Progetto Reti Informatiche

Dario Pagani 585281

I programmi lavorano in modalità verbose su `stderr`, per "zittirli" reindirizzarlo, es `./peer.sh 2> /dev/null` `./boot.sh` per avvia un test 5 peer. Per aggiungerne altri usare `./peer.sh` e battere `start 127.0.0.1 7575` Non usare ^C per terminare il Discovery Server. Preferire ^D ovvero battere '`quit`'

Tipi di dati

`peer_definitive_id`	`uint16_t`	Identificativo univoco di un nodo. Non cambia tra le sessioni
`peer_session_id`	`uint64_t`	Coppia <indirizzo IPv4, porta>. Usato per tenere traccia dei nodi senza un id o si deve comunicare un indirizzo di un nodo

Ingresso nella rete

Ogni qualvolta un nuovo *nodo* vuole entrare a far parte di una rete distribuita gestita da un *Discovery Server* effettua un triplo scambio scambio di mano col ds: Invia `PEER_WANTS_REG` al ds fintanto non riceve `DS_ACK_REG` dal ds ovvero raggiunge il timeout, ad ogni `DS_ACK_REG` il peer risponde con un `PEER_ACK_REG` ed entra a far parte della rete.

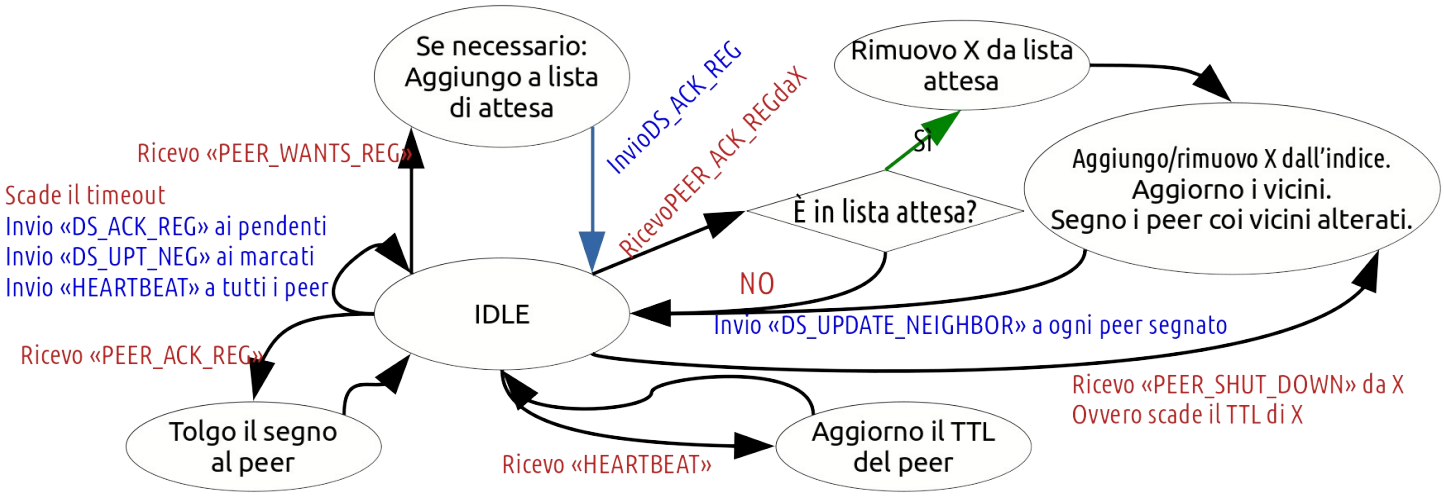

Figura 1: Stati interi del DS. La logica della console omessa

Protocollo tra Discovery Server e Nodi

Usa il protocollo UDP/IPv4. I messaggi sono a lunghezza fissa di 48 byte. La prima parola quadrupla individua il tipo di messaggio, le altre sette cambiano in funzione del tipo. I principali messaggi:

