UNIVERSITÀ DI PISA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA
SPECIFICHE PROGETTO A.A. 2020/2021

# Reti informatiche, cod. 545II, 9 CFU

Prof. Giuseppe Anastasi, Ing. Francesco Pistolesi

Si richiede di progettare un'applicazione distribuita peer-to-peer che implementi un sistema per la condivisione di dati costantemente aggiornati sulla pandemia di COVID-19.

## 1. VISIONE D'INSIEME

Considerare uno scenario in cui vari peer cooperano per elaborare statistiche aggiornate sulla pandemia. Gli utenti dei peer inseriscono costantemente dati sui nuovi casi, sui tamponi effettuati, sugli esiti di questi tamponi e così via. Ogni peer è responsabile di raccogliere nuovi dati e di elaborarli, condividendo poi con gli altri peer sia i dati che le elaborazioni effettuate. L'architettura peer-to-peer da utilizzare è mostrata in Fig. 1. Oltre ai peer, il network include un server centrale, detto *discovery server (DS)*. Il DS si occupa di registrare i peer e mantenere un registro dei neighbor di ogni peer.

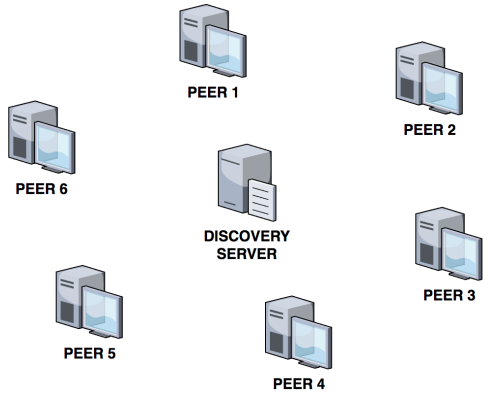


Figura 1: Architettura della rete con peer e discovery server (DS)

Per effettuare il boot, ogni peer contatta il DS tramite un boot message UDP, inviando IP e porta[1]. Ricevuto un boot message, il DS aggiungerà il peer che lo ha inviato in un registro interno, e comunicherà al peer una lista di neighbor (i loro numeri di porta) che saranno direttamente contattabili dal nuovo peer. Se il nuovo peer non riceve risposta dal DS entro un dato timeout[2], trasmette di nuovo il boot message fintantoché non

[1] Assumendo di avviare tutti i peer su localhost, l'IP lo stesso per tutti. Supporre che l'ubicazione di un peer sia espressa dal numero di porta associata al peer.
[2] Il timeout può essere impostato a piacere.

riceve risposta dal DS. Se il nuovo peer è il primo peer del network, il DS lo registra e gli invia una lista vuota di neighbor. Il DS avrà il compito di integrare questo peer nel network a mano a mano che nuovi peer effettueranno il boot. La politica di integrazione è una scelta progettuale.

Una volta effettuato il boot, un peer può:

- ricevere comandi da standard input;
- ricevere messaggi da altri peer.

Un peer può ricevere comandi da standard input per aggiungere nuove informazioni (*entry*) nel registro giornaliero (*register*) del peer, mantenuto in memoria o su file.
Una entry ha quattro field: una *date*, un *type*, e una *quantity*. Il type può essere *tampone*, *nuovo caso*. La quantity specifica il numero di persone a cui la entry si riferisce. Per esempio, se oggi 1° Dicembre 2020 si registrano in un peer tre contagi, potrebbe essere inserita nel register la entry '`2020:12:01,N,+,3`', dove 'N' indica nuovo caso, e '3' è il numero di persone che hanno fatto il tampone. Il formato delle entry è una scelta di progettazione sia che siano salvate su file sia che si mantengano in memoria.

I register dei peer vengono chiusi alle ore 18 di ogni giorno, o su richiesta del DS. Dopo la chiusura, i register della data corrente non possono più essere modificati e iniziano a essere modificabili quelli del giorno successivo. La Fig. 2 mostra un peer e un register, con i field di una entry.

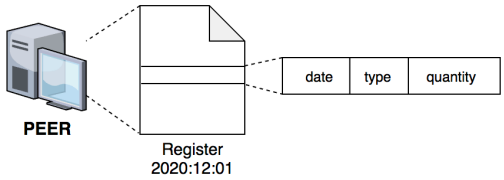


Figura 2: Register di un peer per il giorno 1° Dicembre 2020, e field di una entry

