemente consigliata, permette di ammazzare eventuali pacchetti «zombie» che potrebbero girare infinitamente sulla rete se il mittente di un messaggio ad un destinatario inesistente, dopo la spedizione esce dalla rete (ad esempio in seguito ad un guasto): in questo caso tutti gli altri nodi continuerebbero a reinoltrare all'infinito il pacchetto. Grazie alla marchiatura (effettuata per i soliti motivi di sicurezza da ben due nodi contigui diversi dal mittente) quando arriva un pacchetto da reinoltrare la procedura SendOut controlla se il campo «Stamp1» o «Stamp2» contiene il medesimo indirizzo di rete del proprio nodo: in caso affermativo vuol dire che il pacchetto è già passato di lì una volta (facendo un giro completo della rete) senza trovare il destinatario, ed il mittente, ugualmente, è assente (se no l'avrebbe assorbito lui). Nel caso invece che uno dei due campi «Stamp1» o «Stamp2» siano nulli, come detto prima, lo marca. Nel

terzo ed ultimo caso che i due «Stamp» siano già stati marchiati da altri processi (quindi non contengono l'indirizzo di rete di quel nodo e non sono nulli) il pacchetto può sicuramente essere rispedito verso la macchina successiva tramite i processi «ReceiverAmiga» e «SenderLink» che commenteremo più avanti.

Torniamo al processo «Dispatcher». Il funzionamento, tutto sommato, è piuttosto semplice: essendo i pacchetti di lunghezza variabile (multipla di 32 byte) la prima cosa che fa è leggere dal canale «prev.in» (tenete sempre sott'occhio la figura 2) un numero di byte pari alla costante MIN.LEN (pari appunto a 32). Nei primi 32 byte di un pacchetto troviamo infatti tutte le informazioni che ci interessano essendo lì contenuto l'intero header: tipo del pacchetto, lunghezza del pacchetto (espressa in multipli di 32 byte), mittente, destinatario, «Stamp1», «Stamp2», CheckSum, CRC, ecc. ecc.

