Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

ALGORITMI AFFIDABILI PER MULTICASTS SU TOKEN RING

M. Scevarolli - L. Simoncini

Nota Interna B4-58
Ottobre 1986

* INDICE *

INTRODUZIONE

Negli ultimi anni con la evoluzione dei sistemi di calcolatori, ed in particolare dei sistemi di comunicazione su cui sono basati, si è evidenziata la necessità di offrire primitive di comunicazione sempre più potenti, ma soprattutto affidabili.

Uno degli obiettivi attuali è definire come primitiva di comunicazione, più potente di quella standard punto a punto fino ad ora utilizzata, il "broadcast", cioè la diffusione del messaggio dal mittente a tutti gli altri nodi corretti della rete.

Il broadcast è però solo una prima tappa, l'obiettivo dei progetti più avanzati è infatti un suo ulteriore raffinamento: il multicast, cioè "una generalizzazione di varie comunicazioni tra varie locazioni, analogamente ad una conversazione attorno ad un tavolo" /ESPRIT/.

Il multicast meglio si adatta alle esigenze delle moderne reti, ed in particolare dei sistemi distribuiti, in quanto su di un modello di cooperazione globale si innestano, nella realtà, numerose cooperazioni ristrette, si pensi a gestori di particolari Data-Base che non saranno certo replicati su tutti i nodi, o a processi suddivisi in sottoprocessi paralleli (o concorrenti) su di un gruppo di processori.

Attualmente la comunicazione per diffusione, sebbene non in modo affidabile, è supportata dall'hardware in reti quali Ethernet e Token-Ring, ma è poco esplorata come tale e per niente in prospettiva multicast.

Si noti che un protocollo è considerato affidabile se possiede le seguenti proprietà:

terminazione: ogni messaggio diffuso da un mittente corretto arriva a tutti i destinatari corretti entro un qualche tempo limite;

atomicità: tutti i processori corretti devono o ricevere e dare il commit allo stesso messaggio o a nessuno;

ordinamento: l'ordine in cui vengono accettati i messaggi deve essere lo stesso per tutti i processori corretti.

Degli articoli studiati proponenti protocolli di broadcast affidabile solo il /BD.1/ evidenzia e sfrutta le potenzialità di reti a diffusione, ma in modo tale da non consentire dirette estensioni a multicast.

In questa nota interna è esposto il progetto di due protocolli che permettono la gestione affidabile di un numero qualsiasi di gruppi di multicast, con particolare attenzione alle caratteristiche della rete Token-Ring, scelta come supporto, ed alla efficienza, ottenuta parallelizzando quanto più possibile.

Si è scelto il Token-Ring per uniformarsi alle specifiche del progetto Esprit "P818 Delta-4", ispiratore di questo studio, che esaminate le diverse reti a diffusione esistenti lo considera la più interessante.

Nel definire i protocolli vengono modellate nuove categorie di guasti, quali la perdita di messaggi per buffer-overflow, che sono specifiche delle reti a diffusione e viene meglio caratterizzato il comportamento failstop che devono avere i nodi della rete, come richiesto dal progetto.

Il secondo algoritmo, estensione del primo, è tollerante guasti malicious (o bizantini) proprio per una ricerca di maggiore affidabilità; esso può essere semplificato ottenendo un algoritmo più potente di quello base, tollerante guasti omission, ma ugualmente semplice ed efficiente.

Entrambi sono caratterizzati da complessità limitate, in particolare da ridotti costi in messaggi e da una durata variabile ma contenuta.

Caratteristica essenziale è la loro compatibilità con il protocollo standard di comunicazione punto a punto; quanto proposto non si sovrappone ad esso ma vi si affianca, in modo da espandere le potenzialità del sistema senza modifiche dell'esistente.

Viene definita una suddivisione dei protocolli in tre livelli, simile a quella ISO/OSI /Tan/, che suggerisce una possibile implementazione modulare.

Tale proposta è importante sia perchè mostra più chiaramente, seppure in modo semplificato, la fattibilità dei protocolli ed il loro impatto sulla rete, sia perchè costituisce una base, assieme al modello ISO/OSI, per la definizione di un nuovo standard che comprenda tutti i tipi di comunicazione.

CAPITOLO 1

RASSEGNA CRITICA DELLA LETTERATURA

In questo capitolo tratto quegli aspetti legati alla progettazione di protocolli affidabili di multicast (o broadcast) in relazione con reti di tipo diffusione, ed in particolare per Token-Ring

Articoli interessanti dal punto di vista precedentemente definito sono [CM], [BD.1] e [BD.2]; il primo perche' pur non facendo assunzioni sulla sottorete di comunicazione appare evidentemente orientato verso il tipo broadcast, in quanto rende banale la implementazione del livello sottostante a quello del protocollo, che e' responsabile della diffusione fisica dei messaggi (*trasmission protocol*), gli altri perche' la scelta in tal senso e' esplicita e basilare.

Gli articoli [CASD] ed [SGS] sono invece studiati per reti punto a punto e cercano di sfruttarne le caratteristiche, si noti pero' che la loro formulazione lascia spazio ad una implementazione ragionevolmente simile (le modifiche non dovrebbero stravolgere la struttura degli algoritmi definiti), anche se si sostituisse la rete classica con una composta da molti piccoli cluster; una tale scelta puo' essere resa trasparente al livello dei protocolli.

1.1 VANTAGGI DELLA TOPOLOGIA BROADCAST.

Le reti di tipo broadcast permettono di limitare i comportamenti erronei dei processori guasti, almeno per gli aspetti relativi alla comunicazione.

Se infatti progettiamo una architettura di sottorete di comunicazione tale che (1) esiste un solo cammino tra un qualunque processore dato e tutti gli altri e (2) non e' mai necessario che i messaggi siano fatti *"avanzare"* da altri (tutti i processori sono in pratica *"vicini"* fra loro), allora otteniamo di non doverci piu' preoccupare della manomissione o mancata ritrasmissione del messaggio spedito.

In realta' sorgono altri problemi che, pur non annullando i vantaggi sopra esposti, li limitano.

Il punto debole sono le unita' di interfaccia sulla sottorete che devono rilevare la presenza di un messaggio e quindi raccoglierlo se e' a loro (o anche a loro) indirizzato.

In tale condizione non vi e' piu' la garanzia, ed e' difficile verificarlo, che un processore riceva tutti i messaggi a lui diretti, ad esempio possono succedere i seguenti inconvenienti:

- la memoria buffer e' piena quando un messaggio e' ricevuto dalla unita' di interfaccia (*buffer overflow*);
- la unita' di interfaccia "e' distratta" nel momento in cui transita il messaggio;
- in una rete a *"contention"* (tipicamente Ethernet), una collisione non rilevata che incide solo su alcune unita' di interfaccia causa loro la perdita di un messaggio.

Una conclusione che si puo' trarre da queste considerazioni e' che le reti di tipo broadcast permettono la diffusione di messaggi, ma non supportano direttamente broadcast tolleranti ai guasti.

Ritengo questi tipi di inconvenienti, e specificatamente i primi due, particolari casi di *"guasti ai collegamenti"* (della classe omission) per le reti broadcast, in quanto il comportamento esterno e' lo stesso.

Questo tipo di fallimenti e' stato attentamente considerato nella progettazione dei protocolli di multicasts su reti Token-Ring successivamente esposti.

1.2 LOGICA MULTICAST DEI GRUPPI.

Nella evoluzione dei sistemi distribuiti sta sempre piu' assumendo importanza la possibilita' di disporre di un supporto di comunicazione che implementi non solo comunicazioni punto a punto o broadcast, ma anche di tipo ***multicast***, cioe' tra sottoinsiemi qualunque (in genere di cardinalita' superiore a due ma piccola rispetto ad N) dell'insieme P dei processori.

Tra gli articoli analizzati solo il [CM] si preoccupa di questo aspetto, proponendo un protocollo che puo' certamente essere definito piu' di multicast che di broadcast, ed assumendo la proprieta di ordinamento in maniera locale ai singoli gruppi (Par. 1.1, Nota Interna B4-57 /ScSi/).

Il protocollo ha pero' due punti deboli:

(1)- non e' estendibile a tollerare guasti piu' gravi di quelli omission (assume nodi failstop) a causa del suo funzionamento ad *"acknowledge negativo"* ed al "potere", difficilmente controllabile, che di conseguenza viene assegnato al Token-Site.

(2)- valutandone l'implementazione su di una rete ad anello, come previsto per i protocolli da me proposti nel capitolo 2 la performance diviene meno interessante rispetto al modello teorico.

I principali problemi implementativi che ne limitano le buone caratteristiche sono cosi' sintetizzabili:

- Diviene piu' aleatorio il vincolo, su cui si basa il calcolo della durata del protocollo, sul tempo massimo T di permanenza del "Token" (non e' il Token-Free dell' anello) ad uno stesso Token-Site, in quanto non e' possibile comunicare quando si vuole ma bisogna attendere la disponibilita' del Token-Free;

- Il ritmo di trasferimento del "Token" incide pesantemente sulla durata del protocollo (per un messaggio termina solo quando ha ricevuto il commit) poiche' ogni acknowledge comporta l'attesa di un round (intero giro del Token-Free sull'anello). Anche assumendo un ritmo ideale pari ad un trasferimento per ogni *ack* problemi di perdite di messaggi (con relative richieste di ritrasmissione) e soprattuto di loro mancanza nella coda di commit a causa del presumibile eccessivo differenziale tra velocita' di acknowledge e ritmo di trasmissione, provocano un rallentamento consistente oltre ad un aumento dei costi.

- Diviene difficile gestire l'appartenenza di un nodo a piu' di un gruppo di multicast per il rischio, che diviene certezza quando il ritmo di trasferimento del "Token" e' pari ad uno per *ack*, di assumere il titolo di Token-Site contemporaneamente per piu' gruppi.

