Immagine che contiene testo, screenshot, clipart

Descrizione generata automaticamente

Ingegneria di Internet e Web

Progetto A.A. 2019/2020

UDP-GBN

Massimo Mazzetti

0253467

**INDICE**

[1. Traccia del Progetto 2](#_Toc119632712)

[2. Architettura e scelte progettuali 4](#_Toc119632713)

[3. Implementazione 5](#_Toc119632714)

[4. Limitazioni 6](#_Toc119632715)

[5. Piattaforma utilizzata per sviluppo e testing 7](#_Toc119632716)

[6. Esempi 8](#_Toc119632717)

[7. Valutazione Prestazioni 9](#_Toc119632718)

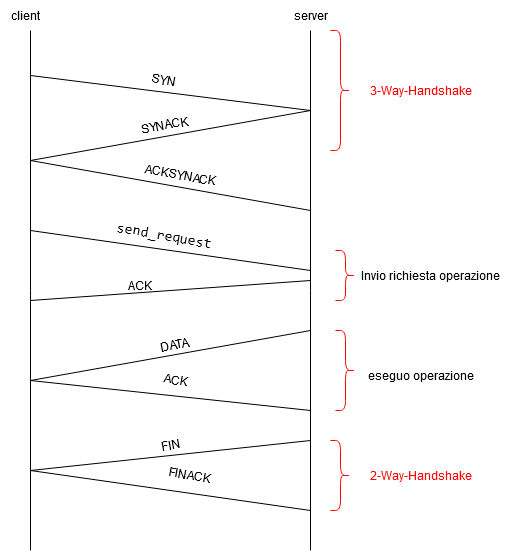
[8. Configurazione, Installazione eD Esecuzione 10](#_Toc119632719)

# Traccia del Progetto

|  |  |
| --- | --- |
|  | **Reti di Calcolatori ed Ingegneria del Web - A.A. 2019/20**  **Progetto B1: Trasferimento file su UDP**  Lo scopo de progetto è quello di progettare ed implementare in linguaggio C usando l’API del socket di Berkeley un’applicazione client-server per il trasferimento di file che impieghi il servizio di rete senza connessione (socket tipo SOCK\_DGRAM, ovvero UDP come protocollo di strato di trasporto). Il software deve permettere:  • Connessione client-server senza autenticazione;  • La visualizzazione sul client dei file disponibili sul server (comando list);  • Il download di un file dal server (comando get);  • L’upload di un file sul server (comando put);  • Il trasferimento file in modo affidabile.  La comunicazione tra client e server deve avvenire tramite un opportuno protocollo. Il protocollo di comunicazione deve prevedere lo scambio di due tipi di messaggi: messaggi di comando: vengono inviati dal client al server per richiedere l’esecuzione delle diverse operazioni; messaggi di risposta: vengono inviati dal server al client in risposta ad un comando con l’esito dell’operazione.  **Funzionalità del server**  Il server, di tipo concorrente, deve fornire le seguenti funzionalità:  • L’invio del messaggio di risposta al comando list al client richiedente; il messaggio di risposta contiene la filelist, ovvero la lista dei nomi dei file disponibili per la condivisione;  • L’invio del messaggio di risposta al comando get contenente il file richiesto, se presente, od un opportuno messaggio di errore;  • La ricezione di un messaggio put contenente il file da caricare sul server e l’invio di un messaggio di risposta con l’esito dell’operazione.  **Funzionalità del client**  I client, di tipo concorrente, deve fornire le seguenti funzionalità:  • L’invio del messaggio list per richiedere la lista dei nomi dei file disponibili;  • L’invio del messaggio get per ottenere un file  • La ricezione di un file richiesta tramite il messaggio di get o la gestione dell’eventuale errore  • L’invio del messaggio put per effettuare l’upload di un file sul server e la ricezione del messaggio di risposta con l’esito dell’operazione.  **Trasmissione affidabile**  Lo scambio di messaggi avviene usando un servizio di comunicazione non affidabile. Al fine di garantire la corretta spedizione/ricezione dei messaggi e dei file sia i client che il server implementano a livello applicativo il protocollo Go-Back N (cfr. Kurose & Ross “Reti di Calcolatori e Internet”, 7° Edizione). Per simulare la perdita dei messaggi in rete (evento alquanto improbabile in una rete locale per non parlare di quando client e server sono eseguiti sullo stesso host), si assume che ogni messaggio sia scartato dal mittente con probabilità p. La dimensione della finestra di spedizione N, la probabilità di perdita dei messaggi p, e la durata del timeout T, sono tre costanti configurabili ed uguali per tutti i processi. Oltre all’uso di un timeout fisso, deve essere possibile scegliere l’uso di un valore per il timeout adattativo calcolato dinamicamente in base alla evoluzione dei ritardi di rete osservati. I client ed il server devono essere eseguiti nello spazio utente senza richiedere privilegi di root. Il server deve essere in ascolto su una porta di default (configurabile). |

# Architettura e scelte progettuali

L'applicazione presenta un'architettura di tipo Client-Server.

Il client e il server utilizzano il protocollo di livello 4 UDP.