- ➔ `PEER_WANTS_REGISTRATION` contiene l'id univoco se il peer à fatto parte della rete in passato
- ➔ `DS_ACK_REGISTRATION` contiene l'id univoco assegnato dal ds al peer
- ➔ `DS_UPDATE_NEIGHBORS` contiene i due nuovi vicini del peer in formato `peer_session_id` e l'orario del messaggio in modo che il peer e ds possano scartare aggiornamenti arrivati in ritardo rispetto ad altri.
- ➔ `HEARTBEAT` e `PEER_ACK_NEIGHBORS` contengono un orario fissato dal ds

Definito in `shared/protocol.h`.

Topologia della rete nodo a nodo

La rete è organizzata ad anello, come mostrato dagli archi neri in Figura 3. I nodi vengono salvati nel *Discovery Server* in una lista ordinata per `peer_session_id`, i vicini vengono assegnati in questo modo: l'arco uscente è il nodo maggiore e l'arco entrante è il nodo strettamente minore. Purtroppo, questa organizzazione fa si che ci siano due nodi (la testa e la coda della lista) che potrebbero avere una distanza molto grande.

Svantaggio di questa topologia è il tempo di attraversamento che potrebbe essere molto lungo in caso di tanti nodi: ad esempio si consideri di avere 100 nodi e un tempo di attraversamento nodo a nodo medio di 100ms allora il tempo di attraversamento sarà O(10'000ms) = O(10s) che è abbastanza grande.

Il vantaggio della topologia è la semplicità di realizzazione dal lato *Discovery Server*.

Protocollo tra i nodi

Ogni nodo ha, quando c'è più di un peer, due sessioni TCP dette di **controllo** con i due vicini. Per l'aggiornamento dei registri viene aperto una sessione TCP temporanea detta **dati** tra il nodo che vuole inviare i dati e il nodo che dovrà scaricarli.

La connessione di controllo usa messaggi a lunghezza fissa di 56 bye.

La connessione dati usa messaggi a lunghezza variabile. Questo canale viene aperto da un nodo per inviare aggiornamenti ai registri stato e dati di un altro nodo, al termine dell'operazione il canale viene chiuso. Il formato della trasmissioni è illustrato in Figura 2. Le voci di registro della seconda parte sono di autori che sono stati preannunciati nella prima parte. Definito in `peer/net.h`.

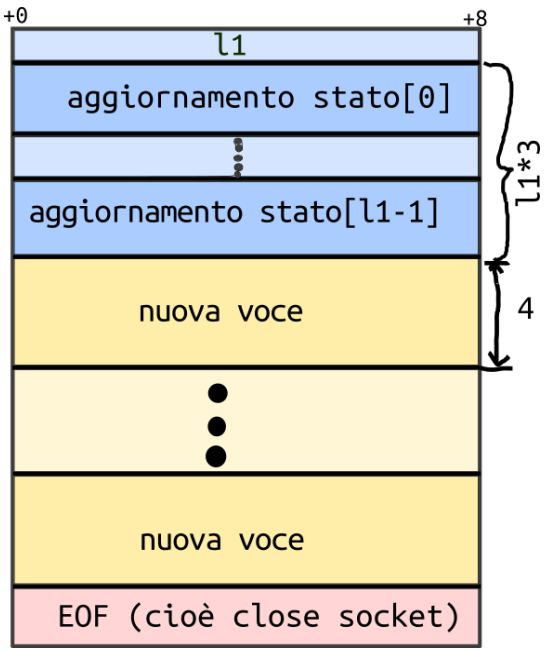

Figura 2: Trasmissione su canale DATA

Gestione del registro

Il **Registro di Stato** è un file unico per ogni nodo che contiene l'id definitivo del peer generato dal ds alla prima connessione e, per ogni peer che fa parte della rete l'orario UNIX dell'ultimo aggiornamento dati cioè fino a che punto del tempo i dati provenienti da quel peer sono completi.

