Progetto Reti Informatiche 2020/2021

Applicazione che implementa una rete peer-to-peer per la condivisione di dati relativi alla pandemia COVID-19. L'applicazione è formata da un discovery server (DS) che si occupa di mettere in comunicazione i diversi peer quando essi si vogliono connettere o disconnettere alla rete. Ogni peer poi ha la possibilità di inserire dati, richiederli agli altri peer ed effettuare delle elaborazioni su di essi.

CONTENUTO PROGETTO

ds.c

> Contiene il codice per eseguire il compito del discovery server. Attraverso un socket UDP accetta messaggi di connessione e disconnessione da parte dei peer mantenendo aggiornata la lista e richieste di blocco e sblocco della rete per mantenerla statica durante le operazioni di flood. Accetta inoltre semplici comandi da tastiera per visualizzare informazioni sulla rete o per disconnettersi. Inoltre mantiene aggiornati i peer sulla data da considerarsi corrente per l'inserimento dei dati.

peer.c

> Contiene il codice per permettere ai peer di connettersi e disconnettersi dalla rete, aggiungere dati, condividerli ed elaborarli. Attraverso un socket UDP comunica con il server e con gli altri peer per scambiare messaggi di coordinazione. Attraverso un socket listener TCP accetta richieste di dati sensibili, scambiandoli sempre attraverso connessioni TCP. Sono disponibili dei comandi da terminale per inserire dati e per richiederne elaborazioni.

util/

> Contiene i seguenti file di utilità (file.c) con i relativi header (file.h):
>
> - **date.c** file per la gestione e manipolazione di strutture 'Date'.
> - **msg.c** file per la gestione di socket udp e tcp e relativi invii e ricezioni di messaggi.
> - **neighbors.c** file che definisce la struttura Neighbors per la gestione dei vicini.
> - **peerlist.c** file per mantenere e modificare la struttura PeerList, lista di PeerElement, che rappresenta lo stato della rete.
> - **util_c.c** file che contiene funzioni generiche per il peer.
> - **util_s.c** file che contiene funzioni generiche per il discovery server.
> - **util.c** file che contiene funzioni generiche.
>
> Nei file.h sono presenti le definizioni delle costanti e commenti che descrivono le diverse funzioni.

makefile

> Compilazione e pulizia del progetto

exec.sh

> Compila il progetto attraverso il makefile e poi avvia il ds e 5 peer in locale.

data/

> Contiene il registri dei peer secondo uno schema ad albero

support/

> Contiene i seguenti file di documentazione:
>
> - **codes.txt** sono presenti i dettagli degli header usati per la comunicazione
> - **commands.txt** comandi di default per testare i peer
> - **ds.txt** descrizione in metacodice del comportamento del DS
> - **peer.txt** descrizione in metacodice del comportamento dei peer

README.md

> documentazione del progetto in formato markdown

documentazione.pdf

> documentazione del progetto in formato pdf (questo file)

UTILIZZO

> Per testare l'applicazione basta scaricare il progetto ed eseguire './exec.sh' che avvierà il DS e 5 peers in locale e inizializza dei dati in 'data/' (nuovi casi tra il 28/07/21 e il 03/08/21). Usare poi i comandi default nel file commands.txt per eseguire delle semplici operazioni.

TOPOLOGIA

> Lo stato della rete è mantenuto dal ds come una lista circolare in ordine di numero di porta in cui ogni peer ha un vicino prev e un vicino next. La rete ha la possibilità di essere bloccata attraverso una operazioni sulla variabile lock e mantiene il numero di peer connessi.

ALGORITMI E STRUTTURE DATI

Peerlist

> Sono principalmente comuni operazioni su una lista circolare ordinata di in cui il primo elemento non è necessariamente il più piccolo.

Messaggi

> Entrambi i protocolli tcp e udp sfruttano lo scambio di messaggi e acknowledgment. Si è preferito minimizzare la lunghezza dei messaggi andando così ad

> aumentarne la quantità per lo scambio di tante informazioni.