Segue il controllo del CheckSum per

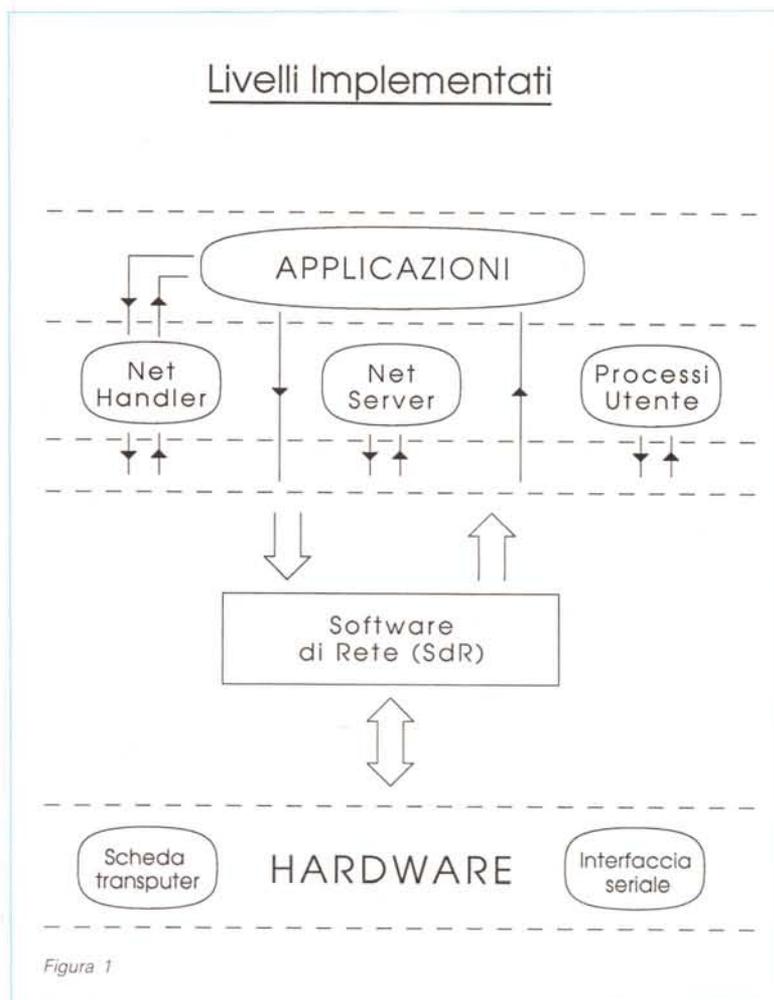


Figura 1

Processi OCCAM

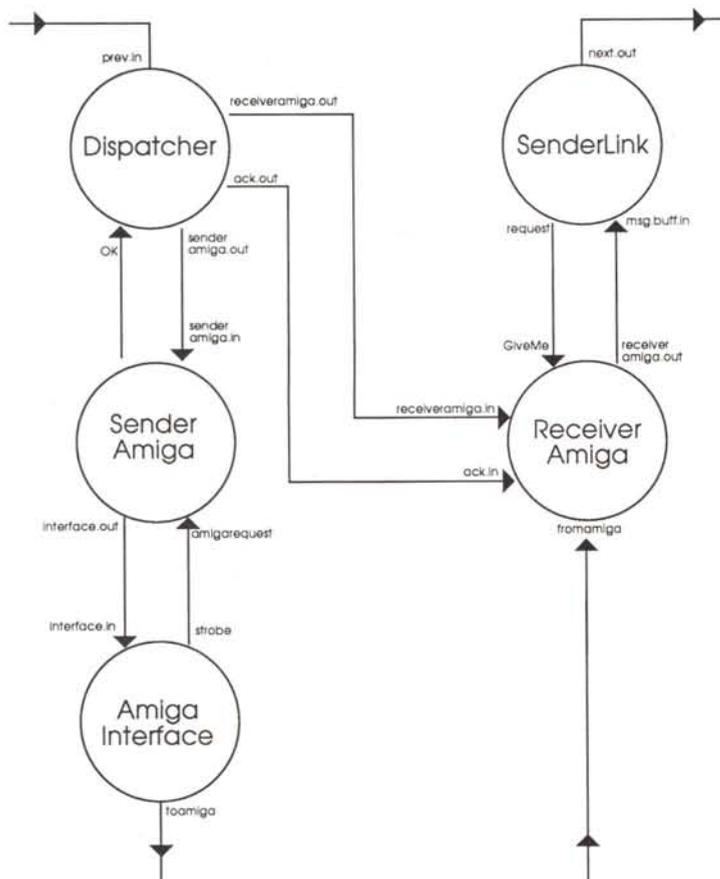


Figura 2

verificare la correttezza dell'header più altri controlli che permettono di diminuire le probabilità di prendere per buono un header scassato (limiti di alcuni campi, tipo di informazione in essi contenuta, ecc.) in cui casualmente la prova del CheckSum abbia dato esito positivo.

Dopo aver ricopiato in alcune variabili locali alcuni campi dell'header (con la tecnica illustrata ad inizio articolo) il processo «Dispatcher» si chiede se si tratta o meno di pacchetto di ACK (il che implica anche una lunghezza totale del pacchetto pari a MIN.LEN). In caso affermativo, se l'ACK è per un altro nodo viene invocata la procedura «SendOut» prima descritta, se è per il nodo in questione viene inoltrato verso il processo buffer «SenderAmiga».

Prima di continuare con l'analisi del pacchetto ed eventuali decisioni in merito (l'unica cosa assodata è che non si tratta di un pacchetto di ACK né per noi

né per altri nodi) è necessario leggere dal canale di ingresso «prev.in» l'eventuale rimanente porzione di pacchetto non ancora ricevuta.

Il successivo «IF len>1» provvede a leggere tutti i rimanenti byte del corpo del pacchetto in modo da poterlo gestire nella sua pienezza. Quindi calcolo dei byte validi all'interno del corpo e su questi controllo del CRC per testare la corretta ricezione del messaggio. Si aprono a questo punto tre possibilità: il pacchetto arrivato era stato spedito dallo stesso nodo in questione, arriva da un altro mittente ed è per un altro destinatario, il nodo in questione è effettivamente il destinatario del pacchetto. Nel primo caso se arriva un pacchetto spedito dallo stesso nodo, vuol dire che non esiste il destinatario dato che in una struttura circolare come quella di ADPnetwork un pacchetto che ritorna al mittente ha attraversato tutti i nodi col-

legati in rete e tutti l'hanno reinoltrato non riconoscendosi come destinatari dello stesso. Nel secondo caso il processo «Dispatcher», tramite la sua già citata funzione «SendOut» e relativi controlli di marchiatura, reinoltra su rete il pacchetto. Nel terzo caso il pacchetto è inviato verso l'Amiga.