Questa somma di difficolta', oltre al punto (1), mi hanno orientato verso la semplice struttura di algoritmo ideata in [BD.1], ho pero' tenuto in grande considerazione i concetti di ordinamento, di "gruppo" e di "passaggio del Token" (nel senso di privilegio) esplicitati in [CM].

1.3 SINTESI DEGLI ASPETTI ESSENZIALI DELL'ARTICOLO [BD.1].

Si assumono le seguenti proprieta' del sistema:

Autenticazione - i messaggi possono essere firmati usando firme digitali;

Rete - in assenza di guasti il sistema garantisce tempi di trasmissione limitati superiormente per qualsiasi processore;

Clock - i processori non guasti mantengono un tempo locale che non differisce piu di un valore costante rispetto ad un tempo assoluto.

Tali assunzioni sembrano realistiche per un sistema omogeneo di processori vicini. Assumeremo che i clock siano perfettamente sincronizzati, in modo da pensare alla computazione come procedente in *lock-step round*.

Supponiamo che ne' un canale ne' un link guasto possano generare messaggi validi, inoltre che la autenticazione e la convalida sia effettuata da un livello inferiore a quello del **Reliable Broadcast**, e quindi i messaggi errati vengono eliminati e non visti.

Dopo un tempo limite l'assenza di messaggi viene considerata come messaggio nullo. In tali condizioni vale la:

Broadcast Network Failure Property.

In risposta ad un broadcast quei processori che ricevono un messaggio (non nullo) ricevono lo stesso. □

Cio' vale anche in presenza di malicious fault, in quanto e' impossibile fisicamente spedire piu' di un messaggio.

Broadcast Network Ridondanti.

Per garantire una "*1-connectivity*" tra i processori in presenza di realistiche ipotesi di guasto, e' utile replicare il canale di comunicazione per un numero R di volte. Come in precedenza ogni processore sara' collegato ad ogni canale.

□

DEF. *K-connectivity.*

Si dice che un sistema gode della proprieta' di *K-connectivity* se esistono K cammini distinti, fatti da links e canali, tra ogni coppia di processori.

[]

Nel seguito useremo i seguenti parametri:

- π . numero attuale di processori guasti;
- λ . numero attuale di links guasti;
- Γ . numero attuale di canali guasti.

Proprieta' 1

In una rete broadcast R-ridondante la 1-connectivity e' garantita se $R > \lambda + \Gamma$.

□

1.3.1 Protocollo affidabile tollerante guasti omission.

Indichiamo con # un valore di default conosciuto a priori da tutti i processori ed usiamo la notazione:

C_S : indica l'insieme di canali da cui il processore p_J ha ricevuto i messaggi m_J appartenenti a V durante il round r.

V : indica il dominio di tutti i possibili valori dei messaggi validi, escluso Ø.

Si noti che m_j e' unico a causa della restrizione agli omission faults.

PROTOCOLLO P1.

ROUND 1 : solo per p_1 (il mittente con messaggio v app. a V)

```
broadcast(v,*);          :diffusione completa
decide on v;             :commit
```

ROUND 2 : per tutti i p_j, $j \neq 1$

```
if |C_j| ≠ ∅ then
     :m_j e' l'unico messaggio ricevuto nel round 1
     foreach k_j in {1,2,..,R}-C_j  do
                              broadcast(m_j,k_j);
     od
     decide on m_j
     fi.
```

Alla conclusione del round 2, per tutti i p_j:

```
if not yet decided then
     if |C_j| ≠ ∅ then
          :m_j e' l'unico msg. ricevuto nel round 2
          decide on m_j
               else
          decide on #;
     fi
fi.
```

1.3.2 Protocollo affidabile tollerante guasti malicious.

L'avere introdotto la ridondanza ci ha portato un miglioramento nel grado di connessione del sistema, ma ha anche reso vulnerabile il sistema ai guasti di tipo malicious (infatti e' ora possibile spedire messaggi conflittuali).

Per ottenere un protocollo affidabile sono ancora necessari due soli round, ma rispetto a prima deve crescere il numero di processori coinvolti.

Notazione:

$W = \{\emptyset, \perp\} \cup V$;

S^K : insieme di tutte le K-uple ottenibili dall'insieme S;

$B_j^r(i)$ app. W^R : messaggi del processore i in ingresso sugli R canali durante il round r.

Se un processore p_j riceve piu' di un messaggio dallo stesso canale in uno stesso round dallo stesso mittente, pone

$$B_j^r(i) = \{\perp\}^R$$

Un insieme B e' consistente in v sse $B=\{\emptyset, v\}^R$ per qualche v appartenente a W, unico e diverso da $\perp$.

Funzione Filtro : Σ: $W^R \longrightarrow W$

$\Sigma(B) = v$ se B e' consistente in v,

\# altrimenti.

Funzione di decisione : Δ^z: $W^N \longrightarrow W$

$\Delta(M) = v$ se $\Theta(M) = v$ & $\Omega(M,v) \geq z$,

\# altrimenti.

ove $\Theta(M)$ seleziona il piu' frequente elemento diverso dal vuoto, e $\Omega(M,x)$ conta le occorrenze di x in M.

La funzione filtro serve ad ogni processo per decidere quale valore diffondere durante il secondo giro. La funzione di decisione permette di estrarre dalle N R-uple ricevute nel secondo round, il valore dell'intero broadcast, in dipendenza del parametro z che a sua volta dipendera' dalla quantita' massima di guasti previsti.

PROTOCOLLO P2

ROUND 1 : solo per p_s (mittente)

broadcast $(v_s, *)$;

ROUND 2 : per tutti i p_j

m_j <-- $\Sigma(B_j(1))$;

broadcast $(m_j, *)$;

Al termine del round 2, per tutti i p_j

M^j <-- { i=1,2,..,N : Σ (B_j (i)) };

decide on $\Omega^z(M^j)$;

TEOREMA 1.1

Se $N>t+\pi+2*\lambda$ allora P2 con z=t+1 implementa un broadcast affidabile per processori che possono fallire in modo malicious.

[]

CAPITOLO 2

PROTOCOLLI AFFIDABILI PER MULTICASTS SU TOKEN-RING

Nel campo delle reti locali utilizzanti sistemi di comunicazione del tipo *broadcast*, specialmente se di dimensioni medio-grandi, e' ormai considerato possibile ed auspicato disporre di primitive affidabili di comunicazione *asimmetrica*, cioe' fra gruppi di nodi e non fra coppie, che siano studiate e implementate sfruttando le particolari caratteristiche della sottorete di comunicazione (si veda il capitolo 1).

Lo studio che presento in questo capitolo modella due protocolli affidabili ed efficienti, l'uno tollerante omission faults, l'altro malicious, che permettono la gestione di *multicasts* anche paralleli ogni volta che le condizioni lo consentono.

La specifica sottorete di comunicazione su cui mi sono basato e' il Token-Ring, cosi' come da letteratura [BCKKM], e i protocolli proposti si affiancano a quello standard per le comunicazioni *simmetriche* che continuano ad essere possibili.

I protocolli sono affidabili in quanto garantiscono le proprieta' di ***atomicita'***, ***ordinamento*** (all'interno dei singoli gruppi) e ***terminazione***.

Per semplicita' ed aderenza alla realta' ho assunto che il Token-Ring non sia replicato, questo pero' non esclude la presenza di altre sottoreti, quali Ethernet, che fungano da ridondanza, come ipotizzato nelle specifiche del progetto ESPRIT "P818 Delta-4", che costituiscono base e motivazione di questo lavoro.

Il sistema e' composto da un numero N di nodi che ospitano

insiemi di processi, ognuno dei quali puo' comunicare singolarmente con gli altri, ma puo' anche appartenere ad uno o piu' *"gruppi di multicast"* individuati da specifici indirizzi.

Al fine di distinguere tali indirizzi multipli, nella interfaccia sull'anello assumo sia implementato un adeguato meccanismo di riconoscimento che permetta di prelevare i pacchetti di cui si e' destinatari, e che nel progetto ESPRIT e' indicato come MAR (*Multiple Address Recogniser*).

Si noti che al livello del protocollo non e' necessario ne' utile vedere i processi ma solamente i nodi che, una volta accettati i messaggi, li distribuiranno agli effettivi destinatari, percio' nel resto del capitolo considerero' solamente i nodi e non i processi al loro interno.

Dal punto di vista pratico un *"gruppo"* rappresenta un insieme di nodi che, o per una data applicazione o per necessita' di sistema, devono piu' o meno frequentemente scambiarsi e condividere informazioni.

In un sistema di medio-grandi dimensioni i *gruppi* coinvolgeranno generalmente solo piccoli sottoinsiemi, probabilmente distinti gli uni dagli altri; ad esempio in una fabbrica ove la rete connetta insiemi di piccoli elaboratori e terminali suddivisi nei reparti, e' facile che i gruppi siano composti da nodi dello stesso reparto, e che le comunicazioni fra nodi di reparti diversi siano simmetriche.

I protocolli permettono parallelismo tra comunicazioni relative a gruppi distinti, mentre inducono ordinamento e sincronizzazione fra quelle di gruppi *"collidenti"*, cioe' aventi almeno un elemento in comune.

2.1 ASSUNZIONI GENERALI SULLA STRUTTURA E I GUASTI.

La struttura assunta della sottorete di comunicazione e dei pacchetti e' quella standard definita da W.Bux et al. in [BCKKM], ove pero' utilizzo alcuni dei campi rimasti indefiniti per meglio gestire i multicast.

I campi che utilizzo sono due dei tre bits indefiniti presenti nell'*Access Control field* del *Frame Header*, che chiamo ***Phase Bits*** (PBs) e assumono il seguente significato:

PBs (bit 5 e 6): segnalano a quale delle quattro possibili fasi appartiene il pacchetto:

- -fase INIT di un multicast;
- -fase RIP-*etizione* di un multicast;
- -fase di PREN-*otazione*;
- -fase di *comunicazione simmetrica*.

Lo scopo di questi bits e' identificare i diversi tipi di pacchetti che possono circolare, in quanto non e' possibile farlo guardando la struttura globale che non cambia.

