Relatório Socket UDP

Victor de Oliveira Carmona 073805
Douglas A. S. Costa 104825

Introdução

Fizemos uma mini biblioteca atrelada a um mini banco de dados com algumas funções baseada no ISBN de cada livro. As funções são:

1 - Listar todos os ISBN com os títulos ;
2 – Descrição do livro a partir do ISBN ;
3 - Todas as informações do livro a partir do ISBN ;
4 - Lista todas as informações de todos os livros ;
5 - Altera o número de exemplares em estoque ;
6 - Número de exemplares em estoque a partir do ISBN .

Ambiente de implementação

Fizemos isso na linguagem C usando as bibliotecas apropriadas. Quem faz todas funções é o servidor UDP da biblioteca representado aqui pelo arquivo “server.c”, e quem faz as requisições de listagem e de mudanças é o cliente, representado aqui pelo arquivo “client.c”.
O client é bem simples, apenas imprime as opções, pede a opção desejada pelo cliente e as informações necessárias para cada caso e as envia separado por traço para o servidor. Em sua inicialização é passado o ip da máquina em que está o servidor está rodando.
O servidor já é um pouco mais complexo, além de interpretar as informações vindas do client, ele ainda faz abre um arquivo mantido como banco de dados e busca as informações nele, assim como altera também. Depois ainda retorna essas informações para o client.
Para a realização das medidas de tempo , foi utilizada a função :gettimeofday(struct timeval *tp , struct timezone *tzp) , da biblioteca <sys/time.h>. Medimos os tempos de cada opção de consulta 50 vezes e tiramos as médias com seus devidos desvios padrões.

Vantagens

Podemos notar que as vantagens são que o UDP é um serviço sem conexão, não mantendo um relacionamento longo com o servidor. Ele apenas monta datagramas com as informações necessárias (ip fonte e ip destino, porta fonte e porta destino, além dos dados e outras informações adicionais) e as envia. Comparado ao UTP ele é menos confiável, pois não possui um algoritmo mais complexo para garantir que os dados chegaram consistentes no destino, porém isso lhe dá a vantagem de ser mais rápido.

Código Fonte

Código fonte do client.c

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>

char sendline[1000];
char recvline[1000];
char opcao[1];
char isbn[10];
struct timeval *tv1, *tv2;

int main(int argc, char**argv) {
	int sockfd, n;
	struct sockaddr_in servaddr, cliaddr;

	if (argc != 2) {
		printf("Passe o ip como parâmetro\n");
		exit(1);
	}

	sockfd = socket(AF_INET, SOCK_DGRAM, 0);

	bzero(&servaddr, sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = inet_addr(argv[1]);
	servaddr.sin_port = htons(32000);

	while (1) {

		menu();

		scanf("%s", &opcao);

		int i;
		for (i = 0; i < 50; ++i) {

			verificaOpcao();

			printf("%s\n", sendline);
```

```
                        tv1 = malloc(sizeof(struct timeval));
                        tv2 = malloc(sizeof(struct timeval));

                        gettimeofday(tv1, NULL );

                        sendto(sockfd, sendline, strlen(sendline), 0,
                                        (struct sockaddr *) &servaddr, sizeof(servaddr));
                        n = recvfrom(sockfd, recvline, 10000, 0, NULL, NULL );

                        gettimeofday(tv2, NULL );

                        suseconds_t time = (*tv2).tv_usec - (*tv1).tv_usec;
                        //printf("i: %d\n", i);
                        printf("Tempo total %d : %d\n", i,time);

                        recvline[n] = 0;
                        printf("%s\n\n\n", recvline);
                }
        }
}

void menu() {
//imprime o menu para o cliente
        printf("---------- Livros----------\n\n");
        printf(" 1 - Lista todos os ISBN com os títulos\n");
        printf(" 2 - Descrição a partir do ISBN\n");
        printf(" 3 - Todas as informações a partir do ISBN\n");
        printf(" 4 - Lista todas as informações de todos os livros\n");
        printf(" 5 - Altera o número de exemplares em estoque\n");
        printf(" 6 - Número de exemplares em estoque a partir do ISBN\n");
        printf(" 7 - Sair\n");

        printf("Escolha uma opção: ");

}

void verificaOpcao() {

        strcpy(sendline, "");
        strcat(sendline, opcao);

        int op = strcmp(opcao, "0");

        if (op == 7) {
                printf("Programa encerrado...\n");
                exit(1);
```