E qui apriamo una breve parentesi. La scheda di rete da noi realizzata disponeva di 64k byte (il massimo indirizzabile dal T222) utilizzati per un buon 90-95% come buffer di ingresso e d'uscita per i pacchetti. Succedeva che, qualora il buffer d'ingresso su una scheda si riempiva tutti i pacchetti per quel nodo venivano uccisi fintantoché non si faceva posto nel buffer. Questo lusso era possibile in quanto il software di rete in esecuzione sugli Amiga era in grado di accorgersi che un pacchetto non era arrivato a destinazione e quindi provvedeva automaticamente a rispedirlo dopo un breve intervallo di tempo. Così il processo «Dispatcher» ogni volta che invia un pacchetto (reale o di ACK) al processo buffer «SenderAmiga» aspetta sul canale «OK» conferma da questo se effettivamente l'ha bufferizzato o meno. Nel caso infatti, come listato più avanti nel processo «Dispatcher» (al commento «- /* frame per me */»), di pacchetto per quel nodo, l'ACK corrispondente viene generato solo nel caso in cui il pacchetto è effettivamente stato accettato dal processo «SenderAmiga» («pushed» - TRUE). In caso negativo, non essendo generato l'ACK il processo mittente provvederà a sue spese, come detto prima, a rispedire il pacchetto «ucciso» dal destinatario per mancanza di spazio nel buffer. Il tutto, naturalmente, non certo per colpa dei velocissimi transputer ma semmai degli Amiga 500 e 2000 che con il loro 68000 a 7.1 MHz non possono certo rivaleggiare con i 20 Mbit/s dei link fisici e i 25 MHz di funzionamento dei T222 effettivamente in rete.

Spazio tiranno

Contavamo di riuscire a commentare l'intero sorgente OCCAM di ADPnetwork in una sola puntata e invece, giunti a questo punto e constatato il livello di dettaglio imposto per una facile comprensione anche ai non esperti, siamo costretti a continuare il commento sul prossimo numero. Il listato pubblicato, comunque, contiene anche il sorgente dei rimanenti processi (in pratica tutti quelli mostrati nel «pallogramma» di figura 2) che già potete cominciare a guardare e, come crediamo, a capire. Arrivederci...