Per quanto concerne le interfaccie fisiche tra nodi ed anello definisco due assunzioni che ne vincolano il funzionamento indicando in quali casi devono fallire (*failstop*), al fine di mantenere consistente e corretto il sistema.

La scelta di indicare condizioni di *failstop* per le interfaccie deriva direttamente dalle specifiche del progetto ESPRIT, ove si auspica che l'intera sottorete di comunicazione garantisca un tale comportamento.

WEAK ASSUMPTION.

Un messaggio non puo' passare attraverso un nodo corretto senza che questi se ne accorga e ne legga almeno l'Access Control field (1 byte).

La mancata lettura dell'Access Control field o il ripetersi oltre un dato limite di errori in lettura non catastrofici (incapacita' di lettura di parte del Frame Header o del campo informazione quando si e' fra i destinatari) devono spingere il nodo al fallimento.

ð

STRONG ASSUMPTION.

Un messaggio non puo' passare attraverso un nodo corretto senza che questi se ne accorga e ne legga almeno il Frame Header.

L'errore in lettura del FH deve avere come conseguenza il fallimento del nodo, cosi' come il ripetersi oltre un dato limite di errori in lettura del campo informazioni di pacchetti di cui si e' destinatari.

ð

Nel protocollo tollerante *omission faults* utilizzo solamente la *weak assumption*, mentre in quello tollerante malicious faults devo usare anche la *strong*, ma solo per una parte dei messaggi.

Entrambi i protocolli garantiscono che tutti i processori corretti appartenenti ad un gruppo accettino gli stessi messaggi (*atomicita'*), che lo facciano nello stesso ordine (*ordinamento* limitato ai gruppi) ed in un tempo finito (*terminazione*).

I guasti considerati e tollerati sono π per quanto riguarda i processori ed in prima istanza nessuno per quanto riguarda

l'anello.

In realta' non e' cosi', il problema e' che avendo considerato un solo Token-Ring, la rottura dell'unico canale blocca necessariamente ogni comunicazione e va *"tollerata"* per mezzo delle altre sottoreti previste.

Esistono pero' vari possibili malfunzionamenti che non bloccano l'anello, ma che ne riducono l'efficenza e possono creare inaffidabilita' se non considerati; alcuni esempi sono:

- mancata raccolta di un messaggio per *buffer overflow*;
- lettura errata del messaggio riscontrata tramite *Frame Check Sequence*;
- qualsiasi anomalia temporanea che, pur permettendo di rispettare la assunzione debole, non consente la lettura di parte del pacchetto.

Per ogni singolo gruppo, allora, introduco due nuovi parametri di tolleranza che individuano questi malfunzionamenti.

μ - numero massimo di nodi di un gruppo che, durante ogni singola trasmissione, possono commettere errori in lettura (ma rispettano la *weak assumption*).

K - numero massimo di errori in lettura permessi ad ogni singolo nodo nel tempo intercorrente tra due *round*, ove con *"round"* indico l'intervallo temporale tra il passaggio di due Token-Free consequtivi rispetto al nodo stesso. Se un nodo registra K errori e' tenuto a sospendersi ed in tal caso rientra nella categoria π (*failstop*).

Al termine del protocollo solo i nodi veramente guasti possono non avere ricevuto ed accettato il messaggio, in partico-

lare tutti quelli che registrano meno di K errori in lettura nei due round sono corretti e percio' hanno le stesse informazioni degli altri (l'uso seppur limitato della strong assumption nel secondo protocollo permettera' di ampliare la tolleranza K, come mostrero' nel paragrafo 4.3).

Per garantire la sospensione in caso di K errori si utilizzano (nel primo protocollo) due variabili, Mla ed Mlp, col significato rispettivamente di "mancate letture nel round attuale" e "nel round precedente", la loro somma non deve mai superare K.

Al termine di ogni round si memorizza Mla in Mlp e si azzera Mla per utilizzarlo nel nuovo round attuale.

Il vincolo e' limitato a due round in quanto tale e' la durata di ogni multicast, cioe' l'intervallo durante il quale gli errori possono inficiare il corretto funzionamento dei protocolli attivi.

Ho definito questo nuovo formalismo per legarmi maggiormente alle reali condizioni di funzionamento di un Token-Ring, ove la perdita o il deterioramento di un pacchetto sono, per quanto improbabili, estremamente piu' probabili della rottura dell'anello e percio' non devono causare malfunzionamenti.

Calibrando opportunamente μ e K e' possibile adattare la tolleranza dei protocolli ad ogni esigenza.

Definisco ora una terminologia che utilizzero' nel resto del capitolo:

g_j : gruppo di multicast j-esimo;

N_j : numero di nodi appartenenti a g_j;

π_j, μ_j, K_j : parametri di tolleranza relativi al gruppo g_j.

2.2 PROTOCOLLO TOLLERANTE OMISSION FAULTS.

Rispetto alle specifiche del progetto ESPRIT la classe di guasti tollerati e' logicamente piu' ampia in quanto si ammette che un processore continui a funzionare, pur essendo guasto, a patto che esternamente il comportamento errato consista nel "non fare" cio' che dovrebbe, ma *non* nello spedire messaggi errati, in pratica pero' non vi sono sostanziali differenze.

Il protocollo, per motivi di funzionamento fisico della sottorete, di efficienza e correttezza, impone come vincolo che ogni processore partecipi ad al piu' un multicast per volta, sia nella veste di mittente che in quella di destinatario.

Solamente al termine del multicast a cui ha partecipato, o comunque quando vi e' la certezza che la fase essenziale e' superata, un processore puo' attivare o partecipare ad altri.

Per garantire cio' introduco due strutture dati che vengono utilizzate dai gestori del protocollo.

[a]- *Tabella dei Multicast Attivi* (TMCS).

In questa tabella vengono registrati gli indirizzi, o meglio le *"maschere di collisione"*, di tutti i multicast attivi che interessano i singoli nodi.

La "maschera di collisione" e' un codice associato ad ogni indirizzo che, confrontato con quello di altri, permette di dire se fra i due gruppi corrispondenti vi e' collisione o meno, cioe' se essi possono attivare multicast in parallelo.

La tabella viene utilizzata per decidere se e' possibile attivare un nuovo multicast, essendo condizione primaria la as-

senza di collisioni tra ciascuno dei multicast attivi.

Ogni messaggio circolante serve ad aggiungere un elemento alla tabella (sempre che non vi sia gia' compreso), che puo' essere suddivisa in due parti; una relativa ai multicast rilevati nel round precedente (TMCSp), e l'altra per quelli del round attuale (TMCSa).

Considerando il singolo nodo al momento dell'arrivo del Token-Free, cioe' al termine del round attuale, l'aggiornamento delle tabelle consistera' nell'assegnamento di TMCSa a TMCSp e quindi nell'azzeramento di TMCSa.

Ai fini del protocollo, pero', TMCSp ha solamente funzione di ulteriore controllo sui multicast a cui si e' gia' partecipato ma che non sono ancora terminati, e puo' essere eliminata senza intaccare l'affidabilita'; poiche' in tal modo si riduce la complessita, ho scelto di utilizzare solo TMCSa.

La gestione della tabella unica consiste semplicemente nel suo azzeramento al termine di ogni round.

[b]- *Tabella delle Prenotazioni* (TPREN).

Per evitare attesa infinita da parte di alcuni messaggi che vengano sistematicamente bloccati da continue collisioni, malgrado lo schema proposto preveda una sorta di passaggio circolare del *"privilegio"* che limita molto tale rischio, un nodo puo' *"prenotare"* un multicast, e la registrazione avviene in questa tabella.

Nel valutare se un multicast e' possibile bisognera' allora utilizzare sia la tabella TMCS, sia quella delle prenotazioni; quest'ultima mantiene le richieste fino a quando o sono state

esaudite, o e' scaduto il "tempo" (in round) entro cui sono certamente esaudibili e quindi se il nodo che ha fatto la prenotazione non ha attivato il multicas e' guasto o fallito.

Il calcolo di tale "tempo" come upper-bound e' sempre possibile osservando le prenotazioni gia' presenti in TPREN.

Si noti che la dimensione di entrambe le tabelle e' limitabile tenendo conto o del numero di nodi o del numero di gruppi, inoltre le informazioni contenute sono grandi pochi bytes.

I gestori del protocollo utilizzano anche una lista di richieste di multicast ed una lista di messaggi in attesa di commit.

La "maschera di collisione" e' comunque solamente una proposta, cio' che conta e' l'esistenza di un meccanismo che permetta di dire se vi e' collisione tra i gruppi identificati da diversi indirizzi.

2.2.1 Schema intuitivo del protocollo.

La semplice idea che sta alla base del protocollo e' che il messaggio venga ripetuto da un numero di elementi del gruppo sufficiente a garantire che chiunque e' corretto lo riceva sicuramente.

Le assunzioni per un gruppo g_J sono :

(1): $N_J - \pi_J - \mu_J \geq K_J$

(2): ogni nodo di g_J che ha ricevuto il messaggio iniziale (PBs settati al codice INIT) deve ritrametterlo, a meno di non averne gia' contato K_J copie.

Considerando solamente i nodi del gruppo g_J, quando uno di essi diffonde un messaggio iniziale dalla assunzione (1) si ricava che almeno altri $K_J - 1$ elementi del gruppo lo leggeranno correttamente.

Quando uno di essi riceve il Token-Free ripete il messaggio secondo quanto previsto da (2), in tal modo gli al piu' μ_J nodi che hanno perso la prima trasmissione, ed al limite le $K_J - 2$ successive, o si bloccano avendo perso anche la K_J-esima, o la raccolgono correttamente.

Questo schema assicura la atomicita', nel senso che tutti i nodi corretti ricevono lo stesso messaggio, e la terminazione, in quanto entro un round, relativamente al mittente, tutti hanno modo di ritrasmettere e quindi il protocollo finisce.

