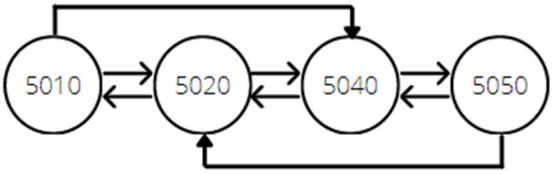
Fabiano Pilia 545397

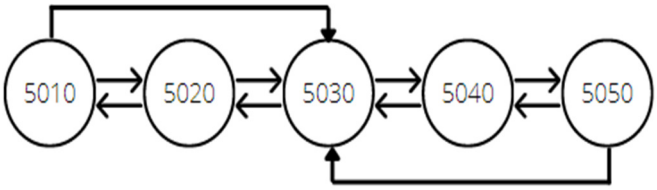
Documentazione relativa al progetto di reti informatiche aa 2021/2021

I peer sono organizzati come una lista ordinata sul numero di porta, la cui struttura dati, indicata di seguito, è salvata nel ds. Il neighbor1 è quello alla sinistra, il neighbor2 è quello alla destra. Nel caso degli estremi, essi hanno come neighbor il primo peer con numero di porta più vicino al proprio, escluso l'altro neighbor.

pre inserimento 5030:

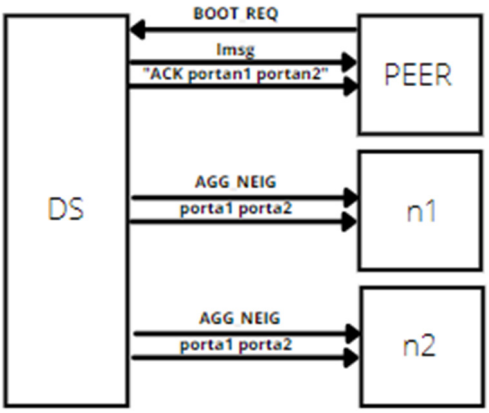


post inserimento 5030:




```c
//Struttura che descrive un peer connesso
struct peer{
    char* ip;
    int porta;
    //variabili che tengono la porta dei neighbors
    int neighbor1;
    int neighbor2;
    //primo nella coda di descrittori di peer
    int primo;
    //ultimo nella coda di descrittori di peer
    int ultimo;
    //variabile per sapere se tale descrittore ha avuto i i neighbor modificati
    //ed aggiornare tramite un messaggio i neighbor dei peer
    int modificato;
    //puntatore al prossimo descrittore in coda
    struct peer* next;
};
```

Nell'immagine a lato è rappresentato ciò che succede all'atto di una richiesta di boot da parte di un peer. Dall'alto verso il basso sono presentati in ordine cronologico i messaggi scambiati in formato testuale tra il ds ed i peer coinvolti nell'integrazione del peer "PEER" nel network. Il ds è conscio di quali peer hanno subito una modifica dei propri neighbor consultando il campo "modificato" delle relative strutture dati. Di conseguenza invia a tali peer i valori delle porte dei nuovi neighbor in un unico messaggio.

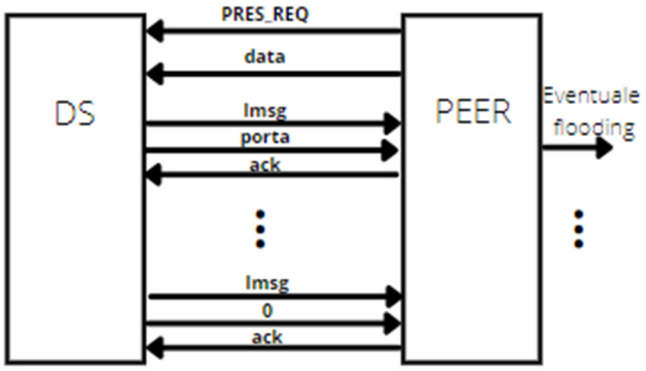


Le immagini di fianco rappresentano il formato di una entry e la relativa struttura dati presente nel peer.


```c
//STRUTTURA DATI CHE RAPPRESENTA UNA ENTRY
struct entry{

    //DATA NELLA QUALE È STATA AGGIUNTA LA ENTRY
    char date[12];

    //TIPO DELLA ENTRY:  t = nuovo tampone;  p = nuovo caso;
    char type[4];

    //NUMERO DI PORTA IN CUI È STATA REGISTRATA LA ENTRY,
    // SERVE PER EVITARE DOPPIONI NEL REG GENERALE
    int porta;

    //NUMERO DI DATI ASSOCIATI ALLA ENTRY
    int quantity;

    //PUNTATORE PER REALIZZARE LE LISTE DEI REGISTRI
    struct entry* next;
};
```

