Universita' degli Studi di Napoli "Parthenope"
Facolta' di Scienze e Tecnologie
Corso di laurea in Informatica

Relazione di Reti di calcolatori Laboratorio

State Channels - Hybrid decentralized P2P Network with routing algorythm

Docente
Alessio Ferone

Candidati

- Silvio Vincenzo
  0124001735
- Lombardi Andrea
  0124001707
- Riccio Davide Cosimo
  0124001740

Anno Accademico 2019-2020

Indice

Capitolo 1

Descrizione del progetto

Il progetto consiste nel creare un sistema software in grado di connettersi ad una overlay network P2P. In particolar modo si fa riferimento ad una rete P2P di State Channels.

1.1 State Channnel

Uno state channel e' un canale tra due peer in grado di conservare uno stato finche' quest'ultimo non viene chiuso. Lo stato e' costituito dal bilancio dei due peer in seguito allo scambio di valuta digitale.

1.2 Requisiti funzionali

I requisiti funzionali posseduti dal sistema sono elencati di seguito:

1. Trasferire valuta dal proprio bilancio ad un altro peer;
2. Aprire uno state channel diretto con un altro peer se e solo se non esiste un cammino che congiunge i due peer all'interno dell'overlay network.
3. Visualizzare i propri state channel;
4. Chiudere uno state channel garantendo la consistenza dei bilanci e la stabilita' del sistema.

Capitolo 2

Descrizione e schemi dell'architettura

Il sistema si basa su un modello di progettazione di reti peer-to-peer ibridamente decentralizzato.

2.1 Hybrid Decentralized P2P

Un modello ibrido comune consiste nell'avere un server centrale che aiuta i peer a trovarsi. E' stato favorito rispetto ai modelli con reti pure non strutturate o alle reti pure strutturate perche' alcune funzioni, come la ricerca, richiedono una funzionalita' centralizzata ma beneficiano dell'aggregazione decentralizzata di nodi fornita da reti non strutturate.

2.1.1 Vantaggi e svantaggi

Come sopracitato la ricerca di un peer e' nettamente piu' efficiente rispetto ad un sistema puramente decentralizzato. L'efficienza e' accentuata dal fatto che il server ha una singola responsabilita': esso e' contattato solo nel momento in cui un peer chiede le info di un altro peer per creare uno state channel. Questo avviene solo quando il peer richiedente non possiede un cammino che lo porta alla destinazione. Indi per cui se un peer, una volta entrato nella overlay network, possiede un cammino che lo porta alla destinazione scelta, non avra' la necessita' di contattare il server. Il server rappresenta il SPF (Single point of failure), ma

la rete continua a garantire il servizio di scambio transazioni utilizzando gli state channel aperti prima del crash; ovviamente non sara' possibile crearne di nuovi.

Capitolo 3

Descrizione e schemi del protocollo applicazione

In questo capitolo verranno descritti i protocolli applicazione. Tutte le comunicazioni utilizzano il protocollo di trasporto TCP, ergo le azioni di apertura del canale di comunicazione saranno sottintese (connect, accept etc).

3.1 Protocolli di comunicazione Server - Peer

3.1.1 Nofity Connection

All'avvio di un peer, esso comunica al server le proprie informazioni necessarie ad instaurare connessioni con altri peer dell'overlay network. Il server a ricezione avvenuta provvede ad aggiungere le informazioni in un apposito file chiamato "info.txt".

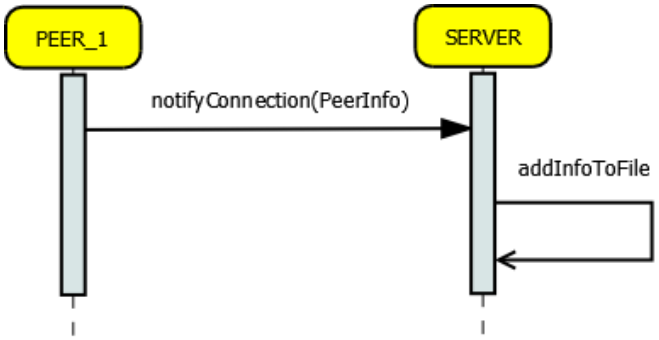

Figura 3.1:

Il pacchetto PeerInfo identifica le informazioni necessarie alla comunicazione di un peer con gli altri, ed e' cosi' strutturato:

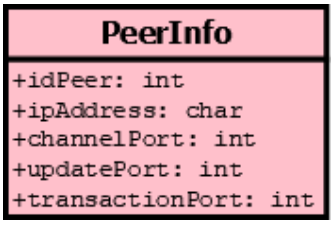

Figura 3.2: Struct PeerInfo

- idPeer: id del peer;
- ipAddress: indirizzo ip della macchina del peer;
- channelPort: porta su cui offre il servizio di creazione di state channels;
- updatePort: porta su cui offre il servizio di aggiornamento delle tabelle di routing.
- transactionPort: porta su cui offre il servizio di gestione delle transazioni nella Overlay Network.

3.1.2 Open state channel

Quando un peer vuole effettuare una transazione con un altro peer, ma non esiste nessun cammino di state channel percorribile tra i due, allora e' necessario aprire un nuovo state channel.