```
	} else if (op == 2 || op == 3 || op == 5 || op == 6) {

		//printf("Digite ISBN: ");

		//isbn = malloc(10*sizeof(char));
		strcpy(isbn, "01");

		//scanf("%10s", isbn);

		strcat(sendline, "-");
		strcat(sendline, isbn);

		if (op == 5) {
			printf("Digite nova quantidade: ");

			//qde = malloc(10*sizeof(char));
			char qde[4];
			strcpy(qde, "25");
			//scanf("%4s", qde);
			strcat(sendline, "-");
			strcat(sendline, qde);
		}
	} else if ((op == 1) || (op == 4)) {
		//continua
	} else {
		printf("Opção inválida");
	}

}
```

]

Código Fonte do server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/time.h>
#include <time.h>

#define PORT "3490"  // the port users will be connecting to

#define BACKLOG 10     // how many pending connections queue will hold

#define BUFSIZE 20

#define MAXDATASIZE 100 // max number of bytes we can get at once

#define MAX 100

#define ARQ 4

typedef struct Livro {
        char isbn[13];
        char titulo[100];
        char descricao[300];
        char autores[50];
        char editora[30];
        char ano[5];
        char quant[3];
} Serv_livro;

Serv_livro le_banco(FILE* bd);
void lista_tudo(char *buf, Serv_livro bd[], char *quant);
void avaliaOpcao();
void responde(char *buf, int menu, Serv_livro livros[], char *quant,char *isbn);
```

```
void closeBD(FILE *bd);
FILE* openBD();
void zeraBuffer(char *buf);
void imprime(int col, int lin, Serv_livro bd[], char *buf);
void lista_titulo(char *buf, Serv_livro bd[], char *quant);
void lista_menu(char *buf, Serv_livro bd[], char *quant, char ISBN[], int ret);
void altera_bd(char *buf, Serv_livro bd[], char *quant, char ISBN[]);
void popula_banco(Serv_livro livros[]);

struct timeval *tv1, *tv2;
char mesg[1000];



Serv_livro livros[MAX];/*vetor contendo os livros*/

int main(int argc, char**argv) {
	int sockfd, n;
	struct sockaddr_in servaddr, cliaddr;
	socklen_t len;

	sockfd = socket(AF_INET, SOCK_DGRAM, 0);

	bzero(&servaddr, sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = htonl(INADDR_ANY );
	servaddr.sin_port = htons(32000);
	bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

	for (;;) {
		len = sizeof(cliaddr);
		n = recvfrom(sockfd, mesg, 1000, 0, (struct sockaddr *) &cliaddr, &len);

		mesg[n] = 0;
		printf("Recebido:\n");
		printf("%s\n\n", mesg);

		tv1 = malloc(sizeof(struct timeval));
		tv2 = malloc(sizeof(struct timeval));

		avaliaOpcao();

		suseconds_t time = (*tv2).tv_usec - (*tv1).tv_usec;
		//printf("i: %d\n", i);
		printf("Tempo proc.: %ld\n", time);
```

```
                sendto(sockfd, mesg, strlen(mesg), 0, (struct sockaddr *) &cliaddr,
                        sizeof(cliaddr));

        }
}

void avaliaOpcao() {

        char opcao[1];
        char *isbn;
        char *qde;
        int qdet;
        int achou = 0;
        int i, j;
        FILE *bd;

        gettimeofday(tv1, NULL );
        //pega o menu do cliente
        opcao[0] = mesg[0];

        int op = strcmp(opcao, "0");
        // Se for essas opcoes recebe o ISBN para fazer a consulta
        if (op == 2 || op == 3 || op == 5 || op == 6) {

                isbn = malloc(13 * sizeof(char));

                for (i = 0; i < 13; i++) {
                        if (mesg[i + 2] == '-')
                                break;
                        isbn[i] = mesg[i + 2];
                }
        }
 //Se for 5 recebe também a nova quantidade
        if (op == 5) {

                qde = malloc(10 * sizeof(char));
                j = 0;
                for (i = 2; i < strlen(mesg); i++) {
                        if (achou == 0) {
                                if (mesg[i] != '-' && achou == 0)
                                        continue;
                                else {
                                        achou = 1;
                                        continue;
                                }
                        }
```

```
                        qde[j] = mesg[i];
                        j++;
                }

                qdet = atoi(qde);
        }


        strcpy(mesg, "\0");
        strcat(mesg, "Opção: ");
