RETI DI CALCOLATORI – prova scritta straordinaria del 31/10/2017

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 (18 punti)

Matricola_____Cognome_____Nome_____Posto_____Fila_____

Q1. Un server DNS locale invia una query di tipo A ad un server DNS top level domain e riceve una risposta R di tipo A. La query è stata risolta in modo iterativo o ricorsivo?

SOLUZIONE: la query è stata risolta in modo **iterativo o ricorsivo, non si può distinguere se il tipo della risposta è A.**

Q2. Quali protocolli di livello applicativo, e quando (cioè sotto quali ipotesi), **non** inviano **mai** ack di livello trasporto in piggybacking?

SOLUZIONE: protocollo **DNS** quando **usa UDP** , protocollo **FTP** quando **trasferisce dati**

Q3. Un client C apre una connessione TCP con un server FTP F sul canale di controllo. Indicare i valori contenuti nei campi seq, ack, e rwnd ed i nomi dei flags a true dei segmenti scambiati per l'apertura della connessione, nell'ipotesi che i numeri di sequenza iniziali di C ed F siano 800 e 2300, rispettivamente, e che inizialmente le finestre di ricezione di C ed F siano di 12000 e 7000 byte, rispettivamente.

SOLUZIONE:
primo segmento: seq= **800** ack= **?** Rwnd= **?** flags a true= **SYN**
secondo segmento: seq= **2300** ack= **801** rwnd= **7000** flags a true=**SYN+ACK**
terzo segmento: seq= **800** (ack non in piggybacking, per FTP) ack= **2301** rwnd= **12000** flags a true= **ACK**

Q4 (6 punti). Al tempo t_0 un server web W ha una connessione TCP aperta con un client C, per cui il valore di cwnd è 6, ed ha nella sua finestra di invio tre segmenti di 1 MSS in volo (con S_f = Y), ed altri 2 MSS di dati nuovi da spedire. Il primo evento successivo a t_0 è la ricezione di un riscontro R, a seguito del quale invia i dati nuovi in finestra: la fine di questo invio avviene al tempo t_1. Indicare il possibile stato, il valore di rwnd e di ssthreshold al tempo t_0 e il possibile stato, il valore di rwnd, di ssthreshold e di cwnd al tempo t_1.

SOLUZIONE:
- Lo stato al tempo t_0 può essere *slow start*? **SI** Se si, al tempo t_0 il valore di rwnd è **3MSS** e quello di ssthreshold è **> 6MSS** mentre al tempo t_1 lo stato è **C.A. o S.S.** *(dipende dalla soglia)*, il valore il valore di rwnd è **>=5MSS-X+Y**, quello di ssthreshold è **<=7MSS se lo stato è C.A.; >7 se lo stato è S.S.** e quello di cwnd è **7** ;

- Lo stato al tempo t_0 può essere *congestion avoidance*? **SI** Se si, al tempo t_0 il valore di rwnd è **3MSS** e quello di ssthreshold è **6MSS** mentre al tempo t_1 lo stato è **Congestion Avoidance** , il valore il valore di rwnd è **>=5MSS-X+Y**, quello di ssthreshold è **6MSS** e quello di cwnd è **37/6 MSS** ;

- *(Q6)* Lo stato al tempo t_0 può essere *fast recovery*? **SI** Se si, al tempo t_0 il valore di rwnd è **3MSS** e quello di ssthreshold è **<= 3MSS** *(e >=2MSS)* mentre *(Q7)* al tempo t_1 lo stato è **Congestion Avoidance** , il valore il valore di rwnd è **>=5MSS-X+Y**, *(e >=ssthreshold)* quello di ssthreshold è **<= 3MSS** *(come a t_0)* e quello di cwnd è **uguale ad ssthreshold** ;

Q5. In uno sistema autonomo si usano sia RIP che OSPF per il routing interno al sistema autonomo. I cammini ottenuti sono sempre gli stessi per i due protocolli, oppure no? Si supponga che i due protocolli operino sugli stessi dati e che non avvenga nessun cambiamento nel sistema autonomo durante l'esecuzione dei protocolli.
SOLUZIONE:
si perchè_____
oppure
no **perché i due protocolli possono usare metriche diverse: RIP usa sempre hop count, OSPF non è detto.**

Seconda parte

E1 (6 punti). Si consideri una variante del protocollo **Go Back N** in cui la ritrasmissione dei pacchetti non avviene con la regola normalmente utilizzata nel protocollo del materiale didattico, ma avviene utilizzando la tecnica usata in **TCP** versione **Tahoe**. Per il resto, si comporta come il normale protocollo Go Back N.

