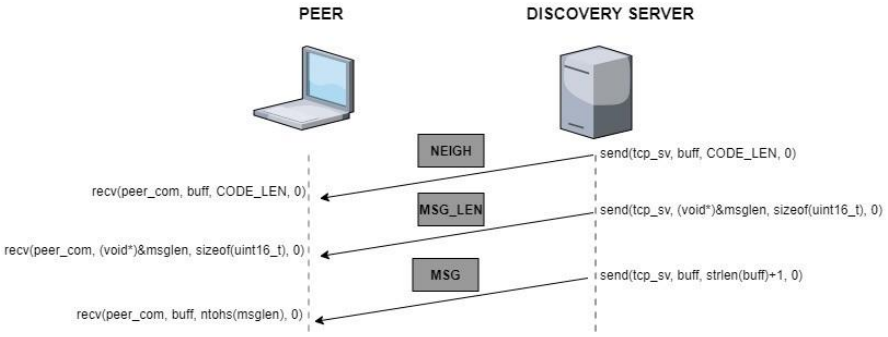
BIANCHI GIULIA, 578264

PROGETTO A.A. 2020/2021, RETI INFORMATICHE

L'applicazione implementata consiste in un software distribuito peer-to-peer, che costituisce un sistema per la condivisione di dati aggiornati sulla pandemia di COVID-19.



a) *Esempio di scambio di messaggi*

Per quanto riguarda le scelte relative al formato dei messaggi, l'opzione implementata è stata quella di inviare dei codici di 5 caratteri per notificare l'inizio di una sequenza di operazioni (ad esempio, il discovery server manda un messaggio con il codice 'NEIGH' ad un peer prima di iniziare ad inviare i suoi nuovi vicini in fase di connessione/disconnessione di un peer) o per notificare il verificarsi una condizione (una volta ricevuti i vicini, il peer notifica il server con un codice 'NGRCV'). Le informazioni successive sono poi scambiate su messaggi di lunghezza e formato variabile, pertanto l'invio/ricezione del messaggio è sempre preceduto dall'invio/ricezione della lunghezza di esso. Tale scelta è stata effettuata per evitare di creare traffico nella rete inviando campi non necessari, anche se ciò comporta comunque l'introduzione di un overhead temporale e spaziale a causa dello scambio della lunghezza dei messaggi. La dimensione dei messaggi è scambiata utilizzando il protocollo binary, mentre il messaggio è scambiato attraverso l'utilizzo del protocollo di tipo text.

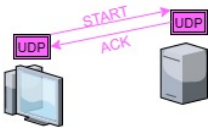


Figura b.1

Il socket UDP è stato utilizzato solo per la fase di boot, in particolare nello scambio dei messaggi di 'START' e 'ACK' tra un peer ed il discovery server, come mostrato nella figura b.1. In tutte le altre situazioni, viene utilizzato un socket TCP, in quanto offre un servizio di trasferimento affidabile dei dati. Ogni peer ha un socket TCP che 'si comporta da server', in ascolto di richieste di connessione. Quando viene ricevuta una richiesta, il peer la accetta e crea un socket TCP per la comunicazione, come è visibile dalla figura b.3. Nel caso sia invece necessario effettuare una richiesta, viene creato un nuovo socket TCP che 'si comporta da client', inviando la richiesta di connessione, che sarà poi chiuso insieme al socket di comunicazione dell'altro peer al termine della richiesta (figura b.3). Il socket utilizzato per la comunicazione con il server, diverso per ogni peer, è mantenuto finché il peer non decide di lasciare la rete o è forzato ad abbandonarla dal server con la *esc* (figure b.2 e b.3). Questa scelta restringe il numero di peer accettabili sulla rete del server per la limitatezza del numero di porte fisicamente esistenti, ma è plausibile immaginare che in un sistema reale possano esistere più server operativi.