Il commit puo' essere dato nel round successivo alla ricezione del messaggio di INIT quando ritorna nuovamente il Token-Free, come prevede il protocollo, oppure direttamente nel round

precedente, quando il Token-Free arriva per la prima volta.

Ho scelto di attendere il round successivo per avere la garanzia che, al momento del commit, tutti i nodi del gruppo corretti posseggano gia' il messaggio.

I problemi sorgono quando il gruppo, visto fino ad ora isolato, lo si inserisce all'interno di un sistema piu' grosso e si permette che alcuni dei nodi (o anche tutti) di g_J facciano parte di altri gruppi.

E' a questo livello che divengono necessarie le strutture dati precedentemente presentate, TMCS e TPREN, che, permettendo ad ogni nodo di mantenere informazioni sui multicast che sono in corso nella rete (e che lo interessano direttamente) e su quelli prenotati, gli consentono di decidere quali azioni puo' intraprendere.

Valuto, allora, il comportamento globale del sistema supponendo che un nodo n desideri effettuare un multicast con indirizzo ind, e che questi sia ammissibile in quanto all'arrivo del Token-Free non collide con nessun altro in TMCS e TPREN.

Il nodo n spedisce il messaggio con il codice INIT nei PBs e registra nella propria tabella TMCS (appena azzerata) la maschera di collisione corrispondente a ind, ogni altro nodo del gruppo che raccoglie il messaggio aderisce al multicast in quanto, poiche' ind non collideva, e' certamente libero da altri multicast.

Gli al piu' μ nodi che possono perdere il messaggio iniziale, per la *weak assumption* se ne accorgono notando il codice INIT, e percio' rinunciano (nuovo vincolo) per un round ad atti-

vare nuovi multicast onde evitare il rischio di causare collisioni (non hanno infatti potuto aggiornare TMCS), possono pero' effettuare una normale comunicazione simmetrica o una di prenotazione.

Quando questi riceveranno la copia del messaggio, segnata in PBs come ripetizione (RIP), la registreranno semplicemente nella lista di attesa commit per il successivo round.

Il Token-Free, dopo che il mittente ha trasmesso il messaggio iniziale, passando di nodo in nodo viene raccolto da quelli corretti che avevano aderito al multicast e che quindi garantiscono le almeno K_d-1 ritrasmissioni necessarie al rispetto delle tolleranze assunte.

Un nodo corretto che ha aderito al multicast e che ha gia' contato K_d ritrasmissioni puo' astenersi dal ritrasmettere (condizione 2) il messaggio poiche' e' ormai inutile, e quindi sfruttare il Token-Free per una comunicazione simmetrica o per prenotare un multicast; perche' possa attivarne direttamente uno nuovo e' necessario che nella valutazione delle collisioni possa escludersi, in quanto in TMCS vi e' ancora ind a cui lui appartiene e che non e' ancora terminato.

Al termine del round il mittente e' a sua volta nelle condizioni appena descritte (K_d ritrasmissioni sono certamente gia' avvenute) ed e' quindi impossibilitato a rieffettuare un multicast uguale o collidente con ind che e' in TMCS, tale "privilegio" passera' ai nodi successivi, che hanno gia' azzerato in precedenza la TMCS che conteneva ind, con lo scorrere del Token-Free.

Questa e' la caratteristica che normalmente garantisce una

transizione circolare del privilegio di comunicazione all'interno di un gruppo j.

La prenotazione serve in casi piu' particolari e meno probabili, cioe' quando altri nodi bloccano alternativamente sottoinsiemi distinti del gruppo a cui si vorrebbe comunicare.

2.2.1.1 Messaggio di prenotazione.

Il messaggio di prenotazione e' estremamente compatto essendo sufficiente utilizzare il solo Frame Header ove i campi indirizzo contengono l'indirizzo di multicast prenotato e in PBs vi e' il codice per la fase prenotazione (PREN), il Frame Check coprira quindi solo il FH.

Data la importanza che riveste la ricezione di questo messaggio speciale e la sua ridotta dimensione propongo che venga leggermente modificata la procedura standard, permettendone la circolazione per un dato numero X di volte (bastera' 2).

Un nodo che non riesce a raccoglierlo interamente e correttamente almeno una volta, malgrado le X possibilita', deve allora fallire (*failstop*) per garantire la correttezza del sistema.

Se non si accetta neppure questa lieve modifica dello standard, bisogna richiedere che ogni nodo che perde il messaggio o non possa attivare nuovi multicast (salvo eventualmente quello precedentemente prenotato) fino a quando non ha riottenuto con comunicazioni ad hoc la informazione persa (questo puo' pero' costituire un rallentamento, generalmente inutile, dell'intero sistema), oppure si blocchi immediatamente, delegando alla fase di riattivazione il compito di riaggiornare le tabelle.

2.2.2 Proprieta' del protocollo.

Enuncio ora una serie di teoremi che dimostrano la affidabilita' del protocollo e ne indicano il costo.

TEOREMA 2.1

Non sono possibili in alcun caso attivazioni di multicast collidenti.

Dim..

Se un nodo desidera attivare un multicast deve confrontarne l'indirizzo con quelli nella tabella in suo possesso per valutare se collide.

Se la tabella non e' corretta a causa della perdita di un messaggio (parte indirizzi) di tipo INIT, il nodo e' tenuto a rimandare di un round ogni attivazione e si evita in tal caso la possibilita' di collisioni.

Poiche' i guasti considerati sono quelli omission non sono possibili messaggi anomali, e quindi non vi sono altri casi oltre a quelli visti sopra.

[]

TEOREMA 2.2

Quando e' trascorso un round, relativamente al mittente, dall'inizio del multicast, tutti i nodi corretti del gruppo hanno ricevuto il messaggio almeno una volta.

Dim..

La dimostrazione e' evidente per quanto gia' detto nel paragrafo precedente.

[]

TEOREMA 2.3

Entro un tempo finito e limitato ogni nodo j riesce ad attivare ogni multicast richiestogli dai livelli superiori.

Dim..

Anche se tutti gli X nodi che, nell'istante in cui arriva dai livelli superiori la richiesta di effettuare un multicast, precedono sull'anello il nodo j rispetto alla posizione del Token-Free attivano uno dopo l'altro un multicast che coinvolge j, entro X round egli ha a disposizione il Token-Free ed e' libero di spedire cio' che vuole.

A questo punto o attiva il multicast, se non vi e' collisione, o lo prenota ed in tal caso e' garantita la sua attivazione entro al piu' Y round successivi, ove Y-1 sono gli indirizzi gia' prenotati.

[]

In realta' basteranno meno di Y round in quanto solamente alcuni dei multicast gia' prenotati lo costringeranno ad attendere perche' collidenti, molti verranno attivati in parallelo guadagnando cosi' tempo.

TEOREMA 2.4

All'interno di ogni gruppo e' garantito l'ordinamento dei messaggi.

Dim..

E' banalmente verificato in quanto non sono consentiti multicast attivi sullo stesso gruppo se non in modo sequenziale.

[]

TEOREMA 2.5

Per ogni multicast di un gruppo j il numero di messaggi necessari varia da un minimo di K_j ad un massimo di $2*K_j-1$.

Dim..

Il minimo e' ovviamente K_j (compreso quello iniziale) per la assunzione (2), mentre il massimo e' $2*K_j-1$ in quanto, dopo un tale numero di ripetizioni, ogni nodo o ha contato almeno K_j copie, essendo $(2*K_j-1)$ meno le K_j-1 perdite ammesse pari a K_j, e percio' per (2) non lo ritrasmettera', oppure e' fallito avendo contato piu' di K_j mancate letture.

[]

2.2.3 Considerazioni sul protocollo.

Indico una serie di considerazioni su particolari caratteristiche che non sono evidenti ad una prima osservazione del protocollo.

- Non e' necessario che un nodo conosca tutti i possibili indirizzi di multicast, ma e' sufficiente che tratti quelli a cui puo' partecipare e quelli che collidono con almeno uno di questi.

Gli indirizzi di multicast relativi a gruppi di cui fa parte saranno registrati in MAR (si veda [ESPRIT]) e permettono di raccogliere i messaggi interessanti (campo informazione del pacchetto).

Quelli che collidono con i precedenti vengono utilizzati solamente per individuare le relative maschere da inserire in TMCS ed in TPREN.

Normalmente percio' la interfaccia sulla rete e' tenuta a leggere almeno il FH, ed in particolare i PBs ed il To-Address.

- Le ipotesi fatte si basano sulla assunzione debole, ma poiche' in generale varra' anche quella forte la sicurezza e l'efficienza sono notevoli ed alcune condizioni possono essere rilassate.

Ad esempio un nodo che pur perdendo un messaggio iniziale, o comunque trovandolo deteriorato, ne raccolga correttamente l'indirizzo, puo' aggiornare la tabella TMCS e quindi evitare di attendere un round prima di attivare nuovi multicast.

In ogni caso e' possibile ignorare tutti i messaggi con indirizzo sconosciuto poiche' non costituiscono pericolo di collisione.

- Il parametro K puo' variare da gruppo a gruppo ed un nodo che partecipi a piu' gruppi ne utilizzera' diversi; in tale caso, pero', quando perde un messaggio iniziale deve utilizzare il K_J minimo per decidere se fallire, a meno di non ricevere prima una copia del messaggio perso che gli permetta di identificare il K relativo.

E' probabilmente possibile individuare dei meccanismi che modificando dinamicamente K, μ e π si adeguino alle condizioni della rete.

- A livello di protocollo tollerante omission faults e' possibile rilassare il vincolo failstop su K, in quanto il singolo nodo e' in grado di decidere, raccogliendo una ripetizione (si veda punto precedente), se era relativa ad un messaggio INIT perso che lo riguardava.

Quando cio' succede puo' azzerare sia **Mlp** che **Mla** onde evitare di fallire inutilmente per altre mancate letture che non inficiano piu' la affidabilita' del protocollo.