(in figura 3.3) Il peer 1 intende aprire uno state channel con il peer 2. Ottiene le informazioni di connessione attraverso la funzione "getPeerInfo(int idPeer)" (ove idPeer e' l'identificativo del peer di cui si vogliono ottenere le specifiche di connessione). In seguito attraverso la funzione "requestStateChannel(ConnectionRequest cr)", richiede al peer 2 la possibilita' di aprire uno state channel. Il peer 2 a sua volta ha la facolta' di rifiutare la connessione oppure di accettarla notificando la decisione attraverso la funzione "reply(ConnectionRequest cr)".

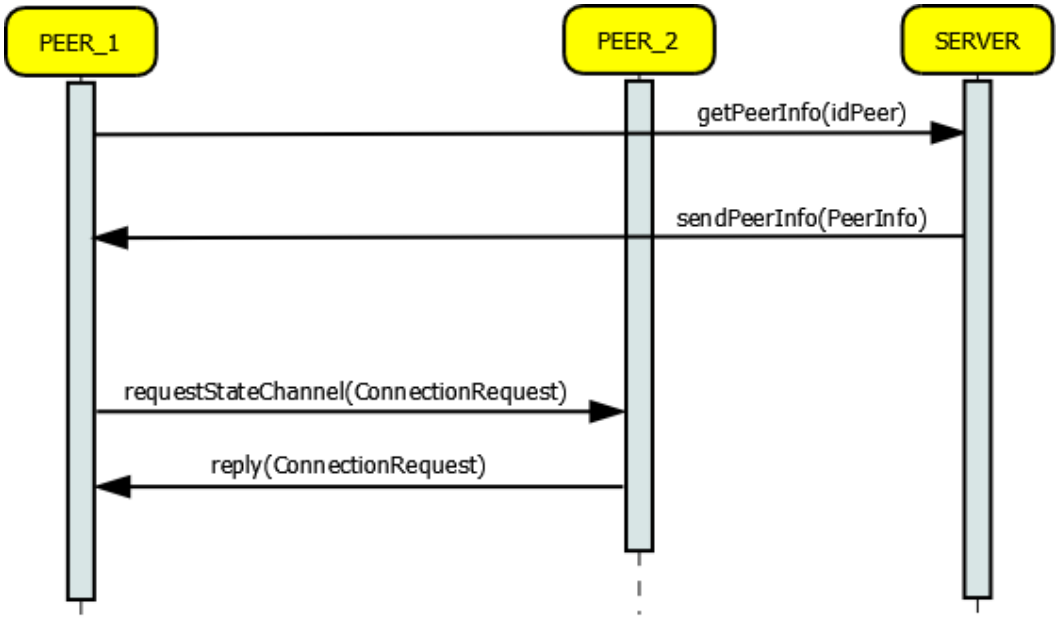

Figura 3.3:

Il pacchetto ConnectionRequest e' utilizzato per la richiesta di connessione e creazione di uno state channel.

Figura 3.4:

Ed e' cosi' strutturato:

- confirmed: campo che puo' assumere 3 valori, 1 per confermare la creazione di uno SC, 0 per rifiutarlo e -1 per chiuderlo;
- usedValue: valore impiegato nello state channel da parte di un peer;
- info: struct PeerInfo necessaria alla memorizzazione delle specifiche di connessione del peer che intende aprire uno state channel.

3.2 Protocolli di comunicazione Peer - Peer

Di seguito saranno riportati i protocolli di comunicazione che avvengono in maniera diretta tra i peer, senza comunicazione col server.

3.2.1 Chiusura di uno state channel

Un peer che vuole chiudere uno state channel, inviera' una notifica (non una richiesta) di chiusura al peer in questione. Quest'ultimo ricevera' un pacchetto di tipo ConnectionRequest con il campo "confirmed" uguale a -1. Una volta avvenuta la comunicazione, entrambi i peer provvedono ad eliminare reciprocamente lo State Channel appena chiuso dalla loro struttura dati "myStateChannels" (lista monodirezionale).

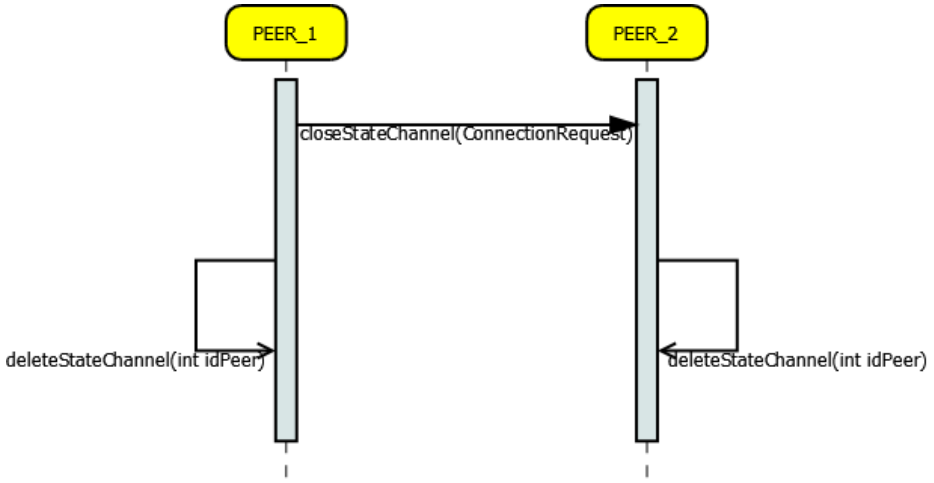

Figura 3.5:

3.2.2 Update della tabella di routing

Per migliorare l'efficienza dell'esplorazione nell'overlay network si e' scelto di implementare un algoritmo di routing. Esso consiste nell' individuare velocemente gli state channels da utilizzare per raggiungere un dato peer. Periodicamente ogni peer chiede ai peer direttamente collegati quali sono i nodi raggiungibili da essi. Si mantiene una hashTable di peerToReach e per ognuno di essi vengono memorizzati gli state channels che possono essere esplorati per raggiungerli.