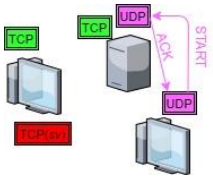


Figura b.2

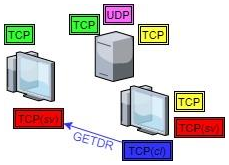
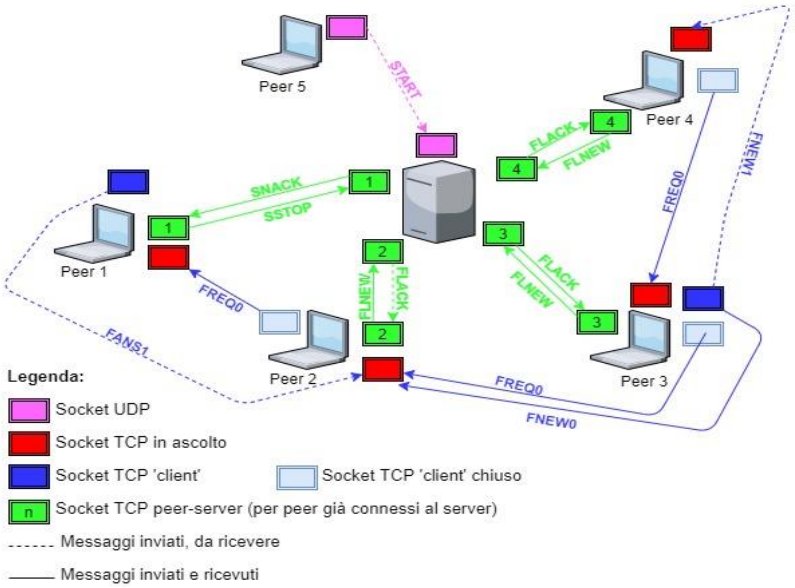


Figura b.3

Durante il flooding (per la propagazione di nuove entry del daily register o per l'ottenimento di un registro mancante, utile al calcolo di un dato aggregato) o durante lo scambio di informazioni tra peer, il server non ha la possibilità di accettare nuovi peer o rimuoverli, per evitare che vi siano inconsistenze nella topologia della rete e nello scambio dei messaggi relativi all'aggiornamento dei vicini, considerando soprattutto situazioni di traffico elevato sulla rete, come è osservabile dall'immagine. Questa scelta presenta comunque uno svantaggio: nel caso venissero effettuate continue richieste di flooding, queste



c) Esempio con flooding nella rete, non sono accettate richieste di START o SSTOP

verrebbero accettate a discapito delle richieste di connessione o rimozione di un peer dalla rete del discovery server, introducendo grosse attese.

I peer sono organizzati seguendo l'approccio della lista circolare, molto pratica per l'individuazione di un criterio di implementazione e terminazione del flooding: il peer che da inizio al flooding invia un messaggio al vicino di 'destra' ed al vicino di 'sinistra', che inoltrano rispettivamente il messaggio al vicino opportuno. Se non ci sono vicini a cui inoltrare il messaggio o l'identificatore del vicino successivo è non conforme al tipo di flooding che si sta effettuando, il flooding termina da un lato (altrimenti verrebbe mandata una doppia richiesta al solito peer). Se un peer avesse un numero maggiore di vicini e inviasse più di due richieste di flooding, potremmo avere più di due peer che inoltrano contemporaneamente un messaggio e chiaramente la sua propagazione in tutta la rete avverrebbe in un tempo minore rispetto a quello richiesto dalla soluzione proposta, la quale risulta essere meno scalabile, introducendo ritardi maggiori per il completamento del flooding all'aumentare dei peer connessi.

