Chat P2P

Architettura: ring (singolo) bidirezionale (la bidirezionalità è usata solo nel caso della procedura di disconnessione volontaria, non per l’invio dati)

Considerazioni preliminari:

* ogni nodo è identificato da un nickname in formato striga
* la socket di recezione conosce l’indirizzo di chi manda il messaggio

La comunicazione p2p avviene tramite UDP ma con implementazione di un protocollo di acknowledge (per gestione disconnessioni impreviste).

NB: Ogni nodo NON ha la conoscenza di tutti i nickname della rete

## Connessione alla rete - JOIN

Ogni nodo della rete può fare da accesso alla rete (non c’è nodo genesys). Un host x che vuole connettersi deve conoscere a priori il riferimento (ip&porta) di un qualsiasi nodo della rete, sia y, e chiamare la procedura di join (connessione) indicando il nickname (stringa) che vorrebbe avere.

L’inserimento del nuovo host avviene sempre tra il nodo che riceve il join e il successivo.

A questo punto parte la procedura di Discovery per sapere se il nickname è già in uso o è disponibile. Il protocollo discovery è qui descritto:

DISCOVERY: y manda nella rete un messaggio DISCOVERY\_QUERY che ha come destinatario il nickname proposto da x. Se e solo se il nickname è libero allora il messaggio ritorna identico ad y (percorrendo tutta la rete). Se è presente un altro nodo con tale nickname, questo riceverà il messaggio e manderà una DISCOVERY\_ANSWER ad y (destinazione) informandolo che il nickname non è disponibile.

Prosegue il JOIN. Supponiamo che il nodo successivo a y sia z, il precedente sia t, e che y abbia i seguenti attributi:

|  |  |
| --- | --- |
| IP\_next | Ip di z |
| PORT\_next | Porta di ricezione di z |
| IP\_prec | Ip di t |
| PORT\_prec | Porta di t |

Come conseguenza al messaggio di join, y invia ad x IP\_next e PORT\_next (relativi a z), e x li imposterà come i propri attributi, inoltre imposta IP\_prec e POST\_prec a quelli di y. Successivamente y modifica i valori IP\_next e PORT\_next impostandoli a quelli di x (noti dalla socket). x comunica a z di impostare IP\_prec e PORT\_prec ai riferimenti di x stesso.

## Disconnessione dalla rete

Sono previste due possibilità di disconnessione. Nella prima un nodo si disconnette “volontariamente”, e avvisa la rete della sua disconnessione, che di conseguenza si riorganizza. Nella seconda un nodo cade improvvisamente senza avvisare la rete, che quindi dovrebbe poter riconoscere la disconnessione e riorganizzarsi autonomamente di conseguenza.

### Disconnessione volontaria

L’architettura può gestire la disconnessione volontaria degli host. Questa situazione è l’*unica* in cui vengono usati i back links.

sia x un host che abbia nodo precedente y e nodo successivo z, e che voglia abbandonare la rete; x manda ad y un messaggio di tipo QUIT comunicando IP\_next e PORT\_next (riguardanti z), che vengono memorizzati da y come propri IP\_next e PORT\_next. Analogamente comunica a z IP\_prec e PORT\_prec riguardanti y, che vengono memorizzati da z. Infine si disconnette.

### Disconnessione imprevista

Attualmente l’architettura può gestire la disconnessione imprevista solo in un modo primitivo e non locale.

## Messaggi

Il formato dei messaggi è una stringa formata come segue:

“tipo\_messaggio#nick\_mittente#nick\_destinatario#messaggio”

|  |  |
| --- | --- |
| STANDARD |  |
| JOIN |  |
| DISCOVERY\_QUERY |  |
| DISCOVERY\_ANSWER |  |
| QUIT |  |

## Sviluppi futuri

1. ring con ridondanza (link al vicino del vicino)
2. comunicazione bidirezionale anche per dati per aumento di efficienza della rete
3. ping continui per capire se i nodi sono attivi o sono caduti
4. crittografia e hashing: il mittente cripta con chiave pubblica del destinatario, tutti gli hop possono provare a decriptare con la propria chiave privata, quindi verificare se sono il destinatario…

Prima di inviare il messaggio, il mittente pinga il destinatario per sapere se è vivo?

Il mittente sa che il messaggio è stato ricevuto?

Fare in modo che i nomi siano più o meno standard

Su ogni host decidere la gestione dei messaggi (coda, pila, altro…?)

### Aspetti non coperti

Un nodo dovrebbe non poter alterare il messaggio (i dati) inviato da un altro nodo, ovvero “parlare per conto di un altro”.

Non è evitabile che un nodo possa scegliere si scartare un messaggio che transita per lui