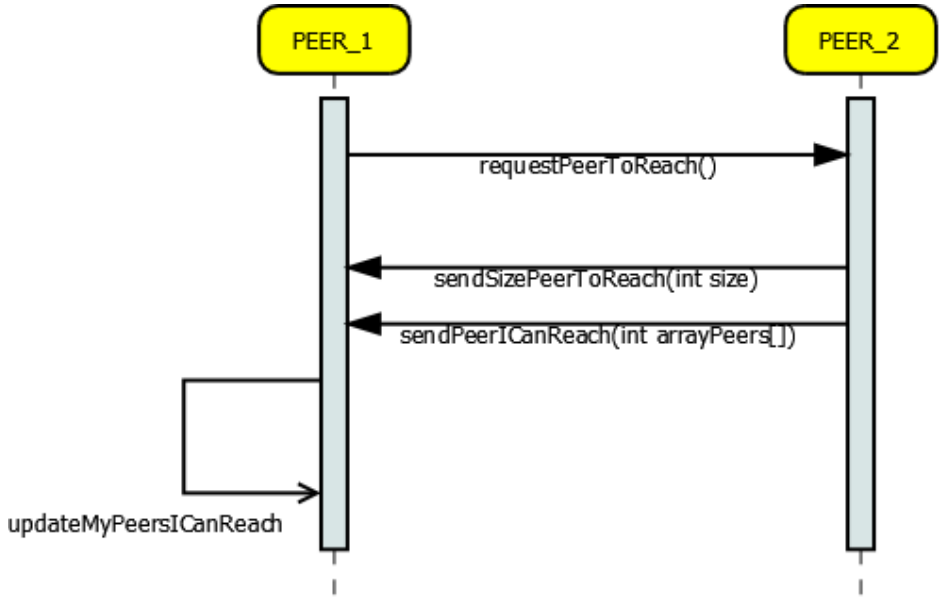

Figura 3.6:

Di seguito un esempio di overlay network con annessa Hashtable dei Peer che si possono raggiungere per un peer.

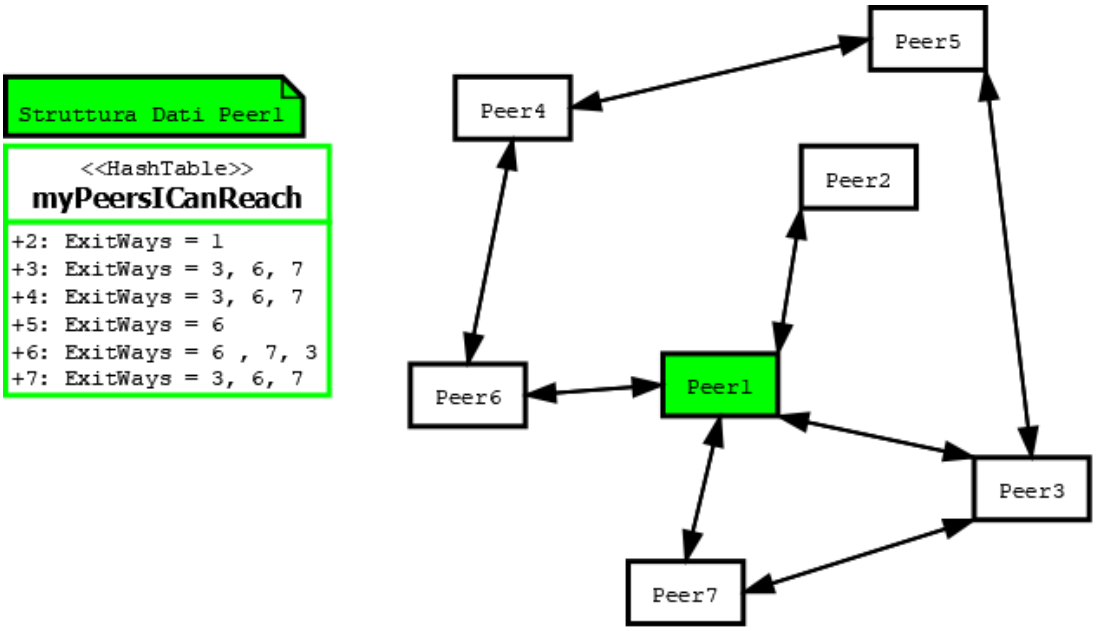

Figura 3.7: Esempio dell'algoritmo di routing e del funzionamento

3.2.3 Transazioni

Il sistema software proposto gestisce lo scambio di valuta tramite transazioni tra peer. E' stato ipotizzato che ogni transazione possa avere uno tra i tre stati di seguito proposti:

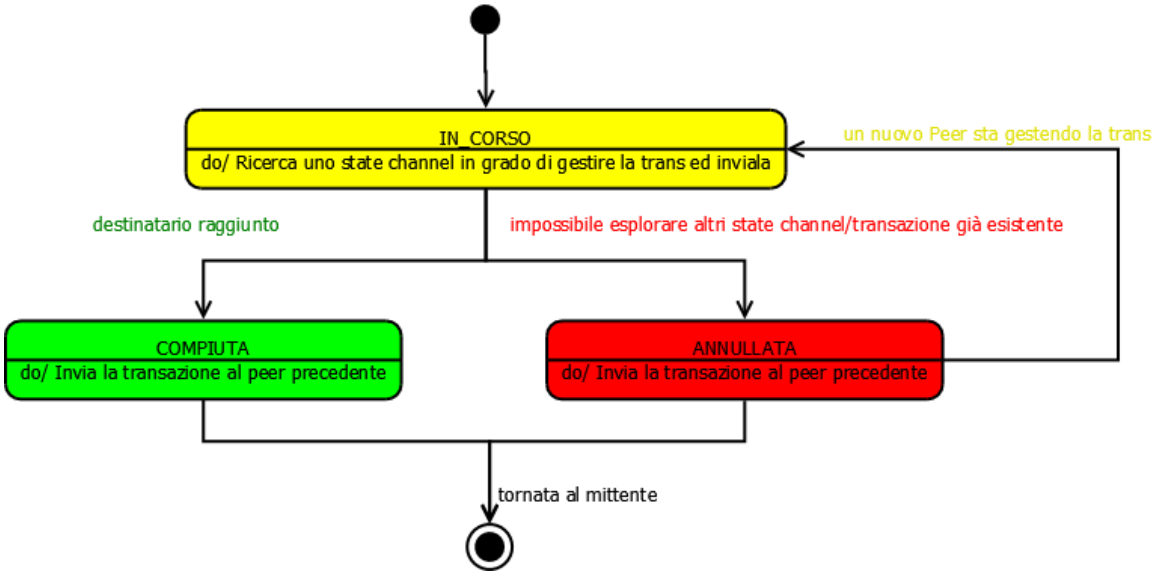

Figura 3.8: Diagramma degli stati di una transazione che circola nell'overlay network.

Lo stato di una transazione e' contenuto all'interno del pacchetto:

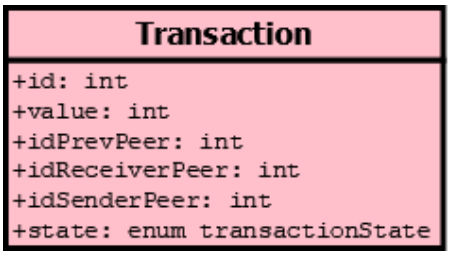

Figura 3.9:

- id e idSenderPeer: la coppia identifica univocamente una transazione all'interno dell'overlay network. L'id rappresenta la sequenza delle transazioni inizializzate da un singolo peer. Mentre l'idSenderPeer e' l'identificativo

univoco del peer. Indi per cui la circolazione di transazioni con lo stesso id all'interno dell'ON non genera incongruenze.

- idPrevPeer: id del peer da cui si riceve la transazione ed a cui si dovra' reinviare l'esito della stessa.
- idReceiverPeer: destinatario della transazione.
- value: quantita' di valuta da trasferire al destinatario.
- state: campo indicante lo stato della transazione che puo' essere TR-IN-CORSO, TR-ANNULLATA e TR-COMPIUTA.

Di seguito verra' riportato un esempio di gestione di una transazione da parte dell'overlay network che copre tutte le casistiche. Ipotizziamo una transazione di importo 5 da parte del Peer 1 destinata al Peer 5.

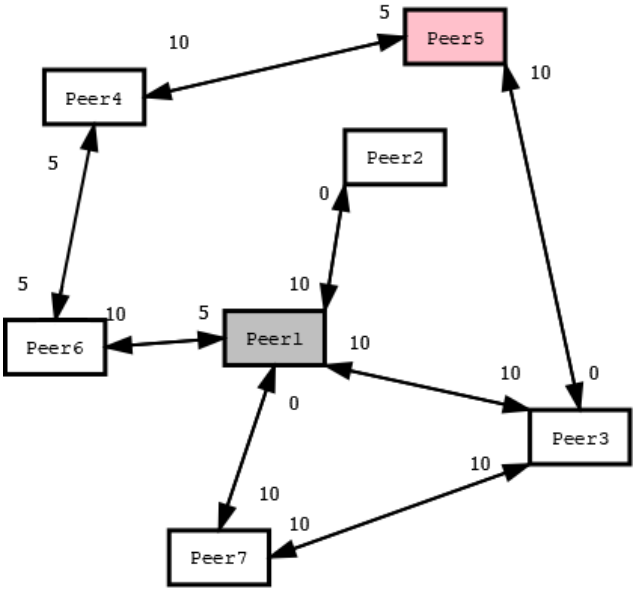

Figura 3.10: Passo 1

1. Si ipotizza che il peer 1 inizi l'esplorazione inviando la transazione prima sullo state channel aperto col peer 3.
2. Il peer 1 impegna per la transazione 5 unita' della propria valuta sullo state channel con peer 3. Il peer 3 ricevera' la transazione con stato TR-IN-CORSO e la gestira'.

