RETI DI CALCOLATORI – Quarto appello a.a. 2008/09

E1 (6 punti). Descrivere con un automa a stati finiti il comportamento di un semplice servizio che, ricevuto il nome X di un server, restituisce il nome canonico di X oppure il messaggio "unresolved". Il servizio deve ricevere le richieste dei suoi clienti sulla porta TCP 9999 e deve cercare di risolverle interagendo con UDP con il suo **localNameServer**. Per la descrizione dell'automa utilizzare gli eventi:

c=accept(9999) // per indicare l'accettazione di una richiesta di connessione
s=receive(c) // per indicare la ricezione di un messaggio su una connessione
<name,value,type> = UDP_rcv(porta) // per indicare la ricezione di una risposta DNS
timeout() // per indicare lo scadere del timeout

e le azioni:

send(c,s) // per spedire di un messaggio su una connessione
close(c) // per chiudere una connessione
UDP_send(server,porta,<QNAME,QTYPE>) // per inviare una richiesta DNS
startTimer() // per (ri)avviare il timer

Per semplicità assumere che ogni risposta DNS consista di una sola tupla, che le tuple di risposta di tipo "NS" contengano nel campo **value** direttamente l'indirizzo IP (anziché il nome) del server di competenza e che il servizio rispedisca non più di una volta una stessa query allo stesso server.

E2 (7 punti). Descrivere con un automa a stati finiti (il lato mittente di) un'estensione del protocollo per il trasferimento affidabile di dati RDT3.0 (alternating bit protocol) in grado di garantire sia la riservatezza che l'integrità dei dati trasmessi. Assumere per semplicità che il livello sottostante non corrompa i dati trasmessi e per descrivere l'automa (supponendo che i problemi di autenticazione siano già stati risolti) utilizzare gli eventi:

rdt_send(d) // per indicare la ricezione di una richiesta di spedire i dati
dpck = udt_rev () // per indicare la ricezione di un pacchetto dal livello sottostante
timeout() // per indicare lo scadere del timeout

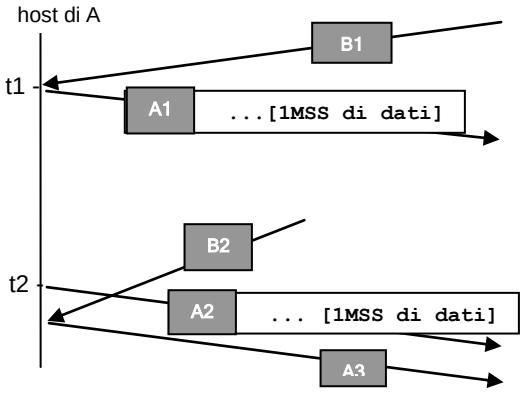
e le azioni:
pkt = make_pkt(...) // per indicare la costruzione di un pacchetto
udt_send(pkt) // per indicare l'invio di un pacchetto sul livello sottostante
stop_timer() // per fermare il timer

Commentare le altre funzioni eventualmente utilizziate ed esplicitare eventuali ipotesi su cui si basa l'estensione proposta.

E3 (7 punti). Consideriamo un'applicazione A che ha già stabilito una connessione TCP con un suo pari. Supponiamo che al tempo t_0 il valore della finestra di congestione **CongWin** dell'host di A sia pari a *4MSS*, gli indici **sendBase** e **nextSeqNum** valgano rispettivamente *X* e *X+2MSS*, che, sempre al tempo t_0, l'host di A debba spedire 2 MSS di (nuovi) dati e che (solo) al tempo t_2 scatti un timeout. Specificare, motivando la risposta:

- il valore dei campi **ACK** e **RcvWin** dei segmenti B1 e B2;
- il valore dei campi **SEQ** dei segmenti A1, A2 e A3, e
- la quantità di dati contenuti nel segmento A3

nei vari scenari possibili. Supporre che tutti i segmenti contenenti dati spediti dall'host di A prima di t_0 contenessero 1MSS di dati.



E4 (7 punti)**.** Scrivere il codice (non lo pseudo-codice) dell'algoritmo di Dijkstra *link state* assumendo che i nodi della rete siano identificati dai primi N-1 naturali e che il nodo origine sia 0 e utilizzare:

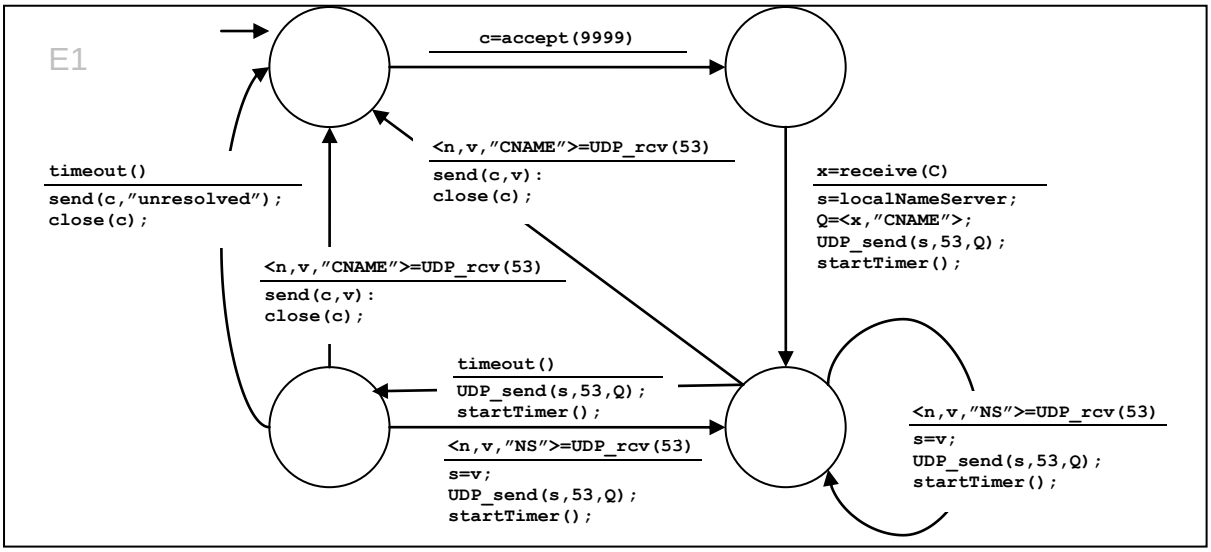
- un vettore **coloured** di N interi per rappresentare l'insieme dei nodi per i quali è stato determinato il costo del cammino minimo da 0 ad essi;
- un vettore **C** di NxN interi per rappresentare i costi dei collegamenti tra nodi adiacenti (C[i,j]=MAX_INT se i e j non adiacenti);
- un vettore **D** di N interi per rappresentare i costi calcolati dei cammini minimi (D[i] indicherà il costo calcolato del cammino dal nodo 0 al nodo i);
- un vettore **P** di N interi per indicare l'immediato predecessore di un nodo nel cammino calcolato;
- un vettore **Band** di NxN interi per rappresentare l'ampiezza di banda dei collegamenti tra nodi adiacenti (Band[i,j]=0 se i e j non adiacenti).

L'algoritmo deve restituire, oltre al vettore D, un vettore **nextHop** di N interi tale che, per ogni destinazione i, nextHop[i] indichi il nodo a cui inviare i pacchetti destinati al nodo i e un vettore **guaranteedBand** di N interi tale che guranteedBand[i] indichi la banda garantita sul cammino da 0 a i.

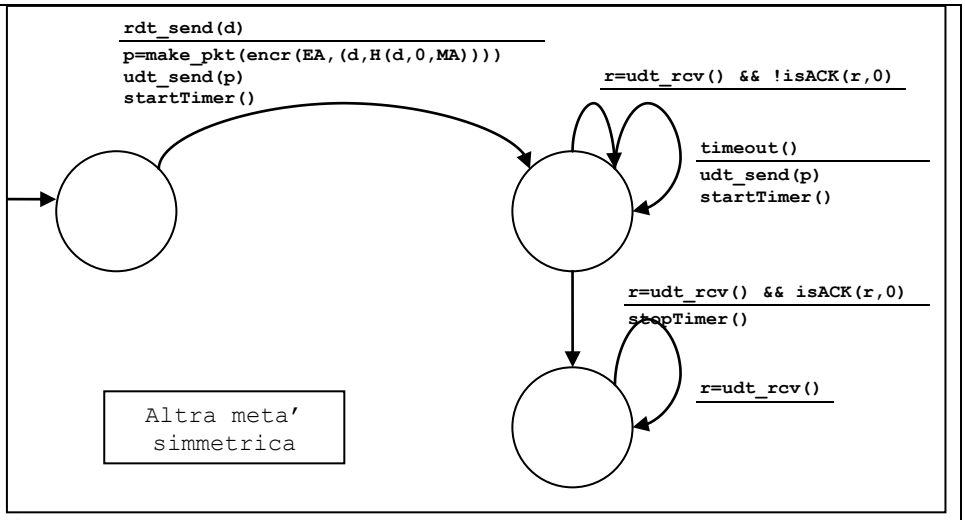
E5 (3 punti)**.** Consideriamo una rete KaZaA in cui ogni super peer ha X figli ed è connesso con al più altri N super peer. Assumiamo che ogni super peer inoltri con flooding le richieste che riceve a tutti i suoi vicini super peer e che il valore iniziale del campo di conteggio dei messaggi di richiesta sia K. (a) Determinare il limite superiore L al numero di messaggi di richiesta trasmessi a seguito di una richiesta generata da un peer "ordinario" (ovvero "non super"). (b) Assumendo che ogni peer (ordinario o super) pubblichi F file, e che nessun file sia pubblicato da più di un peer, determinare il limite superiore M al numero di file che saranno esaminati a seguito di una richiesta generata da un peer "ordinario".

