UNIVERSITÀ DEGLI STUDI DI PALERMO — Dipartimento di Ingegneria

21 Gennaio 2022

# Reti di Calcolatori e Internet (9 CFU)

A.A. 2021/2022

*Docente: Prof. Alessandra De Paola*

(Durata 2:30h)

## Parametri della prova

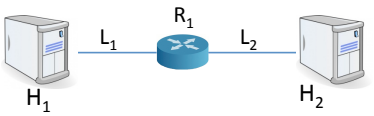
Sia $[X_0\ X_1\ X_2\ X_3\ X_4\ X_5\ X_6\ ]$ il numero di matricola del candidato.

Si calcolino i seguenti parametri:

| | |
|---|---|
| $Y_1 = \lfloor \frac{2*X_6}{5} \rfloor + 52$ | $Y_4 = Y_2 + 5$ |
| $Y_2 = \lfloor \frac{X_5}{5} \rfloor + 3$ | $Y_5 = Y_2 + 1$ |
| $Y_3 = \lfloor \frac{X_4}{5} \rfloor + 70$ | |

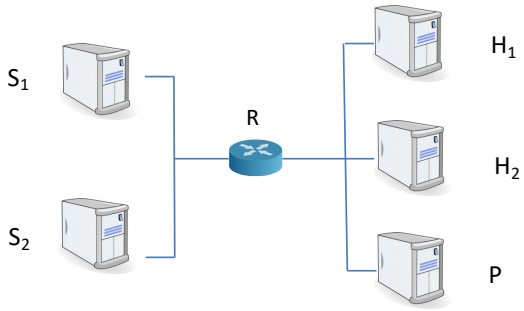
## Quesito 1

Dati due host collegati come in figura, si supponga che l'host $H_1$ invii all'host $H_2$ un file di dimensioni $Y_4$ KB tramite il protocollo TCP, con valore iniziale della soglia sulla finestra di congestione pari a 8 MSS. Si assuma che i link abbiamo entrambi ampiezza di banda pari a $R$=100 Mbps, RTT pari a 0.5 ms, MSS di 512 byte ed overhead trascurabile. Si determini il tempo necessario alla trasmissione dell'intero file, nel caso in cui si perda l'ack del segmento $Y_5$ e che il timeout che scada 2 ms dopo il termine della trasmissione del segmento a cui e' assegnato (si indichino esplicitamente i valori della finestra di congestione e gli specifici ack inviati). Si calcoli infine il throughput della trasmissione.



## Quesito 2

Si consideri la configurazione schematizzata in figura, in cui gli host $H_1$ e $H_2$ sono configurati per navigare il web usando il server proxy P, tramite il protocollo HTTP 1.1. Si assuma che l'host $H_1$ richieda per la prima volta una pagina web costituita da un file HTML di $Y_1$ KB ospitato dal server $S_1$, e dall'immagine $Img_1$ di $Y_2$ MB, ospitata dal server $S_2$. Si assuma inoltre che, successivamente, l'host $H_2$ richieda una seconda pagina al server $S_2$, costituita da un file HTML di $Y_3$ KB e dalla stessa immagine $Img_1$ chiesta in precedenza da $H_1$.
Si descrivano tutte le interazioni tramite opportuni diagrammi di sequenza e si calcoli il tempo totale necessario ad ottenere le due pagine, assumendo che tutti i collegamenti siano caratterizzati da ampiezza di banda pari a 100 Mbps, tempo di propagazione medio pari a 0.10 ms, MTU pari a 1500 Byte e overhead di 40 Byte.




## Quesito 3

Il candidato descriva il funzionamento dei protocolli di routing Inter-AS e Intra-AS, indicandone gli scopi, evidenziando le differenze tra le due tipologie di routing, e spiegando le interazioni e le dipendenze tra i protocolli che appartengono alle due classi.