MCS

Esempio di programma OCCAM (vedi testo).

```

#include "hostio.inc"
.....
/*..... 86/88/88 ..
.....
..... A D P n e t . w o r k 3.1
.....
..... Dispatcher
.....
..... (c) 1989 ADPsoftware
.....
.....
.....
.....
#include "ADPnet.inc"
.....
BYTE FUNCTION CalcCRC32(VAL [BYTE p,VAL INT l)
BYTE cks;
VALOF
SEQ
  cks := BYTE #
  SEQ l=# FOR l
    cks := BYTE ((INT cks) << (INT p(l)))
  \ RESULT cks
;

PROC FindFrame([BYTE p,CHAN OF BYTE prev.in)
  .. /* risincronizzazione su frame valido */
INT pf;
BOOL TF;
SEQ
  TF := FALSE
  pf := #
  WHILE NOT TF
    SEQ
      IF
        pf > ((FRAME.BUFFER-MIN.LEN)-1) -- /* reset finestra sul frame */
          SEQ
            [p FROM # FOR MIN.LEN] := [p FROM pf FOR MIN.LEN]
            pf := #
          TRUE
          SKIP
        pf := pf + 1 -- /* sposta finestra di un byte */
        prev.in ? p(MIN.LEN-1):pf] -- /* leggo un carattere dal link */
        -- /* verifica frame valido */
      IF
        (alpha(INT p(pf+type-pos))=1) AND
        (range(INT p(pf+len-pos))=1) AND
        (((INT p(pf+effLen-pos)) +
          (INT p(pf+msgid-pos))) *
          (INT p(pf+offset-pos))) = #)
        TF := ((p(CSUM-POS)) = CalcCRC32(p FROM pf FOR MIN.LEN,CKSUM-POS))
      TRUE
      SKIP
    [p FROM # FOR MIN.LEN] := [p FROM pf FOR MIN.LEN]
;

INT32 FUNCTION UPROC32(VAL INT32 octet,crc)
VALOF
SKIP
RESULT ((INT32 crc.32.tab[ INT ((crc << octet) ^ (INT32 #FF))] <> (INT32 (oct >> 8)))
;

INT32 FUNCTION CRC32(VAL [BYTE From,VAL INT Len)
INT32 oIdcrc32;
VALOF
SEQ
  oIdcrc32 := #FFFFFFF (INT32)
  .. /*.....
  .. * Copyright (C) 1986 Gary S. Brown. You may use this program, or
  .. * code or tables extracted from it, as desired without restriction
  .. *
  .. * OCCAM 2 Version by ADPsoftware 1988
  .. *.....
  SEQ i=# FOR Len
    oIdcrc32 := UPROC32(INT32 From(i), oIdcrc32)
  RESULT ((#FFFFFFF(INT32)) <> oIdcrc32)
;

PROC Dispatcher (VAL INT16 MyName, -- Numero del nodo,...
  CHAN OF BYTE prev.in,
  CHAN OF INT prev.out,
  CHAN OF INT :: [BYTE receiver,tag,out,sender,maila.out,ack.out,
  CHAN OF BOOL ok)
  MUSE "hostio.lib"
INT len;body;
BOOL pushed;
BYTE cks;
INT16 mitt,dest,stamp1,stamp2;
INT32 crc,crpacked;
[DEFAULT,LEN]BYTE frame;
PROC SendOut(CHAN OF INT::[BYTE out,][BYTE frame, VAL INT len)
IF
  (NOT (stamp1 = MyName)) AND (NOT (stamp2 = MyName))
  SEQ
    --/* questo frame passa per la prima volta da qui */
    IF
      stamp1 = (INT16 #)
      SEQ
        frame[Stamp1-pos ] := BYTE (MyName>>8) --/* lo marchio */
        frame[Stamp1-pos+1] := BYTE (MyName ^ (INT16 #FF))
        stamp2 = (INT16 #)
      SEQ
        frame[Stamp2-pos ] := BYTE (MyName>>8) --/* lo marchio */
        frame[Stamp2-pos+1] := BYTE (MyName ^ (INT16 #FF))
      TRUE
      SKIP
    frame[CKSUM-pos] := CalcCRC32((frame FROM # FOR MIN.LEN),CKSUM-pos)
    out ! len:=frame
    -- frame spedito
  TRUE
  SKIP -- /* frame già passato: kill!!!! */
;

SEQ

```

(continua a pag. 396)

```

(ok ? pushed
NOT pushed
SKIP
TRUE
IF
((dest = MyName) OR (frame[Type,pos] = NEXT.FRAME)) OR (frame[Type,pos] = POL.FRAME)
SEQ
-- /* frame per me */
senderanga.out ! len*MIN.LEN::[frame FROM # FOR len*MIN.LEN]
ok ? pushed
IF
pushed
SEQ
-- /* preparo l'ACK e la spedisco */
frame[Mitt,pos] := BYTE (dest>>8)
frame[Mitt,pos+1] := BYTE (dest /\ (INT16 #FF))
frame[Dest,pos] := BYTE (mitt>>8)
frame[Dest,pos+1] := BYTE (mitt /\ (INT16 #FF))
frame[Len,pos] := BYTE 1
frame[Type,pos] := ACK.FRAME
frame[Stamp,pos] := BYTE #
frame[Stamp,pos+1] := BYTE #
frame[Stamp2,pos] := BYTE #
frame[Stamp2,pos+1] := BYTE #
frame[CSus,pos] := Calcolares([frame FROM # FOR MIN.LEN],CSus,pos)
ack.out ! ACK.LEN::[frame FROM # FOR ACK.LEN]
NOT pushed -- /* non c'era spazio nel buffer... */
SKIP
TRUE
-- /* frame non per me, in transito... */
SendOut(receiveranga.out,[frame FROM # FOR len*MIN.LEN],len*MIN.LEN)
TRUE
SKIP -- /* bad CRC found
PROC Receiveranga (CHAN OF INT::[BYTE receiveranga.out,receiveranga.in,fronanga.ack.in,
CHAN OF INT GiveMe)
#USE "hostio.lib"
[BUACK.LEN][ACK.LEN][BYTE AckBuffer:
[BUFR.LEN][DEFAULT.LEN][BYTE FrameBuffer:
[BUFR.LEN][INT Len:
AckIn,AckOut,FrameIn,FrameOut,
AcAccount,FrameCount,dusy,body,
count:
crc:
swap:
INT32
BOOL
SEQ
swap := TRUE
AckCount,FrameCount := #,0,0
AckIn,AckOut,FrameIn,FrameOut := #,0,0,0
WHILE TRUE
PRI ALT
(IFrameCount > #) OR (AckCount > #) & GiveMe ? dusy
SEQ
-- /* Hostio.lib
[BUACK.LEN][ACK.LEN][BYTE AckBuffer:
[BUFR.LEN][DEFAULT.LEN][BYTE FrameBuffer:
[BUFR.LEN][INT Len:
AckIn,AckOut,FrameIn,FrameOut,
AcAccount,FrameCount,dusy,body,
count:
crc:
swap:
INT32
BOOL
SEQ
swap := TRUE
AckCount,FrameCount := #,0,0
AckIn,AckOut,FrameIn,FrameOut := #,0,0,0
WHILE TRUE
PRI ALT
(IFrameCount > #) OR (AckCount > #) & GiveMe ? dusy
SEQ

