RETI DI CALCOLATORI – prova scritta del 9/7/2018

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. Il mittente *pippo@disney.com* invia un messaggio di email al destinatario taz@warner.com contentente il testo *OK*. Indicare il contenuto dei campi richiesti sotto (nella parte *risposta* di questo quesito), dei primi due segmenti TCP inviati dal mailserver di pippo, *mailserv.disney.com* al mailserver di taz, *ms.warner.com ,* per poter inviare tale messaggio.

RISPOSTA*:* il *primo* messaggio conterrà: porta origine **P (scelta tra le porte private)**, porta destinazione **25** , flags a 1: **SYN** , dati **nessun dato**, SEQNUM **Y (scelto dal mittente)**, ACKNUM **vuoto (non significativo)**
il *secondo* messaggio conterrà: porta origine **P** , porta destinazione **25**, flags a 1: **ACK**, dati **nessun dato** , SEQNUM **Y+1** , ACKNUM **Z+1 (se il destinatario aveva scelto Z come SEQNUM iniziale)**

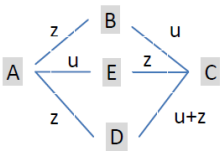
Q2. La trasmissione di alcuni segmenti TCP su una connessione appena stabilita avviene con la seguente sequenza di valori consecutivamente assunti dalla soglia ssthresh, dovuti ad eventi indicatori di congestione : …,8,2,3,109/60,29/20,.... Dire quali eventi indicatori di congestione si sono verificati, nonché il valore di cwnd e lo stato dopo tali eventi (e prima di qualunque altro evento successivo).

RISPOSTA:
passaggio da 8 a 2: evento **3 ack duplicati** valore di cwnd **5** stato **fast recovery**
passaggio da 2 a 3: evento **time out** valore di cwnd **1** stato **slow start**
passaggio da 3 a 109/60: evento **time out** valore di cwnd **1** stato **slow start**
passaggio da 109/60 a 29/20: evento **tre ack duplicati** valore di cwnd **89/20** stato **fast recovery**

Q3. Siano A e B due server, di indirizzo IP 192.168.5.5 e 192.168.6.6, rispettivamente, ed entrambi situati dietro un router NAT , avente indirizzo privato 192.168.1.1 ed indirizzo pubblico 141.131.121.111. Supponiamo che un host C, esterno al NAT, chieda al proprio server DNS di risolvere l'indirizzo simbolico di A e subito dopo quello di B. Indicare – giustificando la risposta – quali indirizzi IP riceverà C in risposta alle sue due richieste.

RISPOSTA: C riceverà come risposte gli indirizzi IP **141.131.121.111** e **141.131.121.111**

Q4. Si consideri la rete a lato, in cui i nodi utilizzano l'algoritmo distance vector con poisoned reverse e in cui **non** è nota la relazione tra u e z (u e z interi positivi). Descrivere il contenuto del vettore delle distanze calcolato dal nodo C, e (delle ultime copie) dei vettori che C ha ricevuto dai suoi vicini, quando la rete ha raggiunto lo stato di quiescenza, nei due casi in cui $u < z$ ed $u > z$. I valori nei vettori delle distanze siano ordinati alfabeticamente per destinatario.



RISPOSTA: vettore ricevuto da B: *se u<z* : **z|0|2z** *(se 2u>z)* **oppure inf|u+z oppure inf** *se u>z* **z|0|2z|u+z oppure inf**
vettore ricevuto da D: *se u < z*: **z|2z** *(se 2u>z)* **oppure inf|0|u+z** *se u > z* **z|2z|0|u+z**
vettore ricevuto da E: *se u < z*: **u|u+z oppure inf|u+z|0** *se u > z* **u|u+z oppure inf|u+z|0**
vettore *calcolato* da C: *se u < z* **u+z|u|u+z|z** *se u > z* **u+z|u|u+z|z**

Q5. Una rete Ethernet lunga 240 metri ha frequenza di trasmissione di 12 Mbps e velocita' di propagazione di $2x10^8$ m/sec. Un nodo A della rete inizia a trasmettere un frame di 128 bytes al tempo t0. A partire dalla trasmissione di quale bit A sarà **sicuro** che il suo frame non subirà collisioni?

RISPOSTA: A sarà **sicuro** che il suo frame non subirà collisioni a partire dal **30** esimo bit.
Il ritardo di propagazione e' di $240/2x10^8$ = 1,2 microsec. Un altro eventuale nodo della rete, diciamo B, che volesse trasmettere, si accorgerebbe (nel caso pessimo) della trasmissione di A dopo 1,2 microsec. Quindi, trasmetterebbe solo entro tale tempo. Il primo bit eventualmente trasmesso da B arriverebbe ad A non dopo ulteriori 1,2 microsec. Quindi, dopo 2,4 microsec da t0, A puo' essere sicuro che il suo frame non avra' collisioni. 2,4 microsec equivalgono alla trasmissione di 29 (28,8 per la precisione) bits: quindi A può essere sicuro *durante* la trasmissione del 29 esimo bit. Pertanto *a partire* dalla trasmissione del *30-esimo* bit in poi A puo' essere sicuro che il suo frame non subira' collisioni.