        strcat(mesg, opcao);

        if (isbn != NULL ) {
                strcat(mesg, " ");
                strcat(mesg, "Isbn: ");
                strcat(mesg, isbn);
        }
        if (qde != NULL ) {
                strcat(mesg, " ");
                strcat(mesg, "Qde: ");
                strcat(mesg, qde);
        }
        printf("%s\n",mesg);
        Serv_livro livros[ARQ];
        bd = openBD();
        *livros = le_banco(bd);
        int men = opcao[0] - '0';
        responde(mesg, men, livros, qde, isbn);

        gettimeofday(tv2, NULL );
}

//funcao que analisa a entrada do cliente
void responde(char *buf, int menu, Serv_livro *bd, char *quant,char *isbn){
  char parametro_id[2], c;
  int indice;
                int i = 0;
                int files;
                int count =1,mod;
  switch(menu){
   case 1:/*listar titulos e ISBN*/
                                                        lista_titulo(buf, bd,quant);
                                                        //closeBD(bd);
    break;
  case 2:/*Descrição a partir do ISBN*/
                lista_menu(buf, bd,quant,isbn,3);
```

```
                    //closeBD(bd);
    break;
  case 3:/*Todas as informações a partir do ISBN*/
                    lista_menu(buf, bd,quant,isbn,0);
                    //closeBD(bd);
    break;
  case 4:/*Lista todas as informações de todos os livros*/
                    lista_tudo(buf,bd,quant);
                    //closeBD(bd);
    break;
  case 5:/*Altera o número de exemplares em estoque*/
                    altera_bd(buf ,bd,quant,isbn);
    break;
  case 6:/*Número de exemplares em estoque a partir do ISBN*/
                    lista_menu(buf, bd,quant,isbn,7);
                    //closeBD(bd);
    break;
  case 7:/*encerra*/
    //strcpy(saida,"Encerra");
    break;

  }
          return;

}
//abre o banco de dados
FILE* openBD() {

          FILE *fp;
          fp = fopen("bd.txt", "r");
          if (fp == NULL) {
                    printf("Problema ao abrir o arquivo\n");
          } else {
                    //printf("Arquivo aberto com sucesso\n");
                    return fp;
          }
}

void closeBD(FILE *bd) {
          fseek(bd, 0, SEEK_SET);
}

void zeraBuffer(char *buf){
  int i;
  for(i=0;i<MAXDATASIZE;i++)
    buf[i] = '\0';
```

```
}
	//funcao genérica que imprime o campo desejado
	void imprime(int col, int lin, Serv_livro bd[], char *buf){
	int indice;
	char c;
	int i , j, valida =0, k;
	int files;
	int count =0;
	char *result;

		if (col== 1) {
			strcat(buf, "ISBN -");
			strcat(buf, livros[lin].isbn);
			printf("ISBN: %s\n", livros[lin].isbn);
			}
		if (col == 2) {
			strcat(buf, "Titulo -");
			strcat(buf, livros[lin].titulo);
			printf("Titulo: %s\n", livros[lin].titulo);
			}
		if (col == 3) {
			strcat(buf, "Descricao -");
			strcat(buf, livros[lin].descricao);
			printf("Descricao: %s\n", livros[lin].descricao);
			}
		if (col == 4) {
			strcat(buf, "Autores -");
			strcat(buf, livros[lin].autores);
			printf("Autores: %s\n", livros[lin].autores);
			}
		if (col == 5) {
			strcat(buf, "Editora -");
			strcat(buf, livros[lin].editora);
			printf("Editora: %s\n", livros[lin].editora);
			}
		if (col == 6) {
			strcat(buf, "Ano -");
			strcat(buf, livros[lin].ano);
			printf("Ano: %s\n", livros[lin].ano);
			}
		if (col == 7) {
			strcat(buf, "Estoque -");
			strcat(buf, livros[lin].quant);
			printf("Estoque: %s\n", livros[lin].quant);
			}
```

```
}

//Faz leitura do banco de dados e coloca em um struct para facilitar a leitura
Serv_livro le_banco(FILE *bd){

		int campo;
		int i,j,k=0;
		char c = ' ';
		char	linha[500];
		while (!feof(bd)){//enquanto tiver arquivo
			for (j=0;j<4;j++){//para todas as linhas até o final do vetor de struct
				campo=1;

				fgets(linha,500,bd);
				for (i=0;i<500;i++){//percorre a linha inteira
					c = linha[i];//le o caractere
				if(c!=';'){
					if (campo == 1){
					livros[j].isbn[k++] = c;
					}
					if(campo == 2){
					livros[j].titulo[k++] = c;
					}
					if(campo == 3){
					livros[j].descricao[k++] = c;
					}