```

```

(sequie da pag. 395)
WHILE TRUE -- /* loop globale */
SEQ
SEQ !# FOR MIN.LEN
prev.in ? frame{1}
-- /* verifica correttezza header */
chk := Calcolares([frame FROM # FOR MIN.LEN],CSus,pos)
-- calcolato cks
IF
((NOT (cks = frame[CSus,pos])) OR
(alpha,INT frame[Type,pos])<#) OR
(range[INT frame[Len,pos]]<#) OR
(NOT (((INT frame[Len,pos]) <#)
(INT frame[Mitt,pos]) <#)
(INT frame[Dest,pos]) <#)
(INT frame[Offset,pos]) <#))
SEQ
-- bad frame found
FindFrame(frame,prev.in) -- /* risincronizzazione */
TRUE
SKIP
-- frame ok
len := INT frame[Len,pos]
dest := INT16 (((INT frame[Dest,pos])<#) + (INT frame[Dest,pos] + 1))
mitt := INT16 (((INT frame[Mitt,pos])<#) + (INT frame[Mitt,pos] + 1))
stamp := INT16 (((INT frame[Stamp,pos])<#) + (INT frame[Stamp,pos] + 1))
stamp2 := INT16 (((INT frame[Stamp2,pos])<#) + (INT frame[Stamp2,pos] + 1))
crcpacked := INT32 (((INT32 frame[CRC,pos])<#24) +
((INT32 frame[CRC,pos+1])<#16)) +
((INT32 frame[CRC,pos+2])<#8) +
(INT32 frame[CRC,pos+3]))
-- crc nel pacchetto = crcpacked
IF
frame[Type,pos] = ACK.FRAME -- /* e' un pacchetto di ACK ? */
IF
dest = MyName -- /* e' per me ? */
SEQ
senderanga.out ! ACK.LEN::[frame FROM # FOR ACK.LEN]
ok ? pushed -- /* non e' per me, Forward */
TRUE
SendOut(ack.out,[frame FROM # FOR ACK.LEN],ACK.LEN)
TRUE
-- /* non e' un pacchetto di ACK */
SEQ
IF
len > 1 -- /* il pacchetto e' piu' lungo di MIN.LEN */
SEQ !#MIN.LEN FOR MIN.LEN,(len-1)
prev.in ? frame{1}
TRUE
SKIP
body := ((len*MIN.LEN) - HEADER) - (INT frame[Chop,pos])
crc := crc32([frame FROM Body,pos FOR body,body)
IF
calcolato CRC
crc = crcpacked
IF
(mitt = MyName) AND (NOT (frame[Type,pos] = POL.FRAME))
SEQ
-- /* sono io il mittente ? destinatario inesistente ! */
senderanga.out ! len*MIN.LEN::[frame FROM # FOR len*MIN.LEN]
-- /* per avvertire HAS */
ok ? pushed
IF
pushed
SEQ
frame[Type,pos] := ACK.FRAME
senderanga.out ! ACK.LEN::[frame FROM # FOR ACK.LEN]
-- /* auto ack frame... */

```