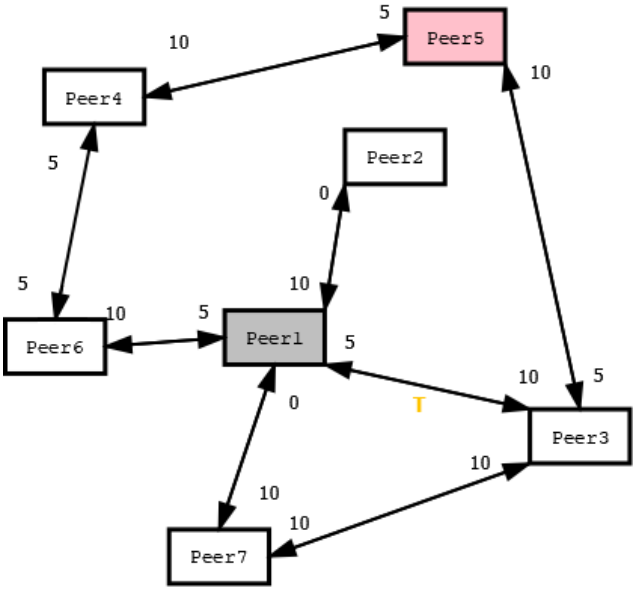

Figura 3.11: Passo 2

Si noti che l'effettivo accredito del valore della transazione ricevuta sul proprio state channel avviene solo a transazione compiuta. Ci si limita d'altro canto ad impegnare la quantita' di valuta necessaria sullo state channel solo quando si inoltra una transazione. L'impegno preventivo della quantita' di valuta impedisce che altre transazioni possano essere effettuate atraversando uno state channel che non dispone effettivamente di quantita' sufficiente per permettere la riuscita della suddetta.

3. In questo caso (figura 3.12) la transazione e' gestita dal peer 3. Realizza di poter raggiungere direttamente il peer 5, ma cio' non avviene in quanto non disponde di una quantita' di valuta sufficiente necessaria a trasferira la valuta della transazione. Per questo motivo la transazione e' inoltrata allo state channel aperto col peer 7.

4. Il valore impiegato dal peer 3 sullo state channel col peer 7 e' maggiore del valore della transazione, indi per cui e' possibile inviare la transazione (ora con stato TR-IN-CORSO) al peer 7.

5. Nella figura (figura 3.14) notiamo come il peer 7, che gestisce attualmente la transazione da noi considerata, trovi come "exit way" per arrivare al

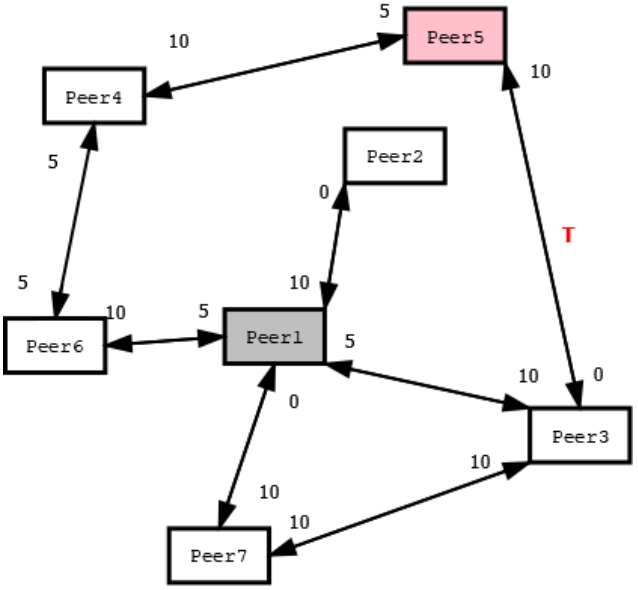

Figura 3.12: Passo 3

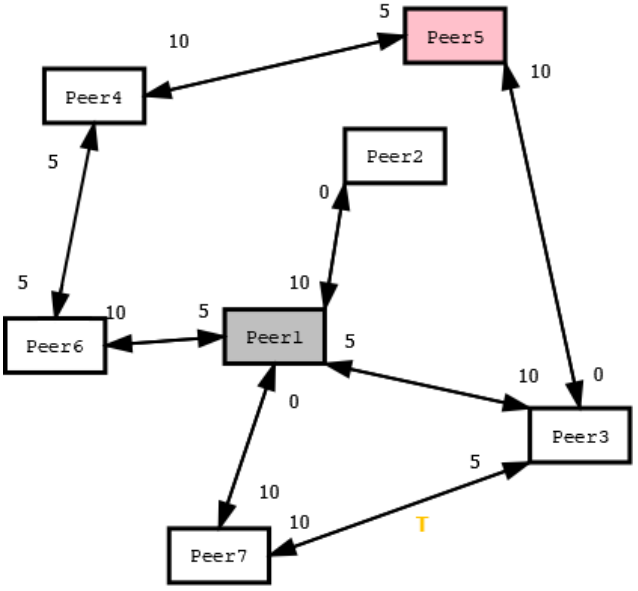

Figura 3.13: Passo 4

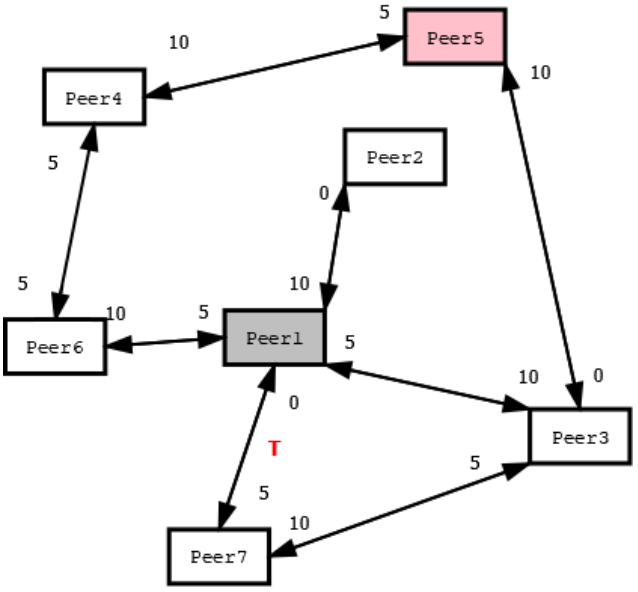

Figura 3.14: Passo 5

destinatario proprio peer 1, ovvero il mittente della transazione. In questo caso il peer 1, una volta ricevuta la transazione, si accorgera' che la sta gia' conducendo (controlla nella hash table "temporaryTransactions"). La transazione quindi tornera' al peer 7 con stato uguale a TR-ANNULLATA, ricorsivamente tornera' al peer 3 e poi al peer 1, liberando per ogni state channel interessato le risorse impegnate in caso di avvenuta transazione.