Oltre all'inserimento di nuove entry, i peer accettano comandi che esprimono *richieste di elaborazione*, cioè il calcolo di una aggregazione dei dati. Una richiesta di elaborazione è valida solo se riguarda register chiusi. Le richieste non valide causano la visualizzazione di un errore, e sono scartate.
Quando riceve una richiesta di elaborazione, un peer (*requester*) verifica se ha già calcolato il dato richiesto[3]. Se è così, lo restituisce, altrimenti il requester deve verificare se ha tutte le entry per calcolare il dato aggregato richiesto. Se le ha, effettua il calcolo, restituisce il risultato e lo memorizza. Se il requester non ha tutte le entry, contatta i suoi neighbor tramite un messaggio REQ_DATA. Se un neighbor ha il dato già calcolato, lo invia al requester con un messaggio REPLY_DATA, il quale a sua volta lo memorizza e lo mostra. Se i neighbor non hanno il dato calcolato inviano un messaggio REPLY_DATA privo di entry. Il requester effettua allora un flooding per richiedere le entry mancanti, contattando di nuovo i neighbor tramite un messaggio FLOOD_FOR_ENTRIES. I neighbor contattano a loro volta i loro neighbor, e così via. Se un peer ha alcune delle entry che mancano al requester, lo comunica all'indietro al neighbor richiedente il quale propagherà l'informazione all'indietro fino al requester. Il requester contatterà quindi i peer che hanno le entry che gli sono necessarie, richiedendo loro di inviargliele tramite un messaggio REQ_ENTRIES. Una volta ottenute tutte le entry, il requester effettua il calcolo, lo memorizza e lo mostra.

La politica con la quale un peer riesce ad essere consapevole di avere (o meno) tutte le entry necessarie a soddisfare una richiesta di elaborazione (o quali entry gli manchino) è una scelta progettuale, così come lo è

[3] Questo può accadere perché, in generale, una richiesta di elaborazione può essere fatta più volte sullo stesso peer.

il protocollo usato da un peer per richiedere le entry mancanti agli altri peer, e il formato dei messaggi scambiati.

## 2. DETTAGLI IMPLEMENTATIVI

L'applicazione distribuita da sviluppare deve implementare quanto descritto nel Paragrafo 1. Le scelte progettuali devono essere spiegate in una **relazione sintetica di non più di 2 pagine**, font Arial, dimensione 12 pt, prediligendo l'uso di figure e schemi intuitivi (anche tracciati a mano), limitando il testo. Nella relazione devono essere brevemente messi in luce potenziali pregi e difetti delle scelte fatte in termini di quantità di traffico generato, possibili bottleneck, scalabilità, e così via.

### 2.1 PEER

Un **peer** è mandato in esecuzione come segue:

```
./peer <porta>
```

dove **<porta>** è la porta associata al peer.

Appena mandato in esecuzione, il processo **peer** mostra a video una guida dei comandi. Dopo l'inserimento di un comando da standard input, oppure dopo la ricezione di un messaggio dalla rete da parte di altri peer (o DS), il peer mostra a video, passo passo, cosa sta facendo, con un formato a piacere[4].

I comandi accettati dal peer sono:

**start** ***DS_addr DS_port***
richiede al DS, in ascolto all'indirizzo *DS_addr*:*DS_port*, la connessione al network. Il DS registra il peer e gli invia la lista di neighbor. Se il peer è il primo a connettersi, la lista sarà vuota. Il DS avrà cura di integrare il peer nel network a mano a mano che altri peer invocheranno la start.

**add** ***type quantity***
aggiunge al register della data corrente l'evento *type* con quantità *quantity*. Questo comando provoca la memorizzazione di una nuova entry nel peer, come quella mostrata in Fig. 2. Il formato delle entry è una scelta di progettazione.

**get** ***aggr type period***
effettua una richiesta di elaborazione per ottenere il dato aggregato *aggr* sui dati relativi a un lasso temporale d'interesse *period* sulle entry di tipo *type*. Considerando tali dati su scala giornaliera. Le aggregazioni *aggr* calcolabili sono il *totale* e la *variazione*. Il parametro *period* è espresso come 'dd1:mm1:yyyy1-dd2:mm2:yyyy2', dove con 1 e 2 si indicano, rispettivamente, la data più remota e più recente del lasso temporale d'interesse. Una delle due date può valere '*'. Se è la prima, non esiste una data che fa da lower bound. Viceversa, non c'è l'upper bound e il calcolo va fatto fino alla data più recente (con register chiusi). Il parametro *period* può mancare, in quel caso il calcolo si fa su tutti i dati.

Considerati $n$ giorni consecuivi (il periodo) e indicata con $q(e_i)$ la quantity della entry $e_i$ relativa al giorno $i$ (per esempio, $e_i$ potrebbe essere il numero di tamponi del giorno $i$), le due aggregazioni si calcolano come segue:

[4] Si può pensare che il peer lavori in modalità *verbose*, e informi l'utente continuativamente su ciò che sta facendo.

**Totale:**
Consideriamo ad esempio il numero di tamponi. Se $q(e_i)$ è la quantità di tamponi effettuati nel giorno *i*, il totale dei tamponi su *n* giorni è semplicemente:

$$\sum_{i=1}^{n} q(e_i)$$

**Variazione:**
Considerando ancora il numero di tamponi come type, si calcola la variazione è la differenza fra i tamponi di ieri e quelli di oggi.

$$q(e_i) - q(e_{i-1})$$

Se il lasso temporale espresso con l'argomento *period* include *n* giorni, il comando restituisce *n* – 1 differenze, una per ciascun giorno del periodo.