## Quesito 4

Completare il codice fornito per realizzare il client di un'applicazione basata sul protocollo TCP, che consenta di ottenere l'orario dal server solo se si e' a conoscenza di una parola segreta. Il client, al momento dell'avvio deve accettare da riga di comando l'indirizzo e la porta a cui contattare il server. Deve poi chiedere all'utente di inserire la parola segreta ed inviarla al server. Nel caso in cui la parola segreta sia corretta, ricevera' dal server l'ora corrente, che andra' visualizzata sullo standard output prima di terminare l'esecuzione. Nel caso in cui la parola segreta sia errata, il client informera' l'utente dell'errore e richiedera' la parola segreta. Dopo tre tentativi falliti il client terminera' la sua esecuzione.

```c
#include <stdio.h>
#include <stdlib.h>      /* exit() */
#include <strings.h>     /* bzero(), bcopy() */
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(char *msg) {
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[]) {
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256], pwd[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s servername port\n", argv[0]);
       exit(0);
    }

    portno = atoi(argv[2]);

    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
          (char *)&serv_addr.sin_addr.s_addr,
          server->h_length);
    serv_addr.sin_port = htons(portno);
```

**Note:**
Per le dimensioni relative ai file si considerino le grandezze come potenze di 2 e quindi in particolare:
1 MB = 1.024 kB 1 kB = 1.024 byte
Per le dimensioni relative ai tassi di trasmissione e alle ampiezze di banda si considerino le grandezze come potenze di 10 e quindi in particolare:
1 kbps = 1.000 bps 1 Mbps = 1.000.000 bps

# Documentazione Programmazione Socket

*//Structures for handling internet addresses*
struct sockaddr_in {// ...
short sin_family; // *e.g. AF_INET, AF_INET6*
unsigned short sin_port; // *e.g. htons(3490)*
struct in_addr sin_addr; // *see struct in_addr* };

struct in_addr{ unsigned long s_addr; //*e.g. INADDR_ANY* };

*//Structure for handling host names*
struct hostent{// ...
char *h_name; //*The real canonical host name.*
int h_addrtype; //*The result's address type, e.g. AF_INET*
int h_length; //*The length of the addresses in bytes, which is 4 for IP (version 4) addresses.*
char *h_addr; //*An IP address for this host.* };

int socket(int domain, int type, int protocol); //*allocates a socket descriptor*

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
//*accepts an incoming connection on a listening socket*

int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
//*associates a socket with an IP address and port number*

int listen(int sockfd, int queuelength); //*tells a socket to listen for incoming connections*

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen); //*connects - initiate a connection on a socket*

int close(int sockfd); //*closes a socket descriptor*

struct hostent *gethostbyname(const char *name); //*gets an IP address for a hostname*

*//Functions to convert multi-byte integer types from host byte order to network byte order (and viceversa)*
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);

*//Functions to convert IP addresses to human-readable form and back*
const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
int inet_pton(int af, const char *src, void *dst);

*//Functions to convert IP addresses from the IPv4 numbers-and-dots notation into binary form (in network byte order) and vice versa*
int inet_aton(const char *cp, struct in_addr *inp);
char* inet_ntoa(struct in_addr in);

ssize_t recv(int sockfd, void *buf, size_t len, int flags); //*receives data on a stream socket*
ssize_t send(int sockfd, const void *buf, size_t len, int flags); //*sends data out over a stream socket*

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen);
//*receives data on a datagram socket*

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

*//sends data out over a datagram socket*

ssize_t read (int fd, void *buf, size_t count); *//reads data from a stream socket*
ssize_t write (int fd, const void *buf, size_t count); *//writes data on a stream socket*

**Utility Functions**

void bzero(void *s, size_t n); *//sets the first n bytes of the area starting at s to zero*
void bcopy(const void *src, void *dest, size_t n); *// copies n bytes from src to dest.*

void* memset(void *s, int c, size_t n);
*// fills the first n bytes of the memory area pointed to by s with the constant byte c*

void* memcpy(void *dest, const void *src, size_t n);
*// copies n bytes from memory area src to memory area dest. The memory areas must not overlap*

int strcmp(const char *s1, const char *s2); *// compares the two strings s1 and s2.*
int strncmp(const char *s1, const char *s2, size_t n); *//compares the first n byte of s1 and s2.*

char *strcat(char *dest, const char *src); *// appends the src string to the dest string.*
char *strncat(char *dest, const char *src, size_t n);
*// appends the src string to the dest string, by using at most n bytes from src.*

char *strcpy(char *dest, const char *src);
*// copies the string pointed to by src, including the terminating null byte, to the buffer pointed to by dest.*

char *strncpy(char *dest, const char *src, size_t n);
*// copies the string pointed to by src, including the terminating null byte, to the buffer pointed to by dest, by using at most n bytes from src.*

size_t strlen(const char *str); *// calculates the length of the string str, excluding the terminating null byte.*

int rand(void); *// returns a pseudo-random number in the range of 0 to RAND_MAX.*