TRACCIA DELLA SOLUZIONE

E1



E2. Affinche' il protocollo *alternating bit* garantisca riservatezza e integrita' dei dati trasmessi, possiamo estenderlo utilizzando i meccanismi impiegati in SSL. Siano E_A e E_B le chiavi di cifratura di sessione per i dati inviati da A a B e da B a A rispettivamente, e siano M_A e M_B le chiavi MAC di sessione per i dati inviati da A e da B rispettivamente e sia H la funzione di hashing crittografico utilizzata. Assumiamo che mittente (A) e destinatario (B) abbiano gia' concordato di condividere E_A, E_B, M_A, M_B e H durante la fase di autenticazione



dove: isACK(r,i) = (decr(EB(r)) == <"ACK",H("ACK",i,H=MB)>)

E3. (a) Se ACK=X+2MSS in B1 allora in t1 il valore della finestra di congestione CongWin e' maggiore di 4MSS ed A potrebbe quindi spedire 2MSS di nuovi dati. Dato che A spedisce solo 1MSS di dati abbiamo che RcvWin=1MSS in B1, SEQ=X+2MSS in A1 e, dato che in t2 scatta il timeout, SEQ=X+2MSS anche in A2. Appena riceve B2, A spedisce un nuovo segmento; dato che in t2 la sua CongWin diventa 1MSS ed ha gia' 1MSS "in volo", B2 deve essere un riscontro positivo[1], ovvero ACK=X+3MSS in B2 e RcvWin≥1MSS in B2. Infine, A3 conterra' 1 MSS di dati e SEQ=X+3MSS in A3.

(b) Con un ragionamento analogo al caso precedente, osserviamo che se ACK=X+1MSS in B1 allora RcvWin=2MSS in B1 e SEQ=X+2MSS in A1. Allo scadere del timer in t2, A rispedira' il segmento piu' vecchio ancora in volo, ovvero SEQ=X+1MSS in A2. B2 deve contenere un riscontro positivo e abbiamo quindi due possibile sotto-casi:

(b1) Se ACK=X+3MSS in B2 allora RcvWin≥1MSS in B2

(b2) Se ACK=X+2MSS in B2 allora RcvWin≥2MSS in B2

In entrambi i casi A3 conterra' 1MSS di dati e SEQ=X+3MSS in A3.

(c) Se ACK=X in B1 e A spedisce un segmento in t1, allora B1 e' un riscontro duplicato ricevuto per la terza volta; la finestra di congestione CongWin di A viene quindi dimezzata e SEQ=X in A1, cosi' come SEQ=X in A2. B2 deve contenere un riscontro positivo e abbiamo quindi due sotto-casi possibili:

(c1) Se ACK=X+2MSS in B2 allora RcvWin=1MSS in B2

(c2) Se ACK=X+1MSS in B2 allora RcvWin≥2MSS in B2

In entrambi i casi A3 conterra' 1MSS di dati e SEQ=X+2MSS in A3.

[1] Se B2 fosse un riscontro negativo con RcvWin=0, A dovrebbe attendere (almeno) lo scadere del timeout prima di inviare il primo segmento di "zero window probing".

E4.
```
//inizializzazione
for (i=1;i<N;i++){
      coloured[i]=0;
      D[i]=C[0,i];
      P[i]=0;
}
for (j=1;j<N;i++){
      w=0; dw=MAX_INT;
      for (i=1;i<N;i++)
            if (coloured[i]==0 && D[i]<dw){w=i;dw=D[i];}
      coloured[w]=1;

      for (v=1;v<N;i++)
            if (coloured[v]==0 && C[w,v]!=MAX_INT && D[w]+C[w,v]<D[v])
                  {D[v]=D[w]+C[w,v]; P[v]=w;}
}
for (i=1;i<N;i++){
      guaranteedBand=Band(P[i],i);
      NextHop[i]=P[i];
      while (NextHop[i]!=0){
            if (Band[P[NextHop[i]],NextHop[i]]<guaranteedBand)
                  guaranteedBand = Band[P[NextHop[i]],NextHop[i]];
            NextHop[i]=P[NextHop[i]];
}
```

E5. a) $L = 1 + N + N(N-1) + N(N-1)^2 + \ldots + N(N-1)^{K-1} = 1 + N\sum_{i=0}^{K-1}(N-1)^i$. b) $M = (X+1)FL$.

(AB)