					if(campo == 4){
					livros[j].autores[k++] = c;
					}
					if(campo == 5){
					livros[j].editora[k++] = c;
					}
					if(campo == 6){
					livros[j].ano[k++] = c;
					}
					if(campo == 7){
					livros[j].quant[k++] = c;
					}
				}else{
						campo++;
						k=0;
				}
				}

			}
		}
```

```
	fclose(bd);
	return *livros;

	}

// imprime todos os titulos do banco
void lista_titulo(char *buf, Serv_livro bd[], char *quant){
  int indice;
    //strcpy(parametro_id,buffer+2);
	char c;
	int i = 0;
	int files;
	int count =1;
	int j;
	for (j=0;j<ARQ;j++){//para todas as linhas
		imprime(1, j, bd, buf);
		imprime(2, j, bd, buf);
	}
}
//dependendo da entrada ele imprime o desejado
void lista_menu(char *buf, Serv_livro bd[], char *quant, char ISBN[], int ret){

 int indice;
    //strcpy(parametro_id,buffer+2);
	char c;
	int i,j,k, valida =0;
	int files, aux;
	int count =0,mod;
	char linha[100];
	char *result;
		for (j=0;j<ARQ;j++){//para todas as linhas
				printf("Banco: %s",bd[j].isbn);
				printf("Passado como parametro: %s\n",ISBN);
			if ( !strcmp(ISBN,bd[j].isbn) ){//se for igual ao procurado

				if (ret == 3) {
					imprime(1, j, bd, buf);
					imprime(3, j, bd, buf);
					return;
				}
				if (ret == 7) {
					imprime(1,j,bd, buf);
					imprime(7,j,bd, buf);
					return;
				}
				if (ret == 0){
```

```
                                        imprime(1,j,bd, buf);
                                        imprime(2,j,bd, buf);
                                        imprime(3,j,bd, buf);
                                        imprime(4,j,bd, buf);
                                        imprime(5,j,bd, buf);
                                        imprime(6,j,bd, buf);
                                        imprime(7,j,bd, buf);
                                        return;
                                }
                        }
                }
                strcat(buf, "Não achou ISBN");
                printf("Não achou ISBN\n");
                return;
}

//lista todos os livros do banco
void lista_tudo(char *buf, Serv_livro bd[], char *quant){
  int indice;
    //strcpy(parametro_id,buffer+2);
        char c;
        int i,j,k, valida =0;
        int files;
        int count =0,mod;
        char linha[100];
        char *result;
                for (j=0;j<ARQ;j++){//para todas as linhas
                                imprime(1,j,bd,buf);
                                imprime(2,j,bd,buf);
                                imprime(3,j,bd,buf);
                                imprime(4,j,bd,buf);
                                imprime(5,j,bd,buf);
                                imprime(6,j,bd,buf);
                                imprime(7,j,bd,buf);
                        }
                        return;
}

        //altera o valor do estoque de um determinado isbn
void altera_bd(char *buf, Serv_livro bd[], char *quant, char ISBN[]){

        int indice;
        char c;
        int i,j,k, valida =0;
        int files;
        int count =0,mod;
```

```
	char linha[100];
	char *result;
		for (j=0;j<ARQ;j++){//para todas as linhas
					if ( !strcmp(ISBN,bd[j].isbn) ){//se for igual ao procurado
					//troca o valor de estoque
					strcpy(bd[j].quant,quant);
				//grava no arquivo todo o struct
				//popula_banco(bd);
				lista_tudo(buf, bd,quant);
				return;
				}
		}
		printf("Não alterou o banco");
		return;

	}
		// reescreve o banco de dados atualizando o estoque
	void popula_banco(Serv_livro livros[]){
	FILE *fp;
	fp = fopen("bd.txt", "w");
	if (fp == NULL) {
		printf("Problema ao abrir o arquivo\n");
	} else {
		printf("Arquivo aberto com sucesso\n");
	}
	int i;
  if (!fp){
	printf("Erro na abertura do arquivo");
		exit(0);
  }else{
		for(i=0;i<ARQ;i++){

			fprintf(fp, "%s;",livros[i].isbn);
			fprintf(fp, "%s;",livros[i].titulo);
			fprintf(fp, "%s;",livros[i].descricao);
			fprintf(fp, "%s;",livros[i].autores);
			fprintf(fp, "%s;",livros[i].editora);
			fprintf(fp, "%s;",livros[i].ano);
			fprintf(fp, "%s;",livros[i].quant);
			if (i!=ARQ-1) fprintf(fp, "\n",livros[i].quant);
		}
	}
}
```