Dati

> Ogni peer conserva i dati su files nel formato data/port/type/aggr/date.txt dove
>
> > - data è la cartella predisposta a contenere i dati
> > - port rappresenta la porta del peer a cui appartengono i registri (es. 5001)
> > - type rappresenta il tipo di dati salvati (es. 't' / 'n')
> > - aggr rappresenta se si tratta di un registro (entries) o di somme (elabs)
> > - date nel formato yyyy_mm_dd
>
> Per un peer 'port', una data 'date' e un tipo 'type' il peer ha:
>
> > - tutti i dati e la somma di essi se esite data/port/type/entries/date.txt e data/port/type/elabs/date.txt
> > - solo parte dei dati se esiste solo data/port/type/entries/date.txt
> > - solo la somma di essi se esiste solo data/port/type/elabs/date.txt
> > - nessun dato altrimenti

Scambio di dati

> La prima volta che viene chiesta la somma dei dati relativi ad una data ed un tipo il peer *req_peer* fa girare un messaggio `flood-some`, ogni peer *prop_peer* che lo riceve e ha dei dati invia la propria disponibilità ad essere contattato attraverso una `prop-some`. A quel punto per ogni `prop-some` ricevuto il *req_peer* avvia una connessione tcp con ogni *prop_peer* e con una `send-all` richiede i dati posseduti che verranno inviati sequenzialmente tramite messaggi `new-entry`, che sommerà e salverà in un file elab. Da lì in poi un nuovo *new_req_peer* richiedente dati per gli stessi parametri non dovrà contattare nuovamente ogni peer ma facendo girare un messaggio `flood-all` avrà la risposta dal *req_peer* precedente che gli risponderà con una `prop-all` e interromperà la propagazione della `flood-all`. Sempre con la stessa procedura *req_peer* risponderà alla `send-all`, previa connessione, con tutte le entries da lui possedute. Prima di mandare le flood i peer chiedono ai vicini se hanno la somma e in quel caso si accontentano di non avere tutte le rispettive entries.

Disconnessione server

> Quando il DS si disconnette avvisa un peer *first* tramite un messaggio di `server-exit` e si mette in attesa che tutti i peer si disconnettano. *first* si assicura di avere tutte le somme dei giorni validi. Poi propaga la `server-exit` al vicino successivo e si mette a disposizione per inviare eventuali somme mancanti. Il successivo richiede i dati che non ha, avvisa il server di essersi disconnesso e propaga la `server-exit` mettendosi a sua volta a disposizione. Quando la notifica di `server-exit` torna a *first* esso per ultimo di disconnette, facendo capire al server che il messaggio ha fatto tutto il giro e si può disconnettere. Questo fa in modo che tutti i peer abbiano gli stessi dati dopo che la rete è stata chiusa.

VULNERABILITÀ

> La rete è completamente vulnerabile a qualunque peer malevolo che voglia falsificare i dati. Infatti il sistema si basa sulla fiducia in particolare dei vicini ma in generale di qualunque peer che sostenga di avere dei dati. Inoltre c'è la possibilità di bloccare l'accesso e la disconnessione dei peer con un semplice messaggio di lock ad DS.
>
> Il sistema del lock presenta delle vulnerabilità instrinseche. Potrebbe infatti bloccare lo stato della rete per lunghi periodi di tempo in caso di grande attività, ma allo stesso tempo permette di mandare più flood contemporaneamente che in casi di contemporateità potrebbero portare al fallimento delle stesse. Per valutare però queste problematiche con efficacia bisognerebbe testare l'applicazione in un contesto reale, nel quale essendo i dati già distribuiti questi problemi diventerebbero trascurabili.
>
> Inoltre la disconnessione del DS deve essere trattata come caso eccezionale oltre a non assicurare la validità dei dati relativi alla data nella quale avviene, l'incertezza su quali peer si riconnetteranno dopo un eventuale riaccensione della rete può portare al recupero di informazioni false riguarda a query su date precedenti allo spegnimento.