UNIVERSITÀ DEGLI STUDI DI PALERMO — Dipartimento di Ingegneria

18 Febbraio 2022

Reti di Calcolatori e Internet (9 CFU)

A.A. 2021/2022

Docente: Prof. Alessandra De Paola

(Durata 2:30h)

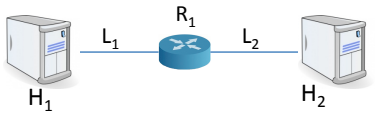
Parametri della prova

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:

$Y_1 = \lfloor \frac{2*X_6}{5} \rfloor + 50$	$Y_4 = Y_2 + 2$
$Y_2 = \lfloor \frac{X_5}{5} \rfloor + 5$	$Y_5 = Y_2 - 1$
$Y_3 = \lfloor \frac{Y_4}{5} \rfloor + 20$	$Y_6 = Y_5 - 1$

Quesito 1

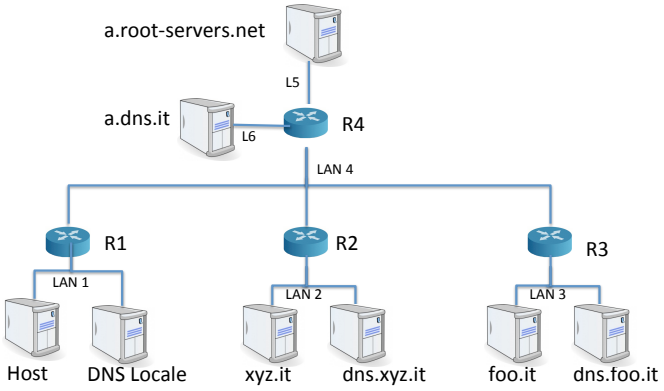
Si assume che tra due host collegati come in figura, con link caratterizzati da ampiezza di banda pari a 100 Mbps , RTT = 2 ms, MSS = 512 byte, sia gia' stabilita una connessione ormai a regime del protocollo TCP Reno. Si calcoli il tempo necessario alla trasmissione di un file di dimensione Y_4 KB dal primo al secondo host, assumendo che il valore della finestra di congestione all'inizio della trasmissione sia di Y_5 MSS e che si abbia un overhead di pacchetto pari a 20 byte. Si assuma inoltre che si perda il segmento di posizione Y_6 della trasmissione e che il timeout sia fissato a 4 ms.



[Continua .....]

Quesito 2

Data la configurazione schematizzata in figura, si assuma che l'host richieda la pagina web `www.xyz.it/page.html` costituita da una pagina HTML di Y_1 KB, dall'immagine `www.foo.it/pics/img1.jpg` di Y_2 MB e dall'immagine `www.xyz.it/pics/photo.jpg` di Y_3 KB. Assumendo che al momento della richiesta la cache del server DNS locale sia vuota, mostrare tramite un opportuno diagramma le richieste DNS e HTTP necessarie affinche' l'utente possa visualizzare la pagina richiesta, evidenziando le principali informazioni contenute nei messaggi. Si calcoli il tempo totale necessario per ottenere la pagina (incluse le richieste DNS), assumendo che tutti i collegamenti siano caratterizzati da ampiezza di banda di 100 Mbps, tempo di propagazione medio pari a 0.1 ms, MTU=1500 Byte e overhead trascurabile.



Quesito 3

Descrivere il ruolo dei router nelle reti a commutazione di pacchetto, illustrandone la struttura e il funzionamento. Si descrivano inoltre le operazioni svolte dai router alla ricezione di un datagramma, con particolare riferimento alla gestione dei campi dell'intestazione IP.

Quesito 4

Scrivere il codice core di un server UDP che implementa una lotteria distribuita. Il server, al suo avvio, estrae un numero casuale tramite una opportuna funzione. Ogni client ha a disposizione tre tentativi per indovinare il numero casuale estratto dal server. Quando un client invia al server un intero senza segno, il server verifica innanzi tutto se quel client ha raggiunto il numero massimo di tentativi a sua disposizione. In caso affermativo, viene inviato un messaggio di errore. In caso negativo, viene verificato se il numero inviato dal client e' uguale al numero estratto dal server, e viene inviato al client un messaggio tramite cui lo si informa della vittoria o della sconfitta.

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.*