La fruizione del servizio avviene in quattro fasi:

* Instaurazione della connessione tramite 3-Way-Handshake.
* Il client richiede che operazione fare (GET, PUT, LIST).
* Il server esegue la richiesta da parte del client in modo affidabile tramite il protocollo Go-Back N.
* Chiusura della connessione tramite il 2-Way-Handshake al termine dell’operazione.

I messaggi scambiati sono di 3 tipi:

* Comandi;
* Dati;
* Risposta.

Si è scelto un server concorrenziale a processi per semplicità sviluppo e manutenibilità in quanto difficilmente si incorrerà nella saturazione delle risorse di sistema; tuttavia, rimane il fatto che essa risulterà più lenta rispetto ad altre soluzioni a causa del cambio di contesto.

A fine di permettere la rilevazione di connessioni morte è stato realizzato un sistema che consente un numero massimo di ritrasmissioni e un tempo massimo per inviare una richiesta da parte del client.

L’Instaurazione avviene, come già detto, tramite 3-Way-Handshake, cioè vengono scambiati 3 messaggi:

* SYN: Inviato dal client al server per richiedere di connettersi;
* ACKSYN: Inviato dal server al client come riscontro del SYN e manda al client un numero di porta che sarà associata alla socket del figlio che servirà il client;
* ACKSYNACK, inviato dal client al server, riscontra il SYNACK e serve al server per avere conferma che il client è ancora presente ed ha ricevuto le informazioni necessarie per l’instaurazione della connessione, così se ciò non è vero il server può deallocare le risorse dedicate a quello specifico client e renderle di nuovo disponibili per altre eventuali richieste di connessione.

La richiesta dell’operazione avviene con 2 messaggi:

* Il client invia un dato contenente quale operazione svolgere:
* Il server invia un ACK per riscontrare quell’operazione e inizia a eseguire l’operazione.

La chiusura della connessione, come già detto, tramite 2-Way-Handshake, cioè vengono scambiati 2 messaggi:

* FIN: inviato da chi invia dati quando finisce l’operazione e conseguente chiusura della connessione;
* ACKFIN: riscontro del FIN.

# Implementazione

I messaggi scambiati tra server e client come visto nella sezione precedente sono di tre tipi (comandi, dati e risposta) e sono stati implementati tramite una struct data\_packet.

|  |
| --- |
| struct data\_packet {      int type;      long seq\_no;      int length;      char data[MTU];  }; |

Essa viene utilizzata per lo scambio di dati, messaggi di comando, messaggi di sincronizzazione e messaggi di riscontro, per lo svolgimento di tale compito possiede quattro attributi:

* type (int), indica il tipo di messaggio; che può essere un comando (PUT, LIST, GET), un dato (NORMAL) oppure un messaggio di sincronizzazione (SYN, FIN).
* seq\_no (long), il quale indica il numero di sequenza del pacchetto, in caso di SYN indica un codice identificativo della richiesta.
* length (int), il quale indica la lunghezza in byte del dato nel campo “data”.
* data (char [MTU], dove MTU vale 1024), il quale contiene il dato che viene scambiato( NORMAL nel caso contiene un dato relativo al tipo di servizio richiesto dal client (chunk di file oppure un nome di file su server che può essere scaricato/caricato dal client).

Per simulare la perdita di messaggi è stata creata la funzione:

|  |
| --- |
| bool simulate\_loss(float loss\_rate)  {    double value;    struct timespec tms;    if (clock\_gettime(CLOCK\_REALTIME, &tms))    {      srand(time(NULL));    }    else    {      int64\_t micros = tms.tv\_sec \* 1000000;      micros += tms.tv\_nsec / 1000;      if (tms.tv\_nsec % 1000 >= 500)      {        ++micros;      }      srand(micros);    }    int r1 = rand() % 100;    int r2 = rand() % 100;    int r3 = rand() % 100;    unsigned short vec[3] = {r1, r2, r3};    seed48(vec);    value = drand48();    if (value < loss\_rate){      return true;    }    return false;  } |

La funzione prende in ingresso un parametro float relativo alla probabilità di perdita inserita dall'utente, ed in base ad esso, attraverso un procedimento di generazione di valori randomici per l'impostazione del seed della funzione drand48() della libreria stdlib.h, produce un valore float pseudorandomico nell'intervallo [0, 1). In base al valore ottenuto la funzione ritornerà un booleano per indicare se debba essere applicato o meno l'evento di perdita.

# Limitazioni

# Piattaforma utilizzata per sviluppo e testing

Lo sviluppo è avvenuto in ambiente Unix in particolare si è utilizzata una **WSL2** (Windows Subsystem for Linux) su cui è installata la distribuzione “**Ubuntu 20.04.4 LTS**”.

La versione del Kernel è “**5.10.60.1-microsoft-standard-WSL2**”.

Per la compilazione è stato utilizzato “**gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0**”.

Per la scrittura del codice è stato utilizzato il text editor “**Visual Studio Code**”.

# Esempi

# Valutazione Prestazioni

|  |  |  |  |
| --- | --- | --- | --- |
| Perdita | Finestra | Timer [ms] | Tempo esecuzione [ms] |
| 0 |  |  |  |
| 0.2 |  |  |  |
| 0.5 |  |  |  |
| 0.8 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Configurazione, Installazione ed Esecuzione