RETI DI CALCOLATORI – prova scritta del 9/2/2018

Per essere ammessi alla prova orale è necessario ottenere una valutazione sufficiente della prima parte e una votazione totale di 15 o superiore.

Prima parte (15 punti)

Matricola_____Cognome_____Nome_____

Q1. Un server DNS D invia ad un altro server DNS E una query e riceve una risposta contenente un resource record di tipo NS. Da che cosa D riesce a dedurre che l'indirizzo del prossimo server da contattare è IPv4 oppure IPv6?

RISPOSTA: lo deduce da**l resource record nella sezione supplementare, relativa al nome simbolico del prossimo server da contattare,** ed è IPv4 se **il campo tipo di tale resource record è A** mentre è IPv6 se **il campo tipo di tale resource record è AAAA**

Q2. Al tempo t_0 un host A apre una connessione TCP con il suo server di posta elettronica. Tra il tempo t_0 ed il tempo t_1 A riceve, nell'ordine, 4 riscontri non duplicati, seguiti da 4 riscontri duplicati e da 2 riscontri non duplicati, e non riceve niente altro tra i tempi t_0 e t_1 . Sapendo che tra t_0 e t_1 non scade alcun timeout, e che il valore di cwnd al tempo t_1 è 29/10, indicare il valore iniziale (cioè al tempo t_0) della soglia ssthresh.

RISPOSTA: Il valore di ssthresh al tempo t_0 è **maggiore di 4.**

Q3. Quanti e quali timer usa RIP? E per quali funzionalità?

RISPOSTA: RIP usa **3** timer, che sono:
timer **periodico** che serve per **inviare messaggi di aggiornamento almeno ogni 25-35 secondi**
timer **di scadenza** che serve per **la validità dei percorsi (è il TTL)**
timer **per la garbage collection** che serve per **eliminare i percorsi scaduti dopo un po' di tempo** (i percorsi scaduti si mantengono per un po' anche dopo la scadenza)

Q4. In quale protocollo di SSL vengono generate le chiavi crittografiche e i vettori di inizializzazione?

RISPOSTA: le chiavi crittografiche e i vettori di inizializzazione vengono generate nel protocollo **Handshake**

Q5. Una rete *Ethernet standard* lunga 2200 metri, ha una velocità di trasmissione effettiva di 12 Mbps e la velocità di propagazione nella rete è di $2x10^8$ m/sec, la durata del jamming signal è uguale al tempo di trasmissione di un indirizzo MAC, mentre la durata del tempo di sensing è uguale al doppio del tempo di vulnerabilità. Al tempo t, due nodi A e B iniziano *contemporaneamente* ad eseguire il protocollo MAC della rete (in modalità 1 persistente), entrambi per trasmettere un frame di 1518 byte. Se, dopo la collisione, A genera il numero casuale di attesa 0, mentre B genera 1, e se, dopo una collisione, il protocollo viene ripetuto dopo l'attesa del tempo casuale e subito dopo la fine della ricezione del jamming signal da parte dell'altro nodo, quando termina la trasmissione con successo del frame di A? Si supponga che nessun altro nodo della rete debba trasmettere.

RISPOSTA: La trasmissione con successo del frame di A termina al tempo t+ **44+11+11+4+44+1012= t+1126** microsecondi.

Seconda parte

E1 (8 punti). In un host, il lato destinatario di TCP Reno viene realizzato mediante tre threads: uno che si occupa delle interazioni con il livello applicativo, uno che si occupa delle interazioni con il timer e dell'invio dei riscontri nuovi, ed uno che si occupa delle interazioni con il livello di rete. Descrivere con uno *pseudocodice* il comportamento del thread che *interagisce con il livello di rete*. Per semplicità, si supponga che tutti i segmenti ricevuti siano lunghi 1 MSS, e che la finestra di ricezione possa contenere al massimo N segmenti. Si usino (tra le altre) le seguenti variabili e funzioni/procedure:
- *RecWind[]* finestra che contiene i dati corretti ricevuti e non ancora prelevati dal livello applicativo;
- *buildack(A)* che costruisce un riscontro A;
- *send(A)* per inviare il riscontro A (tramite il livello di rete);
- *checksegm(S)* funzione booleana che restituisce *true* se e solo se S è corretto;
- *wait(S)* per aspettare di ricevere dal livello di rete un segmento S;
- *insert(A,B,C)* per inserire il segmento A nella finestra B a partire dalla posizione C.