Descrivere, mediante un automa a stati finiti che utilizza pseudocodice (come nell'automa [GBN] del materiale didattico) e non con descrizione delle attività a parole, il comportamento del **sender** della variante del protocollo Go Back N sopra descritta. Si supponga che il receiver si comporti correttamente rispetto a tale tecnica.

L'automa ha un unico stato. Si riportano solamente le transizioni. Le modifiche rispetto all'automa di base sono in grassetto.

RDT_send(data)

```
if (nextseqnum < base+N) {
   sndSgm[nextseqnum] = make_segment(nextseqnum,data)
   UDT_send(sndSgm[nextseqnum])
   if (base == nextseqnum)
      start_timer
   nextseqnum++
                                    }
else
   refuse_data(data)
```

Inizializzazione

---

base=1

nextseqnum=1

ndup=0

rcvSgm=UDT_rcv() &&(corrupted(rcvSgm) || ! isACKinWindow(rcvSgm))

NOP

timeout()

UDT_send(sndSgm[base])

start_timer()

rcvSgm=UDT_rcv() && ! corrupted(rcvSgm) && isACKinWindow(rcvSgm)

---

base = getacknum(rcvSgm)

If (base == nextseqnum) stop_timer() else start_timer()

ndup=0

le due seguenti transizioni possono anche essere unificate, utilizzando un *if* per distinguere i due casi sul valore di ndup

rcvSgm=UDT_rcv() && ! corrupted(rcvSgm) **&& getacknum(rcvSgm)<base&&ndup<2**

---

ndup++

rcvSgm=UDT_rcv() && ! corrupted(rcvSgm) **&& getacknum(rcvSgm)<base&&ndup=2**

---

ndup=0

UDT_send(sndSgm[base])

start_timer()

E2 (6 punti). In una rete wifi aperta, si decide di realizzare le azioni svolte dall'AP (avente SSID=X e indirizzo MAC M) per associare un nuovo host, a livello applicativo. Le azioni sono tali da permettere all'host (alla fine dell'associazione) di iniziare a ricevere e inviare traffico in internet. Descrivere con uno pseudocodice (in cui sono spiegate le funzionalità dei comandi utilizzati) le azioni su descritte.

Si danno due versioni: quella completa (la prima) e quella succinta (la seconda). La prima è più completa: risponde anche alla domanda occulta (su DHCP).

```
While true {
        receive(MSG);
        if MSG.type==richiestaAssociazione
                {
                 IPA=SelectFreeAdrress(); // funzione che sceglie un IP libero: se non esiste, allora risponde NULL
                 if IPA!=NULL
                        {
                         risp.type=rispostaAssociazione; //costruzione della risposta
                         risp.MACdest=MSG.MACmitt;
                         risp.MACmitt=AP_MAC;  //AP_MAC= MAC dell'AP
                         send(risp);
                         receive(MSG);
                         if MSG.type==DHCPdiscover
                                {
                                 offer.type=DHCPoffer;
                                 offer.transID=MSG.transID;
                                 offer.leasetime=APLT;  //APLT= lease time dell'AP
                                 offer.YourAddress=IPA;
                                 offer.ServerAddress=AP_IP; //AP_IP= indirizzo IP dell'AP
                                 send(offer);
                                 receive(MSG);
                                 if MSG.type==DHCPrequest
                                        {
                                         ack.type=DHCPACK;
                                         ack.transID=MSG.transID;
                                         ack.leasetime=offer.leasetime;
                                         ack.YourAddress=IPA;
                                         ack.ServerAddress=AP_IP;
                                         send(ack);
                                         occupyAddr(IPA);  /per settare ad occupato l'indirizzo IP in IPA
                                        }
                                }
                        }
                }
        }
```

Nella versione succinta non si costruiscono i messaggi, ma si mandano e basta.

```
While true {
        receive(MSG);
        if MSG.type==richiestaAssociazione
                {
                 IPA=SelectFreeAdrress();
                 if IPA!=NULL
                        {
                         risp=rispostaAssociazione;
                         send(risp);
                         receive(MSG);
                         if MSG.type==DHCPdiscover
                                {
                                 offer=DHCPoffer;
```

```
                    send(offer);
                    receive(MSG);
                    if MSG.type==DHCPrequest
                        {
                         ack=DHCPACK;
                         send(ack);
                         occupyAddr(IPA);
                        }
                    }
                }
            }
        }
```