Seconda parte

E1 (7 punti). Una rete di calcolatori inaffidabile utilizza il protocollo **Selective Repeat** per mascherare l'inaffidabilita'. Una variante del protocollo, detta **Selective Repeat-SRT**, viene utilizzata per gestire l'invio dei flussi di dati in tempo reale (soft real time). Per tali flussi, ogni pacchetto della sequenza deve essere ricevuto entro una determinata scadenza per poter essere efficacemente utilizzato, altrimenti viene scartato perchè inutile. In particolare, i pacchetti della sequenza devono essere ricevuti ogni T (costante di sistema nota al protocollo) istanti di tempo. Supponiamo che il primo pacchetto della sequenza arrivi corretto al tempo t0. Questo viene sempre passato alla applicazione. In generale, il pacchetto i-esimo (i>1) deve arrivare corretto entro t0+(i-1)T per essere passato alla applicazione, altrimenti, se arriva corretto più tardi, alla scadenza di cui sopra (cioè al tempo t0+(i-1)T) si invia al mittente il riscontro di quel pacchetto (per evitare inutili ritrasmissioni), si inseriscono nella finestra, al suo posto, i dati contenuti nella variabile *neutraldata*, e si passa a considerare il pacchetto (i+1)-esimo, e quando il pacchetto i-esimo arriverà dopo la sua scadenza, verrà ignorato. Questa politica viene applicata a tutti i pacchetti della sequenza (tranne il primo: vedi sopra). Descrivere, mediante un automa a stati finiti che utilizza *pseudocodice* (come nell'automa *[SR]* del materiale didattico) e *non* con descrizione delle attività a parole, il comportamento del **receiver** della variante del protocollo Selective Repeat sopra descritta.

SOLUZIONE: L'automa ha un solo stato, e oltre alle transizioni di [SR] ha una transizione in più dovuta a timeout. Si riportano solo le transizioni (in grassetto le modifiche/aggiunte relative ad [SR]).

```
rcv_base=1
forEach y: isReceivedSgmt[y]=false
next_to_receive=1
```

```
rcvSgm=UDT_rcv() && (corrupted(rcvSgm)|| !isInEWindow(rcvSgm))
```

```
NOP
```

```
rcvSgm=UDT_rcv() && !corrupted(rcvSgm)&&isInEWindow(rcvSgm)
```

```
y=seqN(rcvSgm)
if (isReceivedSgmt[y]==false)
    {rcvSgmt[y]=extract(rcvSgm); isReceivedSgmt[y]=true}
sndSgm = make_segment(ACK,y)
UDT_send(sndSgm)
if (y==rcv_base)
    while (isReceivedSgmt[rcv_base]==true) do
   {      deliver_data(rcvSgmt[rcv_base])
          isReceivedSgmt[rcv_base]=false
          rcv_base++
    }
```

timeout

```
if  ((isReceivedSgmt[next_to_receive]==false) && ( next_to_receive>1))  // in questo caso  next_to_receive= rcv_base
  {
       rcvSgmt[next_to_receive]=netraldata;
       isReceivedSgmt[next_to_receive]=true;
       sndSgm = make_segment(ACK,next_to_receive);
       UDT_send(sndSgm);
       while (isReceivedSgmt[rcv_base]==true) do  // in questo caso  next_to_receive= rcv_base
                                    {
                                      deliver_data(rcvSgmt[rcv_base])
                                      isReceivedSgmt[rcv_base]=false
                                      rcv_base++
                                     }
  }
next_to_receive++;
startTimer(T);  // anche: startTimer((rcv_base- next_to_receive+1)T)  (ottimizzazione)
```

E2 (8 punti). Descrivere in modo dettagliato (il più dettagliato possibile) mediante pseudocodice il comportamento di un *receiver* del protocollo CSMA/CA, di indirizzo *mioaddress*, limitatamente alle azioni relative alla ricezione di un RTS. Il protocollo è realizzato mediante threads. Si usino le seguenti funzioni/procedure:

- *receive(f)* e *send (f)* per ricevere ed inviare il frame f;
- *checksum(f.x)* per calcolare la cheksum di f e confrontarla con il campo x
- *makeFC(f.x)* per calcolare il campo FC del CTS a partire da quello di f
- *computecheck(c)* per calcolare la cheksum di c
- *starttimer(t)* per far partire il timer con valore iniziale t
- *wait(e)* per aspettare l'evento e
- *oktoreceive* variabile booleana: se *false* non si può ricevere niente

Descrivere il contenuto o la funzionalità delle altre variabili e funzioni/procedure eventualmente utilizzate.

***SOLUZIONE**:*

```
receive(frame);
if checksum(frame.FCS) // se è corretto
 {
    if (frame.type==RTS)&& oktoreceive&&(frame.address1=mioaddress) //se è RTS destinato a me e posso ricevere
        {
            oktoreceive=false;  // il protocollo è occupato in una ricezione (non può ricevere altro)
            oktosend=false;  // gli altri thread non possono trasmettere
            CTS.FC=makeFC(frame.FC);  // si copia FC
            CTS.D=frame.D; //campo che serve per il NAV
            CTS.address1=frame.address2;
            CTS.FCS=computecheck(CTS);
            starttimer(SIFS);
            wait(timeout);  // aspetta un SIFS
            send(CTS);
            starttimer(CTS.D);
            wait(timeout);  //aspetta il NAV
            oktosend=true;  //da ora si può trasmettere
        }

 }
```