- Le uniche operazioni costose in termini temporali ed operativi sono quelle di aggiornamento della tabella e di confronto per valutare la presenza di collisioni.

Si noti che, sebbene nello schema formale presentato successivamente esse appaiano eseguite al momento dell'arrivo del Token-Free, in realta' possono e sono valutate in precedenza.

In particolare ogni volta che un messaggio viene raccolto vengono eseguite tutte le operazioni necessarie, aggiornate le tabelle ed eventualmente modificato il comando da eseguire all'arrivo del Token-Free.

Questo garantisce una risposta immediata come e' necessario per non rallentare il sistema di comunicazione.

- Si noti, infine, che tutte le valutazioni sono fatte considerando condizioni pessime, cioe' attribuendo le K-1 mancate letture tollerate allo stesso gruppo di messaggi (iniziale piu' sue copie), mentre non sara' generalmente cosi'.

2.2.4 Recovery di un nodo.

Quando un nodo si riattiva avra', presumibilmente, le tabelle vuote, comunque inconsistenti, ed i problemi da affrontare sono cosi' sintetizzabili:

(1)-riorganizzare le tabelle in modo da potere operare correttamente;

(2)-ottenere da altri nodi eventuali messaggi persi durante il periodo di fallimento, o comunque il nucleo essenziale di informazioni per la cooperazione del sistema.

problema (1).

La tabella TMCS non costituisce un vero problema in quanto e' sufficiente attendere un round e riprendere a riempirla normalmente perche' divenga corretta e consistente.

La tabella delle prenotazioni deve invece essere in qualche modo riempita, ad esempio richiedendola ad uno degli altri nodi che condivide gli stessi indirizzi, questa comunicazione puo' essere inglobata in quella o in quelle che necessitano per (2).

Una alternativa alla richiesta dell'intera tabella, che comunque e' di piccole dimensioni, consiste nel richiedere solamente quanti round bisogna attendere perche' tutte le prenotazioni siano esaudite o scadano, e basarsi su di esso come termine minimo per la riattivazione vera e propria; fino ad allora si puo' leggere, fare comunicazioni simmetriche,partecipare a multicast di altri ma non attivarne di propri.

problema (2).

Quali siano e se esistano dei dati da rilevare dagli altri nodi per il corretto funzionamento del sistema distribuito dipende dalle specifiche applicazioni gestite e si utilizzera' una normale routine di recovery.

Da (1) e (2) si ricava che un nodo e' ufficialmente fallito fino a quando non e' riuscito a raccogliere tutti i dati di cui necessita per un corretto funzionamento.

2.2.5 Perdita del Token-Free.

La perdita del Token-Free e' il piu' grave inconveniente considerato nelle specifiche della sottorete ad anello, se esso avviene non e' pero' necessario alcun particolare meccanismo al di la' di quelli standard gia' esistenti e definiti in [BCKKM].

Una volta ristabilito il Token-Free, supponendo che tutti i nodi corretti si siano accorti di quanto e' accaduto (ognuno ha il proprio monitor), per evitare che la variazione del round crei problemi di consistenza delle tabelle bisogna che nessun nuovo multicast sia attivato durante il primo round, in modo da consentire la terminazione di quelli rimasti sospesi, ed una sorta di reinizializzazione dei gestori.

Durante il primo round sono quindi ammesse solo ripetizioni, prenotazioni e comunicazioni simmetriche.

2.2.6 Schema formale del protocollo.

Nel dare una definizione ad alto livello dal protocollo utilizzo un formalismo ECSP-like; nello schema ho individuato i vari tipi di messaggi che i livelli inferiori passano al gestore del multicast.

nodo i-esimo.

```
[ receive (INIT,msg,mitt,IND)                          ;i app. ad IND
    ----->insert ( [msg,mitt], lista-commit);
          MCS.flag<---IND;      ;indica il mcs. a cui si partecipa
          cont<---1;            ;conta le copie ricevute.
          ADD (TMCS, IND);
          if [IND,Mitt] IN TPREN then <cancellalo>
                                ;se il multicast era prenota-
                                ;to lo cancello da TPREN.


[]receive (RIP,msg,mitt,IND)          ;i app. ad IND,mitt e' quello
                                      ;originario, e non chi ha ri-
                                      ;petuto.
    ----->if MCS.flag=IND then cont<---cont+1;
                  else insert ([msg,mitt],lista-commit);
                                      ;la insert controllera' se
                                      ;[msg,mitt] e' gia' presente.
          if [IND,mitt] IN TPREN then <cancellalo>;


[]receive (PREN,mitt,IND)             ;i puo' anche non appartenere
                                      ;ad IND ma lo conosce.
    ----->insertpren ([mitt,IND], TPREN);
                              ;e' una insert speciale che calco-
                              ;lera' il tempo di scadenza e lo
                              ;assocera' alla coppia.


[]receive (erroreLettura, type)
    ----->if type=INIT then wait<---true      ;i ha perso un msg
                              ;INIT, wait indica che bisogna at-
                              ;tendere un round per la attivazio-
                              ;ne di multicast.
                        else if type=PREN then STOP;
          Mla<---Mla+1;
          if Mla+Mlp=K then STOP
                              ;aggiorno i cont. di mancata lettu-
                              ;ra, ed assumo K generale e unico.
```

```
[]receive (INIT,mitt,IND)              ;i non app. ad IND ma lo
                                       ;conosce.
    ----->ADD (TMCS,IND);
          if [mitt,IND] IN TPREN then <cancellalo>

[]receive (RIP,mitt,IND)               ;i non app. ad IND ma lo
                                       ;conosce.
    ----->if [mitt,IND] IN TPREN then <cancellalo>
                                       ;i ha perso il relativo msg
                                       ;INIT.
                    ;si noti che non lo registro in TMCS  perche'
                    ;o vi e' gia' o, se ha perso l'INIT relativo,
                    ;ho posto wait a true e non vi sono problemi.

[]receive (token-free)
    ----->&lt;dare il commit ai msg. marcati della lista commit&gt;;
          if MCSflag≠0 & cont<K        ;condizione (2).
                    then begin
                         msg<---read(lista-commit,MCSflag);
                         send(RIP,msg,i,MCSflag);
                         MCSflag<---0;cont<---0;
                         end
                    else <scegli cosa fare valutando i dati a
                         disposizione e le varie condizioni>;
          <marca i msg in lista commit>;
          M1p<---M1a;M1a<---0;
          <azzera TMCS>;
          <cancella da TPREN le prenotazioni scadute>;
].
```

figura .1
schema formale del protocollo tollerante omission faults

2.2.7 Proposta di implementazione a livelli.

Aderendo alle richieste del progetto ESPRIT propongo una suddivisione a livelli, similare a quella ISO/OSI, del protocollo di multicast.

I livelli considerati sono gli ultimi tre ove la politica multicast vera e propria e' compito del terzo; i livelli superiori, non definiti, dovrebbero preoccuparsi di una gestione globale delle potenzialita' di comunicazione (simmetrica, broadcast e multicast).

[0]- PHYSICAL LAYER (hardware).

A questo livello deve essere implementato il meccanismo che consente di individuare i messaggi che devono essere raccolti, cioe' il Multiple Address Recogniser (MAR).

Indipendentemente dalla accettazione o meno del messaggio trasmette il *Frame Header* o almeno l'*Access Control field* al livello [1] che li valutera'.

Se vi e' un fallimento completo in lettura genera un segnale di interruzione del nodo (che puo' ad esempio provocare l'attivazione di una routine di autocheck); questo e' un caso di scollegamento grave dalla rete.

Ogni volta che un Token-Free transita esegue il comando che attualmente e' depositato nell'apposito buffer, inoltre lo segnala al livello superiore.

[1]- DATA LINK LAYER (hardware o chip dedicato).

Riceve dal livello [0] i messaggi interi o i *Frame Header* o gli *Access Control field* o la segnalazione del passaggio del Token-Free.

Effettua i controlli di correttezza sui messaggi ed eventualmente li scarta (uso della *Frame Check Sequence*, ecc..).

Aggiorna i contatori di mancata lettura e se i limiti ammessi vengono superati genera un segnale di interruzione del nodo.

Registra nella tabella TMCS gli indirizzi di messaggi con codice INIT (o le relative maschere), e nella tabella di prenotazione TPREN quelli con codice PREN, scartando pero' quelli sconosciuti.

Trasmette al livello [2] i messaggi raccolti, la segnalazione del passaggio del Token-Free, quella di aggiornamento tabelle e i soli *Frame Header* dei messaggi di tipo INIT relativi a gruppi conosciuti ma a cui non si appartiene (di cui quindi non interessa il messaggio).

[2]- MULTICAST LAYER (o *Network Layer*, chip dedicato).

Riceve dai livelli superiori i messaggi di tipo multicast da spedire, e vi trasmette i messaggi ricevuti dalla rete e committed.

Gestisce un buffer per i messaggi in attesa di commit.

Ogni volta che il livello inferiore lo avverte di un aggiornamento della tabella rivaluta i messaggi in attesa di essere spediti ed eventualmente modifica il comando per il *Physical Layer*.

Le funzioni svolte sono quelle a piu' alto livello di coordinamento e di aggiornamento delle tabelle (in particolare TPREN).

A questo livello puo' essere implementata anche una politica di scelta tra comunicazioni multicast e simmetriche.

2.3 PROTOCOLLO TOLLERANTE MALICIOUS FAULTS.

L'obiettivo di questo protocollo, estensione di quello presentato nel capitolo 2.2, e' tollerare, nelle medesime condizioni generali di funzionamento del sistema in precedenza assunte, non piu' solo guasti omission, ma anche malicious: un nodo guasto puo' fare qualunque cosa gli sia fisicamente possibile.

Utilizzo nuovamente i parametri μ e K ad indicare la tolleranza ammessa relativamente alla perdita di messaggi, anche se, come mostrero' in seguito, ne viene parzialmente ristretta la portata.