**stop**
il peer richiede una disconnessione dal network. Il comando stop provoca l'invio ai neighbor di tutte le entry registrate dall'avvio. Il peer contatta poi il DS per comunicare la volontà di disconnettersi. Il DS aggiorna i registri di prossimità rimuovendo il peer. Se la rimozione del peer causa l'isolamento di una un'intera parte del network, il DS modifica i registri di prossimità di alcuni peer, affinché questo non avvenga.

## 2.2 DISCOVERY SERVER

Il discovery server (DS) si avvia col seguente comando:

```
./ds <porta>
```

dove **<porta>** è la porta associata.

All'avvio, i comandi disponibili sono:

- **help**
- **showpeers**
- **showneighbor <peer>**
- **esc**

Quando su un peer è invocato il comando start, il DS è contattato tramite un messaggio UDP e registra il peer memorizzandone IP e porta. Notifica poi l'avvenuta registrazione tramite un ack, e aggiorna il registro di prossimità dei peer facendo in modo che ogni peer abbia due neighbor. Implementare un meccanismo di prossimità dei peer basandosi sul numero di porta. Dato che i peer, per semplicità, saranno tutti sull'host locale, i due peer più vicini (neighbor peer) a un generico peer in ascolto sulla porta *x* sono i peer con i numeri di porta più vicini a *x*. Ogni peer ha solo due neighbor peer.

Un esempio di esecuzione è il seguente:

```
$ ./ds 4242
```

il cui output potrebbe essere simila a ciò che segue:

```
**************************** DS COVID STARTED ****************************
Digita un comando:

1) help --> mostra i dettagli dei comandi
2) status --> mostra un elenco dei peer connessi
3) showneighbor <peer> --> mostra i neighbor di un peer
4) esc --> chiude il DSDettaglio comandi

_
```

I comandi accettati da tastiera dal DS sono i seguenti:

**`showpeers`**
mostra i dettagli sui comandi accettati

**`status`**
mostra l'elenco dei peer connessi al network (IP e porta)

**`showneighbor [peer]`**
mostra i neighbor di un peer specificato come parametro opzionale. Se non c'è il parametro, il comando mostra i neighbor di ogni peer

**`esc`**
termina il DS. La terminazione del DS causa la terminazione di tutti i peer. Opzionalmente, prima di chiudersi, i peer possono salvare le loro informazioni su un file, ricaricato nel momento in cui un peer torna a far parte del network (esegue il boot).

Requisiti

- I dati sono scambiati tramite **socket**. Se non si definisce un formato di messaggio, prima di ogni scambio, il ricevente deve essere informato su **quanti byte** leggere dal socket.

- Per gestire le varie informazioni, è possibile usare **strutture dati a piacere**. Gli aspetti che non sono dettagliatamente specificati in questo documento possono essere implementati liberamente.

- Usare gli **autotools** (comando **make**) per la compilazione del progetto.

- Il codice **deve essere indentato e commentato** in ogni sua parte: significato delle variabili, processazioni, ecc. I commenti possono essere evitati nelle parti banali del codice.

- Va prodotta una **documentazione di 2 pagine** (font Arial, size 12 pt) in cui spiegare, anche con l'aiuto di figure o grafici, le scelte fatte relativamente alle parti lasciate a scelta dello studente, e i formati di strutture dati e messaggi di rete (campi, numero di byte, ecc.).

- Scrivere uno **script che compili il progetto, mandi in esecuzione il DS e faccia il boot di 5 peer**, su finestre diverse del terminale. Fare in modo che in ogni peer ci siano alcuni register già compilati (ovviamente con valori casuali) in modo da poter testare le varie funzionalità.

## CONSEGNA

Il progetto deve essere caricato sul sistema elearn.ing.unipi.it (sulla pagina del corso) non oltre le 72 ore che precedono il giorno dell'esame, usando le credenziali di Ateneo per l'accesso. Per ogni appello, sarà creata una nuova sezione per le consegne.

## VALUTAZIONE

Il progetto è visionato e valutato prima del giorno dell'esame. Prima di effettuare la consegna, **testare il codice su una macchina Debian 8**. Durante l'esame, sarà richiesta l'esecuzione del programma e saranno fatte domande sia sul codice che sulle scelte fatte.

La valutazione del progetto prevede le seguenti fasi:

1. **Compilazione del codice**
   Il client e il server vanno compilati con opzione **`-Wall`** che mostra i vari warning. Non vi dovranno essere warning o errori. L'opzione **`-Wall`** va abilitata anche durante lo sviluppo del progetto, interpretando i messaggi forniti dal compilatore. Se il progetto non compila, non è valutabile;

2. **Esecuzione dell'applicazione**
   In questa fase si verifica il funzionamento dell'applicazione e il rispetto delle specifiche;

3. **Analisi del codice sorgente**
   Può essere richiesto di spiegare parti del codice e apportarvi semplici modifiche.

## FUNZIONI DI UTILITÀ

Una documentazione di alto livello per le funzioni necessarie per leggere e scrivere file a blocchi possono essere visualizzate al seguente indirizzo:

https://www.programiz.com/c-programming/c-file-input-output