Le entry sono salvate in memoria in due liste:
-RegGiornaliero: raccoglie le entry registrate nella giornata corrente. Alla chiusura le entry vengono compattate in una entry relativa ai tamponi ed una ai nuovi casi e spostate nel seguente registro;
-RegGenerale: raccoglie tutte le entry registrate nei giorni passati sul peer e le entry procurate tramite il query flooding per il calcolo dei dati aggregati.
Alla disconnessione il registro generale è salvato su un file "numeroPorta.txt", il quale è letto per inizializzare tale registro in fase di boot.

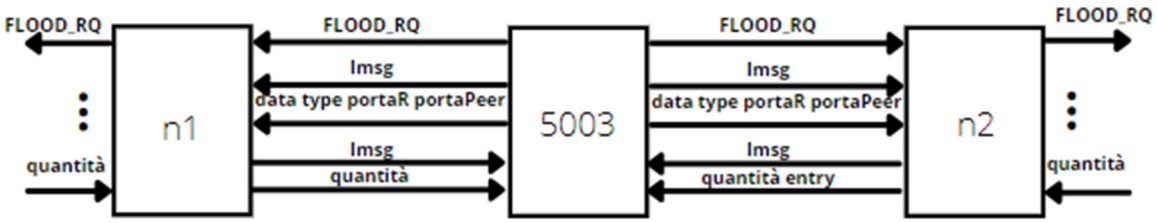
Un peer per sapere se ha le entry necessarie per calcolare un'aggregazione contatta il ds per sapere le presenze registrate in una data. Ciclicamente il ds invia tali presenze al peer, il quale controlla se ha la entry del peer, la cui porta è stata inviata dal ds; in caso affermativo invia un ack, altrimenti effettua un flooding per reperire la entry.
Quando un peer riceve come porta "0" le presenze di quella data sono terminate.
Nel caso di lower bound non specificato ("*") il peer capisce di essere arrivato alla data meno recente in assoluto quando riceve come lmsg in risposta ad una PRES_REQ il valore 0.




```c
struct presenze{
    //Data del giorno considerato
    char data[12];

    //Elenco presenti
    struct id* listaPresenze;

    //Prossimo elemento del registro
    struct presenze* next;
};
```

```c
struct id{

    //porta presente
    int porta;

    //puntatore per costruire
    // una lista di presenze
    struct id* next;
};
```

La seguente immagine presenta i messaggi e il modo in cui avviene il flooding.
Viene contattato prima il ramo di sinistra (n1), se l'esito è negativo (quantità = -1) viene contattato il ramo di destra (n2). La ricezione di -1 da entrambi i rami non blocca il calcolo dell'aggregazione, ad esempio quando sono state perse delle entry.



La disconnessione di un peer avviene tramite un messaggio DISC_REQ al ds, viene eliminato il peer in modo ordinato dalla lista e vengono contattati dal ds i peer con "modificato = 1" per l'aggiornamento dei neighbors, che avviene come già descritto in precedenza.

La disconnessione del ds è comunicata a tutti i peer connessi tramite un messaggio DS_EXIT, i peer rispondono con un "ACK" e terminano la propria esecuzione.

Il meccanismo della richiesta delle presenze al ds rende omogeneo il modo di ottenere le entry mancanti, piuttosto che tenere tali informazioni su un peer e doverle spostare alla sua disconnessione.
Uno svantaggio di tale meccanismo sta nel fatto che debba essere presa in considerazione una entry alla volta ed effettuare un eventuale flooding non su un insieme di entry, ma al massimo una alla volta. Inoltre il server potrebbe rallentare alla ricezione di tante richieste di vario tipo da più peer, rallentando anche gli altri servizi.
Ciò però favorisce i calcoli futuri se come primi calcoli vengono effettuati quelli "pesanti", come "*-*", in quanto si possederebbero già tutte le entry.
La struttura a lista con invio in modo sequenziale di richieste ai vicini in caso di flooding favorisce la velocità di reperimento delle entry se viene scelto il ramo corretto come primo; in caso contrario si effettuerebbe una catena di richieste inutili ed eventualmente costose.