Descrivere il contenuto o la funzionalità delle altre variabili e funzioni/procedure eventualmente utilizzate. Si trascurino i problemi di concorrenza nelle interazioni tra i thread di cui sopra, e la gestione della finestra come vettore circolare.

SOLUZIONE:

```
while true
 {
   wait(S);
   if checksegm(S)  //se S è corretto
   {
        if S.seqn==RF  //se S è il segmento atteso più vecchio (RF punta alla prima posizione libera della finestra; S.seqn
                                  //è il numero di sequenza del segmento)
          {
             insert(S,RecWind,RF); //metti S nella finestra di ricezione nella posizione RF
             arrivato(RF)=true;  //segnala che quel segmento è arrivato corretto
             while arrivato(RF)  // scandisci la finestra fino al primo segmento non ancora ricevuto
                   {RF=RF+1*MSS};
             ACK=buildack(RF);  // il riscontro verrà eventualmente inviato dal thread che interagisce con il timer
          }
       if ((S:seqn>RF)&&(!arrivato(S:seqn)&&(S.seqn<FirstForAppl+N*MSS))  //se S è un segmento nuovo e non ancora
                                        // memorizzato ma che sta nella finestra; FirstForAppl punta al primo segmento che
                                        // deve essere ancora passato al livello applicativo
          {
              insert(S,RecWind,S.seqn);
              send(ACK);
           }
       if (( S.seqn<RF)|| (S.seqn>FirstForAppl+N*MSS)) //se S è doppione o se è fuori finestra
          { send(ACK); }
   }
 }
```

E2 (7 punti). Si consideri una variante dell'algoritmo *path vector* in cui il controllo se un nodo X è presente nel percorso di un path vector ricevuto da un vicino Y non viene effettuato dal nodo stesso, ma dal vicino Y: Y invia ad X il path vector ricevuto (ed opportunamente modificato) se e solo se X *non* è presente nel percorso (cioè si applica la tecnica *split horizon*). La politica utilizzata da tutti i nodi è quella del costo minimo: ogni link ha associato un costo, e *C[]* è l'array che contiene i costi aggiornati dei collegamenti. Il percorso ricevuto verrà eventualmente utilizzato per l'origine del path vector (ovviamente in ordine inverso rispetto a quanto ricevuto: si suppone che i collegamenti siano tutti bidirezionali). Descrivere con uno pseudocodice il comportamento di un nodo R che utilizza la variante su riportata di path vector, limitatamente alle azioni conseguenti alla ricezione di un path vector da un vicino V, e contenuto nella variabile *adv* che ha, tra gli altri, il campo *path* , vettore che contiene il percorso ricevuto, il campo *pathlength* che indica il numero di elementi di tale percorso, e *cost* che indica il costo di tale percorso. Nello pseudocodice si utilizzino le variabili e funzioni/procedure seguenti:
- *bestpath[i]* che contiene il percorso migliore per la destinazione i, e che ha gli stessi campi di adv sopra esposto
- *send(P,A,B)* usata da A per inviare a B il percorso P
- *wait(P,A,B)* usata da B per ricevere da un vicino A (parametro di output della procedura) il percorso P.

Descrivere il contenuto o la funzionalità delle altre variabili e funzioni/procedure eventualmente utilizzate. Per semplicità, i vicini di R sono rappresentati dai numeri interi tra 1 ed M.

SOLUZIONE:

```
while true
  {
     wait(adv,V,R);
     if ((adv.cost+C[V,R])<bestpath[path[1]].cost)  //controlla se il path ricevuto è migliore di quello già presente per la
                                                    // destinazione (che è l'origine del path vector)
        {
            bestpath[path[1]].path=adv.path;  //aggiorna il path
            bestpath[path[1]].cost=adv.cost+C[V,R]  //ed il suo costo
            for i=1 to adv.pathlength  //scorri il path arrivato
                {
                    if adv.path[i] <= M  //se c'è un vicino, segnala di non mandargli il path
                        {nosend[adv.path[i]]=true;}
                }
            adv.pathlength++;  //aggiorna il path inserendo R in esso
            adv.path[pathlength]=R;
            for i=1 to M //manda il path ai vicini che non sono in esso (split horizon)
                { if (!nosend[i])
                        {send(adv,R,i);}
                 }
        }
  }
```