L'idea fondamentale che mi ha spinto a ricercare una estensione per guasti malicious del precedente protocollo e' che anche in un sistema con pretese failstop totali, come quello ipotizzato nel progetto ESPRIT, non sempre (o mai) sara' possibile garantire in modo soddisfacente che nell'intervallo critico tra l'inizio di comportamenti errati ed il loro rilevamento non vengano spediti uno o piu' messaggi.

Se tali sono le condizioni un maggiore grado di tolleranza garantito da una implementazione semplificata di questo protocollo puo' rappresentare una interessante soluzione per un migliore confinamento degli errori.

Ho parlato di soluzione semplificata in quanto tutte le valutazioni ed i vincoli definiti in seguito sono forzatamente basate sui possibili comportamenti anomali estremi (al limite della volonta' negativa) e quindi non realistici.

Rispetto al protocollo tollerante omission faults non cambia ne' la filosofia di fondo ne' la parte sostanziale delle strutture dati utilizzate, e' pero' stato necessario appesantire alcuni vincoli per aumentare il livello di informazione in possesso di ogni singolo nodo, e questo non tanto per lo scambio di messaggi in se quanto per la parte riguardante la sincronizzazione e l'ordinamento delle varie comunicazioni.

2.3.1 Assunzioni e modifiche delle strutture dati.

Elenco ora, suddividendole in punti, le assunzioni fatte.

- Non e' piu' ammesso che un nodo perda la parte indirizzi del *Frame Header* per quanto riguarda i messaggi di tipo INIT; cio' equivale a richiedere che, limitatamente a questa classe di messaggi, valga la *Strong Assumption*.

La conseguenza e' che ogni nodo che non riesce a rispettarla deve autosospendersi (*failstop*), se non lo fa e' comunque guasto e rientra nella categoria π.

si noti che il vincolo riguarda solamente il *Frame Header* e non il campo informazioni, la cui perdita viene normalmente conteggiata nel contatore Ml (mancate letture) e tollerata entro il limite definito da K.

Lo scopo di questo vincolo e' garantire ad ogni nodo informazioni sufficienti a decidere quali fra le azioni ammesse sono corrette.

- Esiste un meccanismo di autenticazione che impedisce ad un nodo di spedire un messaggio con il "nome" di un altro senza che chi lo riceve se ne accorga.

Al posto del meccanismo di autenticazione, se si ritiene impossibile una azione volontaria in tal senso, si puo' dare una suddivisione dei nomi che renda estremamente improbabile che un errore di scrittura (o lettura) trasformi un nome in quello di un altro.

Lo scopo di questa assunzione e' evitare che nodi corretti siano indotti ad aderire a multicast che, in quanto collidenti con altri precedenti non ancora terminati, sono scorretti.

Le sole tabelle, infatti, non consentono in taluni casi che mostrero' meglio in seguito di decidere, prima che il Token-Free ripassi nuovamente, se il precedente multicast e' finito ed il nuovo e' corretto.

- Esiste una funzione che chiamo $CorrettoMittente_j$, propria di ogni nodo j, che prende in ingresso due indirizzi (o nomi) I1 ed I2 e risponde true se I2 e' compreso, rispetto al senso di rotazione del Token, fra I1 e j, false altrimenti.

Tale funzione e' di per se' molto semplice, per una banale implementazione e' infatti sufficiente ordinare sequenzialmente gli indirizzi, rispettando la collocazione fisica dei nodi ed il senso di rotazione del Token. Soluzioni piu' complesse si avranno in dipendenza dell'assegnamento degli indirizzi ai nodi.

- Ogni multicast collidente con quelli nelle tabelle TMCS e TPREN viene rifiutato da tutti i nodi corretti, e quindi neppure

registrato in TMCS.

- Esiste un meccanismo che, con un dato grado di tolleranza, permette di non accettare messaggi di nodi non appartenenti al gruppo individuato dall'indirizzo di multicast (falsificazione dell'indirizzo).

Per questa funzione assumo una tolleranza di errore, durante il corso di un protocollo, pari a β.

Lo scopo della funzione e' delimitare il rischio (comunque basso se non trascurabile) che nodi estranei intervengano nel protocollo a favore di messaggi falsi; nella realta' tale funzione e' probabilmente inutile e sostituibile dalla seguente assunzione:

> "durante i due round necessari alla terminazione di un multicast per il gruppo g_J al piu' β_J nodi esterni al gruppo possono intervenire spedendo messaggi"

che ha implicazioni sui vincoli del protocollo stesso.

Per quanto riguarda le strutture dati utilizzate i cambiamenti rispetto al caso omission sono i seguenti:

.. La tabella delle prenotazioni e' identica; il gestore che la aggiorna mantiene contatori associati alle varie entrate per sapere quando scadono e quando divengono attive.

Una prenotazione e' attiva quando non ne ha piu' altre collidenti davanti ed e' passato il numero minimo di round (uno) dal momento del suo arrivo, in modo da garantire che i multicast attivi siano tutti terminati.

.. La tabella TMCS e' suddivisa nelle due parti TMCSa e TMCSp per mantenere anche gli indirizzi dei multicast del round precedente, che possono non essere ancora terminati.

TMCSa e TPREN vengono utilizzate per decidere se si puo' attivare o meno un nuovo multicast in lista di attesa, mentre l'intera TMCS serve a stabilire se un multicast attivato da un altro nodo e' ammissibile, secondo le seguenti regole:

(a)- Un multicast collidente in TMCSa non e' mai ammissibile e va quindi ignorato;

(b)- Un multicast [IND2,MITT2] collidente con uno precedente [IND1,MITT1] registrato in TMCSp e' ammissibile per un nodo j se e solo se $CorrettoMittente_j$(MITT1,MITT2) e' true;

(c)- Un multicast che non collide con nessuna delle due parti di TMCS ne' con TPREN e' ammissibile;

(d)- Un multicast collidente con uno prenotato da un altro mittente non e' ammissibile.

Esaminiamo meglio il punto (b), che ha lo scopo di evitare la attivazione di [IND2,MITT2] prima che [IND1,MITT1], con cui collide, sia terminato, cioe' prima che il mittente abbia ricevuto il Token-Free successivo a quello di spedizione del messaggio iniziale.

Se un nodo guasto tenta di attivare il nuovo multicast prima di tale momento vi sara' un gruppo di nodi, compreso tra MITT2 e MITT1, che hanno ancora la coppia [IND1,MITT1] in TMCSa in quanto

devono ancora ricevere il Token-Free, e quindi rifiuteranno il multicast per (a).

Tutti gli altri, compresi tra MITT1 e MITT2, hanno la coppia in TMCSp e rifiuteranno il multicast per (b); nella figura .2 e' schematizzata la situazione appena descritta.

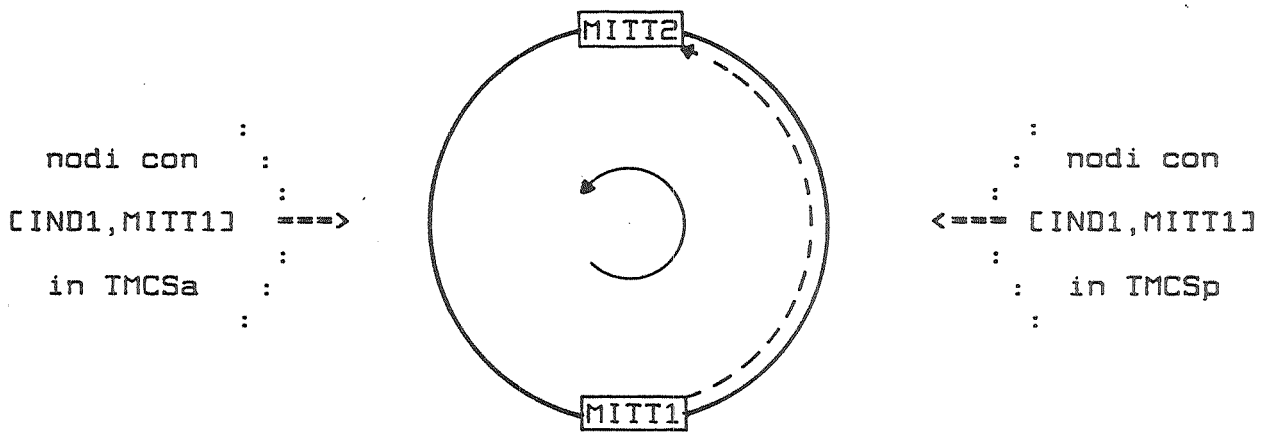

figura .2

Si noti che se non vi fosse la condizione (b), data la asincronia nell'aggiornamento delle tabelle, MITT2 potrebbe fare accettare il suo multicast da un sottoinsieme del gruppo (parte destra della fig. .1), ma non dall'altro che e' ancora impegnato sul precedente. Questo farebbe cadere le assunzioni fatte ed il buon funzionamento del protocollo.

2.3.2 Schema intuitivo del protocollo.

Analogamente al caso omission il messaggio iniziale deve essere ripetuto da un numero di elementi del gruppo sufficiente a garantire sia che chiunque e' corretto lo riceva, sia che tutti diano il commit a quello corretto, malgrado la possibile spedizione di copie errate da parte degli al piu' π_g nodi guasti interni e β_g esterni.

Le assunzioni per un gruppo g_g sono:

(1)- $N_g-\pi_g-\mu_g \geq n_g$

(2)- $n_g-K_g \geq \pi_g+\beta_g$

(3)- ogni nodo che ha ricevuto il messaggio iniziale completo deve ritrasmetterlo, a meno di non averne gia' contato n_g copie.

Le assunzioni (1) e (2) servono ad evitare che i $\pi_g+\beta_g$ nodi guasti di cui tolleriamo l'intervento (e la presenza), diffondendo una stessa copia errata del messaggio la facciano accettare dai μ_g nodi che possono avere perso quella iniziale.