I **Registri dei Dati** sono file, uno per giorno, che contengono i dati relativi a casi e tamponi. Ogni voce del registro è una tupla <autore (`peer_definitive_id`), orario UNIX di creazione, tipo, quantità>. Per semplicità le voci di registro sono memorizzate nel file della giornata in senza alcun criterio di ordinamento particolare. Il nome del file fornisce già un ordinamento intrinseco per data, che è ciò che serve per le interrogazioni.

Interrogazioni

Si osservi la rappresentazione di una possibile evoluzione di un'interrogazione del peer 1 in Figura 3. Se il risultato non è presente in cache allora inoltra l'interrogazione ai due vicini destro e sinistro sul canale di controllo. Se neanche le loro cache lo hanno memorizzato allora fa partire fa partire delle richieste di flooding, una per ogni autore di cui mancano informazioni, che attraverseranno tutti i peer connessi della rete. Se un peer è capace di soddisfare parte di una richiesta di flooding apre una connessione TCP di tipo data con chi ha avviato il flooding e gli spedisce parte dei dati ed inoltra un pacchetto di flooding modificato.

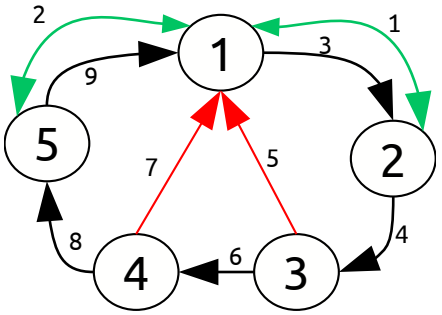

Figura 3: Caso peggiore di una interrogazione

Il pacchetto di flooding è composto da `peer_session_id` per identificare il mittente, dall'id del nodo di cui mancano informazioni nel registro dati e dai margini temporali interessati. Il limite inferiore del periodo viene aumentato dai peer mano a mano che essi riescono a spedire i dati mancati, se i margini temporali coincido vuol dire che si è riusciti a completare la richiesta.

Il flooding termina quando tutti i pacchetti hanno fatto il giro e sono tornati al mittente.

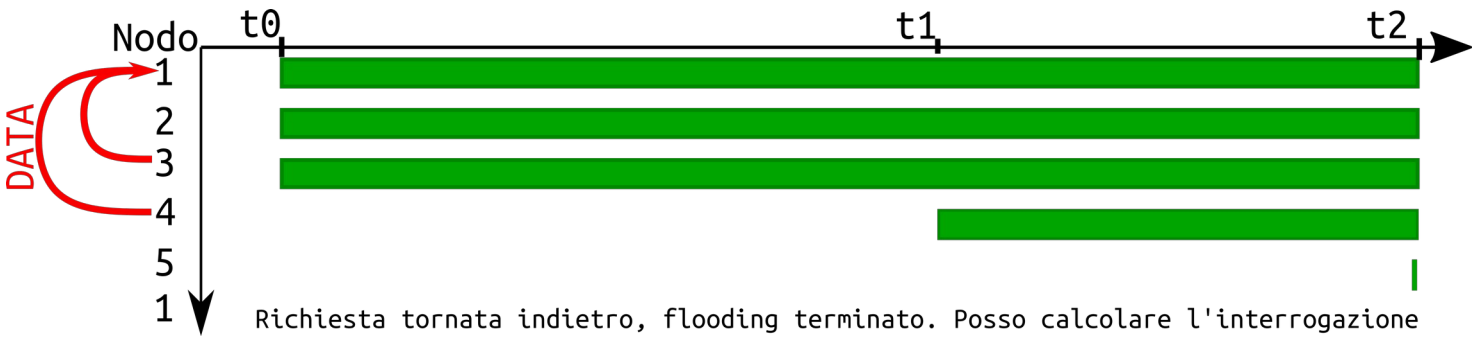

Figura 4: Esempio pacchetto di flooding: il 3 ha potuto soddisfare fino a t1, il 4 fino a t2