6. Il peer 1 una volta ricevuta la transazione come annullata, inoltra la transazione alla exit way identificata dal peer 6 (figura 3.15).

7. Il peer 5 riceve in gestione la transazione, e appura di essere il destinatario (figura 3.17). Setta lo stato della transazione a TR-COMPIUTA e gli viene accreditata la quantita' di valuta pari all'importo della transazione sullo state channel. Da questo momento in poi viene inoltrata al mittente seguendo il percorso a ritroso, e durante tale percorso vengono accreditati tutti i vari importi precedentemente impegnati.

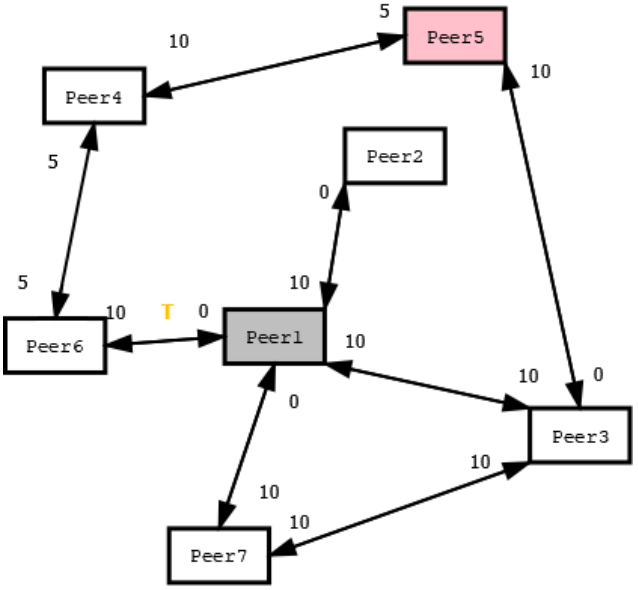

Figura 3.15: Passo 6

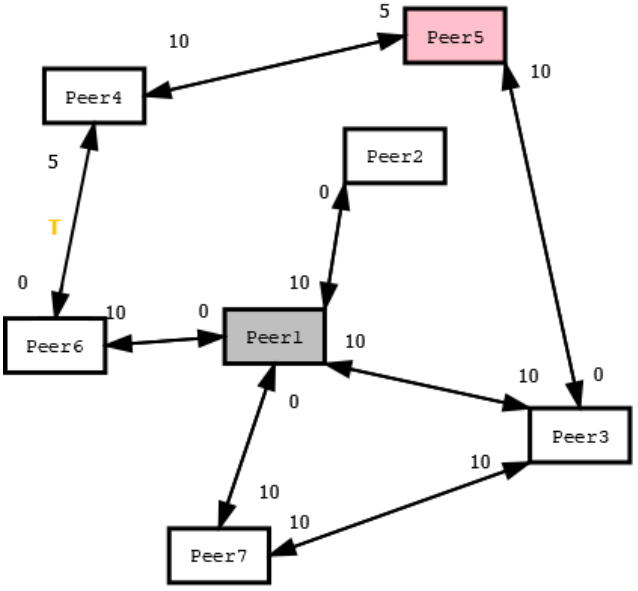

Figura 3.16: Passo 7

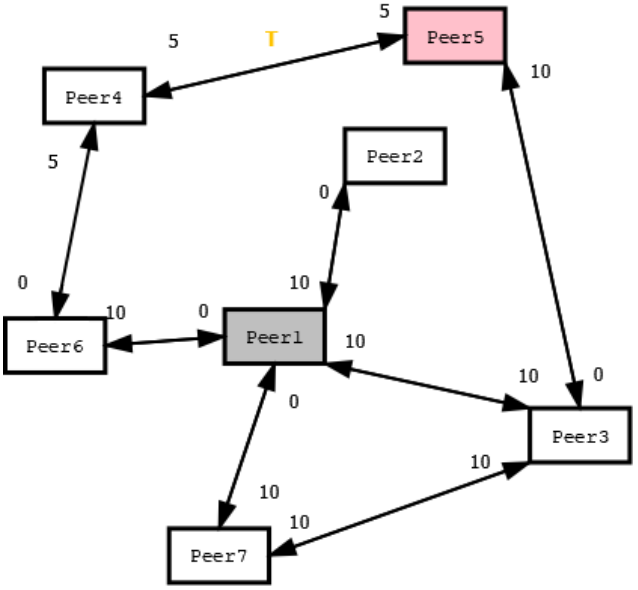

Figura 3.17: Passo 8

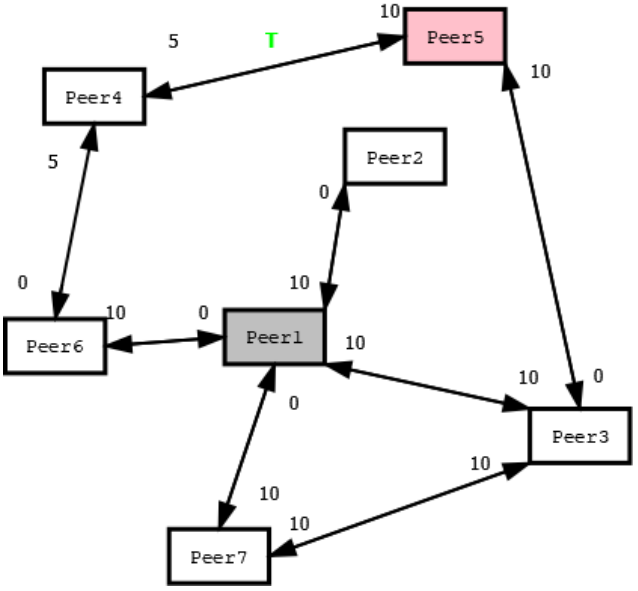

Figura 3.18: Passo 9

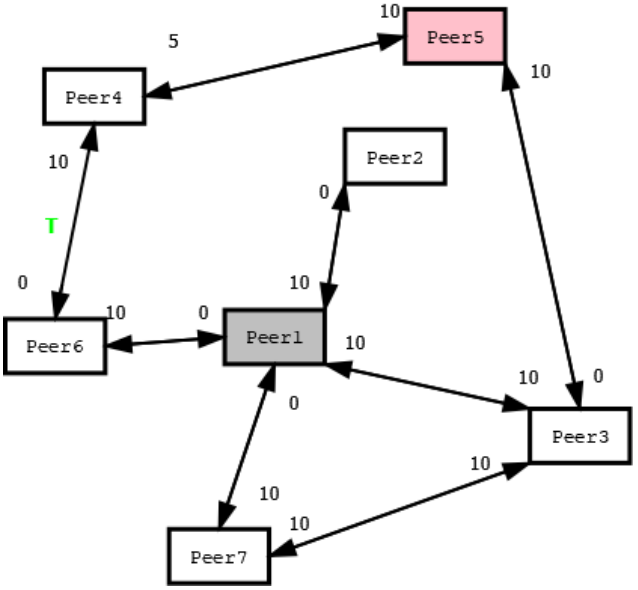

Figura 3.19: Passo 10

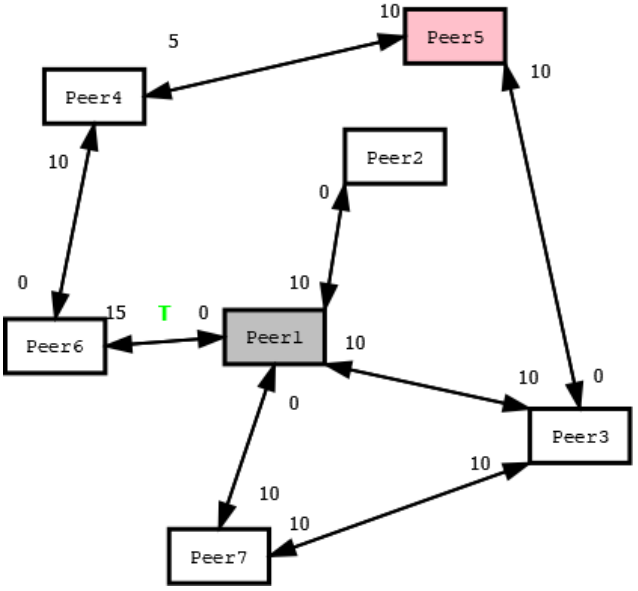

Figura 3.20: Passo 11

Di seguito un diagramma delle sequenze per aiutare la comprensione:

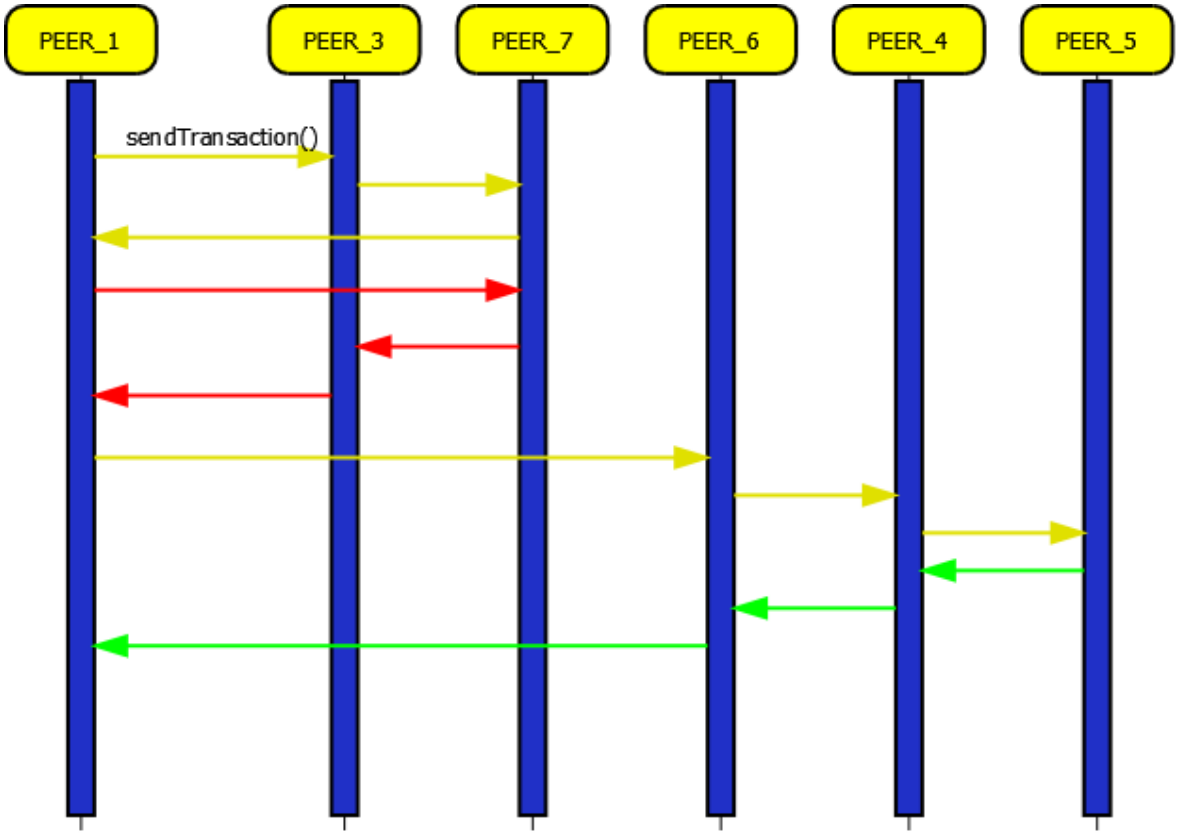

Figura 3.21:

Capitolo 4

Dettagli di implementazione

4.1 Server

Il server offre due diversi servizi:

- salva le specifiche di connessione di un peer all'interno di un file "info.txt";
- fornisce su richiesta le informazioni di connessione di un peer;

Esso si mettera' in ascolto su due porte diverse, e le diverse socket sono gestite tramite l'IO Multiplexing. Alla ricezione di una nuova connessione verra' forkato un nuovo processo per offrire il servizio richiesto.

4.2 Peer

All'avvio di un peer esso notifichera' la sua connessione al Server inviandogli le proprie specifiche di connessione. In seguito verranno creati diversi thread che offrono/chiedono i seguenti servizi:

- aggiornamento della tabella di routing;
- gestione delle transazioni;
- interazione uomo-macchina;

Sono state implementate due HashTable: una per i PeerToReach utilizzate per l'implementazione dell'algoritmo di routing, ed una per le transazioni temporanee che sono gestite dal peer.

```
typedef struct PeerToReach{
    int idPeerToReach;//ID DEL PEER DA RAGGIUNGERE, FUNZIONERA' DA CHIAVE PER LA HASH TABLE
    int* arrayExitWays;//ARRAY DINAMICO DEI PEER DA CUI USCIRE PER RAGGIUNGERE idPeerToReach
    int sizeArrayExitWays;
    struct PeerToReach* nextPeerToReach; //PUNTATORE A NEXT CHE PERMETTE DI CREARE UNA LISTA CONCATENATA
}PeerToReach;
```

Figura 4.1: Struct elemento della hash table dei peer che potenzialmente sono raggiungibili

```
typedef struct TransactionNode{
    Transaction transaction;
    int alreadyExploredExitWays[MAX_STATE_CHANNEL];
    struct TransactionNode *nextTransaction;
}TransactionNode;
```

Figura 4.2: Struct elemento della hash table delle transazioni temporanee

Viene utilizzata anche una lista monodirezionale per memorizzare gli state channels aperti. Il protocollo di trasporto utilizzato per la comunicazione tra peer e il server e' TCP, in quanto vi e' la necessita di una connessione affidabile e connection oriented. Le informazioni permanenti vengono memorizzate su fileSystem. Il bilancio e l'id dell'ultima transazione del peer corrente sono memorizzate nel file "initialize.txt", mentre le transazioni effettuate sono memorizzate in "transactions.txt". Se viene interrotto il processo, prima di arrestarsi, provvedera' a chiudere tutti gli state channel aperti ed a salvare il bilancio corrente e l'id dell' ultima transazione permanentemente (figura 4.3). Essendo le strutture dati riferite da piu' thread sono stati utilizzati metodi approssimativi di sincronizzazione per evitare race condition (mutua esclusione lettura e scrittura per ogni singola struttura dati).

```c
void signal_handler(int signal){

    printf("Processo stoppato, lo aborto\n");
    FILE * myFile;
    myFile=fopen("initialize.txt","w");

    while(myChannels!=NULL){
        printf("Chiudo il canale CON PEER %d\n", myChannels->info.idPeer);
        closeStateChannel(myChannels->info.idPeer);
    }

    fprintf(myFile,"bilancio:%d\nlastTransactionId:%d",myBalance,lastTransactionId);
    fclose(myFile);

    pthread_mutex_destroy(&mutexPeerICanReach);
    pthread_mutex_destroy(&mutexMyChannels);

    raise(SIGABRT);
}
```

Figura 4.3: Funzione attivata alla cattura dei segnali SIGINT e SIGTSTP.

Capitolo 5

Manuale utente

Per la compilazione del peer e del server e' stato fornito uno script file "start.sh" che provvedera' anche ad eseguire il server. Per poter eseguire un peer bisogna posizionarsi nella directory peerSource e lanciare il comando "./peer IDPEER IPSERVER CHANNELPORT UPDATEPORT TRANSACTIONPORT". L'IDPEER e' un intero maggiore di 0.