Il parametro K ha lo stesso significato del caso omission ma, poiche' per l'uso della Strong Assumption sui messaggi INIT tutti si rendono conto di quando inizia effettivamente un multicast che li riguarda, il conto dei messaggi persi inizia solo in quel momento e termina non appena si ha la garanzia che il multicast e' completato; K puo' essere percio' mantenuto piu' basso.

Il commit viene dato al termine del round successivo a quello di ricezione del messaggio di INIT, scegliendo la copia contata almeno $\pi_g+\beta_g+1$ volte (o n_g-K_g); le eventuali altre ver-

sioni vengono scartate.

L'algoritmo di base dei protocolli e' cosi' sostanzialmente uguale, con le sole modifiche gia' presentate parlando della tabella TMCS.

Anche per quanto riguarda le prenotazioni il meccanismo non cambia, bisogna pero' inserire nuovi controlli per evitare che nodi guasti tentino di non rispettare l'ordine, scavalcando chi ha diritto (prima non era possibile).

La equazione (2), pero', non e' corretta se non poniamo una limitazione al numero di interventi che i nodi guasti possono fare durante i due round del protocollo, prima di autosospendersi.

E' infatti possibile per i nodi guasti spedire un secondo messaggio nell'intervallo compreso tra la fine del primo round del protocollo (rispetto al mittente) ed il commit dei nodi del gruppo, che e' distribuito nel round successivo.

Poiche' tale messaggio puo' sommarsi agli altri per garantire che la scelta finale sia corretta alla (2) bisogna sostituire la

$$(2') \quad n_J - K_J \geq 2*\pi_J + \beta_J$$

Se pero' si considera un sistema quale quello ipotizzato nel progetto ESPRIT, cioe' con pretese di funzionamento *failstop*, la condizione (2') diviene eccessivamente pesante in quanto l'intervallo pari ad un round intercorrente tra l'inizio del funzionamento errato di un nodo (spedizione del primo messaggio) e la successiva occasione di intervento e' tale da permettere ai meccanismi di check interni di sospendere il nodo stesso.

Si noti comunque che i vincoli (2) e (2') hanno influenza solo sulle dimensioni minime dei gruppi in rapporto ai guasti che si ammettono, e non sulla struttura del protocollo.

Nella figura .3 esemplifico graficamente quanto appena spiegato.

(1) termina il round di ricezione del msg. e si passa al round successivo, puo' intervenire spedendo un primo msg. errato.

(2) termina il round di ricezione del msg., se non ha contato n(i) copie interviene rispedendolo.

(3) termina il round anche per MITT, tutti hanno avuto la occasione per partecipare al protocollo che puo' considerarsi terminato.

(4) se ancora attivo puo' spedire un nuovo msg. falso che verra' raccolto e conteggiato da tutti i nodi successivi fino a MITT.

(5) a questo punto decide quale msg. accettare come corretto, i nodi guasti successivi non potranno influire.

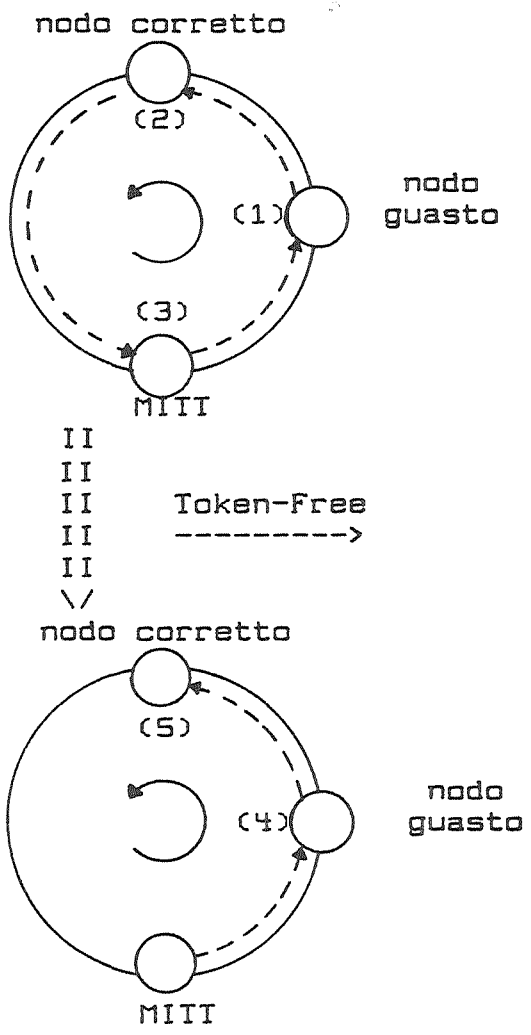

figura .3

2.3.3 Proprieta' del protocollo.

TEOREMA 2.6

Se per un gruppo g_J valgono le equazioni (1) e (2) ed i nodi del gruppo rispettano durante l'esecuzione del protocollo la (3), allora tutti i nodi corretti accetteranno lo stesso messaggio e scarteranno gli altri che eventualmente fossero circolati.

Dim..

Da (1), (2) e (3) si ricava che almeno n_J copie corrette vengono messe in circolazione, poiche' un nodo non guasto puo' perderne al piu' K_J-1 senza autosospendersi, ognuno ne contera' almeno n_J-K_J copie che e' il limite minimo per la accettazione.

Viceversa al piu' $\pi_J+\beta_J$ copie uguali di un diverso messaggio rispetto a quello originale possono essere raccolte prima del momento in cui si deve decidere, e quindi questi non sara' mai accettato.

[]

TEOREMA 2.7

Non e' possibile che un nodo corretto accetti di partecipare ad un multicast collidente con uno non ancora terminato.

Dim..

La dimostrazione si ricava banalmente da quanto esposto nei paragrafi relativi alla tabella TMCS.

[]

TEOREMA 2.8

Entro un tempo finito ogni nodo j riesce ad attivare i multicast della propria lista di attesa.

Dim..

Grazie al meccanismo di *"circolazione del privilegio"*, garantito anche dal teorema precedente, la situazione e' perfettamente simmetrica al caso omission e la dimostrazione ricalca quella del teorema 2.3, ove si consideri che solo il nodo che ha fatto la prenotazione puo' attivare il multicast relativo, e che perdera' il diritto se non lo fara' nei tempi corretti.

[]

TEOREMA 2.9

Per ogni multicast di un gruppo j il numero di messaggi necessari varia da un minimo di n_j ad un massimo n_j+K_j-1.

Dim..

Dalla condizione (3) si ricava direttamente che il minimo e' pari ad n_j, considerando poi che un nodo puo rimanere attivo perdendo al piu' K_j-1 messaggi si ricava anche il massimo che e' appunto pari a n_j+K_j-1.

[]

Si noti che tutte le considerazioni fatte in 2.2.3 sono valide anche in questo nuovo contesto.

2.3.4 Recovery di un nodo & ripristino del Token-Free.

Relativamente al recovery di un nodo caduto le valutazioni fatte in 2.2.4 si possono applicare anche in presenza di guasti malicious, sara' la specifica routine di riacquisizione dei dati necessari alla connessione al sistema che dovra' tenere conto di questa diversita'.

Il ripristino del Token-free viene effettuato con i normali meccanismi descritti in [BCKKM], successivamente bisogna evitare di attivare ed accettare durante il primo round nuovi multicast, esattamente come nel caso omission.

Un problema che puo' sorgere e' la partecipazione per la seconda volta di nodi guasti a multicast che ancora devono terminare, ma questo e' analogo a quanto gia' discusso in precedenza: se si usa la condizione (2') il problema non sorge comunque, se si usa la (2) si e' supposto che i nodi guasti non riescano a spedire piu' di messaggio errato prima di sospendersi, e cio' garantisce anche in questo caso.

2.3.5 Schema formale del protocollo.

Nello schema sotto riportato vengono sottintesi alcuni controlli, quale la autenticita' del mittente, che si suppone siano effettuati ad un livello piu' basso.

Si noti che per maggiore chiarezza e' stata concentrata nella receive(Token-Free) la decisione di cosa deve e puo' essere fatto mentre nella realta', per ovvi motivi di efficienza e velocita' di risposta, essa e' rivalutata ogni volta che arrivano informazioni che modificano lo stato precedente.

nodo i-esimo

```
[ receive (INIT,msg,mitt,IND)                          ;i app. a IND
    ----->if <esiste gia' [IND] in TPREN, attiva e
              con diverso mittente>
           or Collide([IND],TMCSa)
           or (Collide([IND],TMCSp)
                & not Cor.Mitt.(<mitt.con cui collide in TMCSp>
                                ,mitt))
            then  <rifiuta>              ;vale almeno una delle con-
                                         ;dizioni di errore.
            else begin
             if msg ≠null then begin
                           MCSadd<---insert([msg,mitt],Lcommit)
                           MCSflag<---IND;cont<---1;
                          end
                          else begin
                               M1<---1;     ;i non ha raccolto la
                                            ;parte informazioni.
                               MCSflag<---IND;
                             end;           ;uso MCSflag per attivare
                                            ;il conto degli errori in
                                            ;lettura.
             ADD (TMCSa,[IND,mitt]);
             if [IND,mitt] IN TPREN then <cancellalo>;
            end;
```

```
[]receive (RIP,msg,mitt,IND)                    ;i app. ad IND
                                                ;mitt e' quello ori-
                                                ;ginario,non chi ha
    ----->if <esiste [IND,mitt] IN TMCS>        ;ritrasmesso.
             then x<---insert ([msg,mitt],Lcommit);
                                  ;x e' una variabile dummy,  la
                                  ;insert incrementa automatica-
                                  ;mente il cont. associato agli
                                  ;elementi di Lcommit.


[]receive (PREN,mitt,IND)
    ----->insert-pren([mitt,IND],TPREN)
                                  ;la  insert-pren controlla che
                                  ;il  mittente non  abbia  gia'
                                  ;fatto altre prenotazioni.


[]receive (errore-lettura, type)
    ----->case type of
             INIT: STOP
             PREN: STOP
             RIP :  if MCSflag≠0          ;la  perdita di una ripe-
                                          ;tizione mi interessa so-
                                          ;lo se sto partecipando a
                                          ;un multicast.
                          then begin
                               M1<---M1+1;
                               if M1=K then STOP
                               end;


[]receive (INIT,mitt,IND)                 ;i non app. ad IND, ma lo
                                          ;conosce.
    ----->if <esiste gia' [IND] in TPREN, attiva e
              con diverso mittente>
           or Collide([IND],TMCSa)
           or (Collide([IND],TMCSp)
               & not Cor.Mitt.(<mitt.con cui collide in TMCSp>
                                ,mitt))
            then  <rifiuta>               ;vale almeno una delle con-
                                          ;dizioni di errore.
            else begin
                   ADD (TMCSa,[IND,mitt]);
                   if [IND,mitt] IN TPREN then <cancellalo>;
                 end;
```

```
[]receive (Token-Free)
    ----->if MCSflag ≠0 & MCSadd ≠0
             & Lcommit(MCSadd).cont<n(MCSflag)
               then begin
                    [msg,mitt]<---Lcommit(MCSadd).msg;
                    Send (RIP,[msg,mitt],i,MCSflag);
                    MCSadd,MCSflag,cont,M1<---0
                    end
               else begin
                     <scegli cosa fare valutando i dati a  dispo-
                     sizione e le vecchie condizioni>;
                    MCSflag<---0;
                    end.
          <valuta se vi e' un msg. del round precedente a cui da-
          re il commit ed altri da scartare>;
          TMCSp<---TMCSa;
          TMCSa<---Null+<eventuale coppia rappresentante il  mul-
                         ticast appena attivato>;
          <cancella da TPREN le prenotazioni scadute>;
].
```