Figura d

struct daily_register
int day
int month
int year
int tot_tamponi
int tot_casi
int closed
struct daily_register* next_register

e) Struttura dati

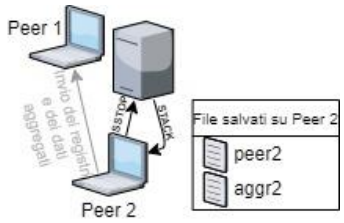
I registri giornalieri sono salvati su strutture dati di tipo *daily_register*, organizzati in una lista *registers*; i dati aggregati sono salvati su strutture dati di tipo *data_aggr*, organizzati in una lista *aggr*.

struct data_aggr
int day1
int month1
int year1
int day2
int month2
int year2
char aggrt
char type
int data
struct data_aggr* next_aggr

f) Struttura dati 'data_aggr'

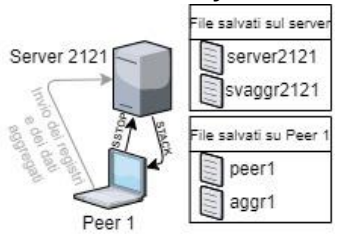
Nel registro giornaliero, sono salvati giorno, mese e anno del registro, il totale dei tamponi e dei casi rilevati in quel giorno, un intero che indica se il registro è stato chiuso ed il puntatore al registro successivo. Ciascun dato aggregato è caratterizzato dal periodo in cui esso è calcolato (day1/month1/year1-day2/month2/year2), il tipo di dato aggregato *aggrt* ('V', 'T'), il tipo di dati *type* su cui si calcola il dato ('T', 'N'), il valore *data* e un puntatore al prossimo dato aggregato.

Un peer possiede tutte le informazioni necessarie al calcolo di un dato aggregato dal giorno day1/month1/year1 al giorno day2/month2/year2 se ha a disposizione ogni registro compreso in quel periodo e ciascun registro è stato chiuso. Se il registro non fosse chiuso, ad esempio, un peer potrebbe essersi disconnesso prima delle 18:00 e gli altri peer potrebbero aver aggiornato il registro giornaliero, pertanto il totale dei tamponi e/o dei casi rischierebbe di essere minore rispetto a quello effettivo. Per ottenere il registro più aggiornato, viene quindi effettuata una richiesta in flooding: se qualche peer possiede il registro chiuso, si riceve quello, altrimenti la richiesta è inoltrata a tutti, per vedere se qualcuno ha dei dati più aggiornati. In mancanza del registro si opera come nel caso della ricerca di un registro più aggiornato, usando dati iniziali differenti (*tot_tamponi*=0, *tot_casi*=0).

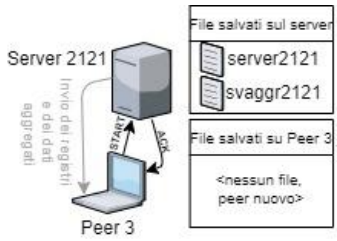


g.1) Peer 2 se ne va, resta ancora un peer

Quando un peer si disconnette dalla rete di un discovery server, i suoi registri e dati aggregati sono salvati su file distinti. Questi file sono aperti e caricati in memoria quando si riconnette alla rete del discovery server. Sempre in fase di disconnessione, i registri e i dati aggregati del peer sono inviati ai vicini, se esistono. Nel caso che il peer che sta abbandonando la rete fosse l'ultimo peer connesso, esso sarà in possesso di tutti i registri più aggiornati ottenuti dall'avvio del discovery server, pertanto per non perdere le informazioni memorizzate fino a quel momento, i registri e i dati aggregati saranno inviati al server, che li salverà in dei file e li invierà al prossimo peer che si connetterà alla rete (il quale sarà naturalmente privo di vicini). Quando il server riceverà o invierà i registri al nuovo e momentaneamente unico peer, verrà effettuata la chiusura sicura dei registri giornalieri settando la variabile *closed*, assicurandosi che la data del registro sia precedente alla data odierna o, se dovesse coincidere con la data corrente, verificando che l'ora attuale superi le 18:00.



g.2) Peer1 se ne va, è l'ultimo perciò invia registri e dati al server



g.3) Arrivo di un peer nuovo, ottiene registri e dati da server