figura .4
schema formale del protocollo tollerante malicious faults

Per quanto riguarda la suddivisione a livelli secondo il modello ISO/OSI niente di interessante si puo' aggiungere rispetto a quanto proposto in 2.2.7.

2.4 CONFRONTO CON [BD.1,2] E VALUTAZIONI COMPARATIVE.

Entrambi i protocolli proposti si ispirano, per quanto concerne l'algoritmo di base utilizzato, agli articoli di Babaoglu e Drummond [BD.1,2].

Le diversita' ed i notevoli arricchimenti rispetto al modello di base sono giustificati sia da alcune limitazioni fisiche, l'uso di un solo Token-Ring, sia da obiettivi piu' ampi, la gestione di multicasts e non solo broadcasts tra gruppi diversi con un grado accettabile di parallelismo ogni volta che i vincoli fisici lo permettano.

La scelta del Token-Ring ha portato ad una rivalutazione dei parametri di affidabilita', con l'introduzione di μ e K, come visto nell'introduzione a questo capitolo.

L'obiettivo della gestione di piu' multicasts, cioe' di comunicazioni tra sottoinsiemi della rete che possono non essere distinti, ha invece portato alla creazione del sistema di tabelle ed ai meccanismi di *"passaggio circolare del privilegio"* e di *"prenotazione"*.

Si noti che l'avere utilizzato il termine *"round"* a denotare l'intervallo temporale intercorrente tra due passaggi consecutivi del Token-Free (con significato locale ad ogni nodo) non e' casuale, il riferimento e' all'analogo termine definito da Babaoglu.

Per Babaoglu il round rappresenta l'intervallo temporale necessario affinche' ogni nodo possa comunicare con tutti gli altri; questo, se si pensa al funzionamento del Token-Ring, e' certamente vero anche per la mia definizione.

La diversita' fra le due visioni sta nella globalita' o localita' del concetto: in [BD.1,2] si assume l'esistenza di un meccanismo che sincronizzi tutti i processori della rete con un funzionamento a *"lock-step round"*, io sfrutto le caratteristiche della rete ad anello per fare evolvere in modo indipendente e sostanzialmente asincrono, ma ordinato, lo "stato" dei singoli nodi.

Ognuno conosce il proprio *round*, che e' diverso da quello di tutti gli altri, ma sufficiente ad ottenere un corretto comportamento globale del sistema.

Negli articoli di Babaoglu e Drummond non vengono poi affrontati due problemi che considero importanti:

(1)-Come possono avvenire comunicazioni parallele all'interno di un singolo gruppo.

Esaminando i protocolli da loro proposti ci si rende conto che, rispettando le definizioni date, non e' possibile piu' di una comunicazione per volta, in particolare solo al termine di una puo' essere attivata la successiva.

Per ottenere un maggiore parallelismo bisognerebbe rilassare il concetto di *round* in modo da comprendere non piu una comunicazione da ogni nodo ad ogni nodo, ma tante comunicazioni quanto e' il grado di parallelismo desiderato.

E' infatti chiaro che con le assunzioni fatte e per l'algoritmo utilizzato il singolo nodo non puo' partecipare a piu' di un multicast (o broadcast) perche' fisicamente non puo' effet-

tuare durante il *round* tutte le comunicazioni che sarebbero richieste.

Questo aspetto e' ancora piu' chiaro se si pensa al Token-Ring: quando un nodo riceve il Token-Free puo' ripetere al piu' un messaggio, e quindi se stesse partecipando attivamente a piu' di un multicast dovrebbe sceglierne uno e rimandare gli altri ai round successivi.

Per questi vincoli fisici, ma soprattutto per le implicazioni di efficienza ed affidabilita', ho scelto di sequenzializzare le attivazione di multicast collidenti, preservando la idea originale di *round*.

Per ottenere cio' ho aggiunto i vari meccanismi di *"sincronizzazione"* globale.

(2)-Come trattare comunicazioni tra sottoinsiemi distinti della rete, cioe' multicasts parallelizzabili in quanto riguardanti nodi diversi.

Per risolvere questo problema ho introdotto il meccanismo di rilevamento delle collisioni (per il caso malicious) o comunque dei gruppi attivi (per entrambi i casi) e quello delle prenotazioni, essi permettono sempre di avere una visione dello "stato" delle comunicazioni che consente di individuare le azioni corrette.

Nella letteratura non esistono validi termini di paragone con queste proposte, essendo questo campo ancora sostanzialmente inesplorato.

Rimango percio' sugli articoli di Babaoglu e Drummond anche se per dare una comparazione mi devo limitare a considerare il singolo multicast, visto isolatamente.

I costi in messaggi sono simili, pur se il confronto deve essere cauto essendo diversi alcuni dei parametri usati, mentre per la durata il termine e' fissato sempre in due round che hanno pero', come piu' volte detto, un significato indefinito in [BD] e ben delineato per i protocolli proposti.

Riassumo ora in una tabella le caratteristiche di performance dei due protocolli, esprimendola in termini dei parametri K ed n (che ingloba π e β).

Si noti che lo stato di "intasamento" della sottorete di comunicazione non ha alcuna influenza sul numero di messaggi necessari, mentre aumentera' la durata del round e quindi dei protocolli.

	Caso Omission		Caso Malicious	
	minimo	massimo	minimo	massimo
m :	K	2*K-1	n	n+K-1
t :	2 round		2 round	

tabella 4.1

I valori di K, n, π, β e μ saranno generalmente dipendenti dal singolo gruppo e varieranno da multicast a multicast.

* BIBLIOGRAFIA *

[BCKKM] W. Bux, F. Closs, K. Knemmerle, H. Keller and H. L. Mueller, *The Token Ring*, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg.

[BD.1] O. Babaoglu, R. Drummond, *Streets of Byzantium: Network Architecture for fast Reliable Broadcast*, IEEE Transaction on Software Engineering, Vol.se-11 N.6 June 1985 pp 546-554.

[BD.2] O. Babaoglu, R. Drummond, P. Stephenson, *The impact of Communication Network Property on Reliable Broadcast Protocols*, Research Report, Dep. of Comp. Science Cornell University (Ithaca New York).

[BJRA] K. P. Birman, T. A. Joseph, T. Raenchle, A. E. Abbedy, *Implementing fault Tolerant Distributed Objects*, (IEEE 1984 ottobre), Proceeding "Fourth Symp. on Reliability in Distributed Software and Data Base Systems".

[BS] R. Banerjee, W. D. Shepherd, *The Cambridge Ring*, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg.

[C] E. C. Cooper, *Circus: A Replicated Procedure Call facility*, Proceeding "Fourth Symp. on Reliability in Distributed Software and Data Base Systems", IEEE Computer Science, order N.564.

[CASD] F. Cristian, H. Aghili, R. Strong, D. Dolev, *Atomic Broadcast: from simple message diffusion to Byzantine Agreement*, 1984-IBM research report, 1985-IEEE pp 200-206 (revisione).

[CM] Jo-Mei Chang, N. F. Maxemchuk, *Reliable Broadcast Protocols*, ACM Transaction on Computer Systems Vol.2 N.3 (1984) pp 251-273.

[ESPRIT] *Esprit 1985 Workplan, Project Proposal, The Connection Machine, A Dependable open Distributed System Architecture*, Parte 2.

[G] P. E. Green Jr, *An introduction to Network Architectures and Protocols*, IEEE Trans. on Com.s Vol.com-28, N.4 April 1980 pp 413-424.

[Lam] L. Lamport, *Using Time instead of Timeout for fault-Tolerant Distributed Systems*, ACM Trans. on Programming Languages and Systems, Vol.6 N.2 April 1984 pp 254-280.

[LL.1] G. LeLann, *Distributed Real-Time Processing*, 1985 BBC Internal Symposium.

[LL.2] G. LeLann, *Syncronization*, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg.

[LL.2] G. LeLann, *Real Time Protocols*, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg.

[P] F. Panzieri, Communication Protocols, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg.

[Ran] B. Randall, *Fault Tolerance & System Structuring*, Computing Laboratory-University of Newcastle Upon Tyne, Number 189 Dec. 1983.

[SA] A. Segall, B. Awerbuch, *A Reliable Broadcast Protocol*, IEEE Trans. on Communications, Vol.com-31 N.7, July 1983.