Medidas de tempo

A primeira análise é sobre o tempo de comunicação do cliente com o servidor. Para isso realizamos uma medida do tempo absoluto com o metodo *gettimeofday* antes do cliente enviar uma mensagem para o servidor, e depois do recebimento da resposta, então subtraímos o último do primeiro e temos o tempo de comunicação.

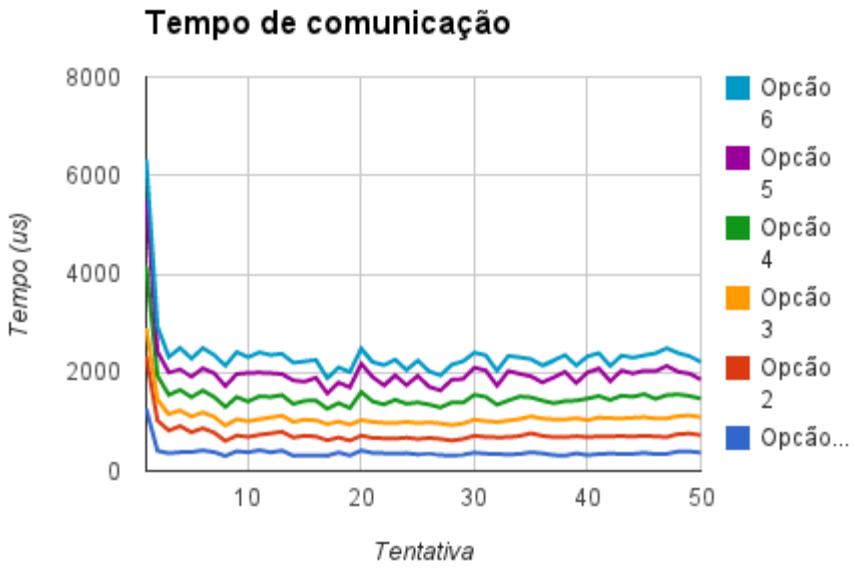
Tabela com o tempo de 50 tentativas de comunicação:

Tempo de Comunicação (us)

	Opcão 1	Opcão 2	Opcão 3	Opcão 4	Opcão 5
1	1259	1067	575	1269	1344
2	410	616	427	480	488
3	360	461	333	391	455
4	381	530	319	416	426
5	384	402	317	395	415
6	418	449	324	444	450
7	383	397	320	404	481
8	304	304	313	379	424
9	401	321	328	453	466
10	378	313	319	399	581
11	423	312	315	468	479
12	378	385	324	416	484
13	414	384	326	417	427
14	309	371	316	364	478
15	315	401	326	383	388
16	319	381	327	404	461
17	309	314	329	309	315
18	376	305	321	376	417
19	314	302	325	338	420
20	416	302	315	571	573
21	366	304	327	415	497

22	360	303	315	373	390
23	355	302	315	474	498
24	361	316	319	369	377
25	335	314	321	421	541
26	360	310	318	360	361
27	323	328	314	324	346
28	310	307	315	460	460
29	330	320	314	423	480
30	372	344	327	505	549
31	351	340	320	494	537
32	343	341	311	350	386
33	336	350	336	410	596
34	343	367	346	456	461
35	380	386	342	387	426
36	358	355	351	365	370
37	322	366	360	326	525
38	311	374	360	373	596
39	359	348	363	359	359
40	321	366	345	439	531
41	345	358	382	437	560
42	353	350	369	370	381
43	347	361	359	462	510
44	347	353	380	427	474
45	368	347	379	466	475
46	342	359	362	401	562
47	348	342	378	472	594
48	395	354	365	440	461
49	396	364	367	396	454
50	374	353	366	376	390

Desvio Padrão	131.5	115.8	42.0	131.8	143.1	80.3



Para medirmos o tempo de processamento, usamos o mesmo procedimento usado no anterior, porém medindo o tempo entre o momento em que o servidor recebe a mensagem de requisição e começa a processar a resposta, e o tempo em que termina de processar, logo antes enviar a resposta.

Tabela com o tempo de 50 processamentos:

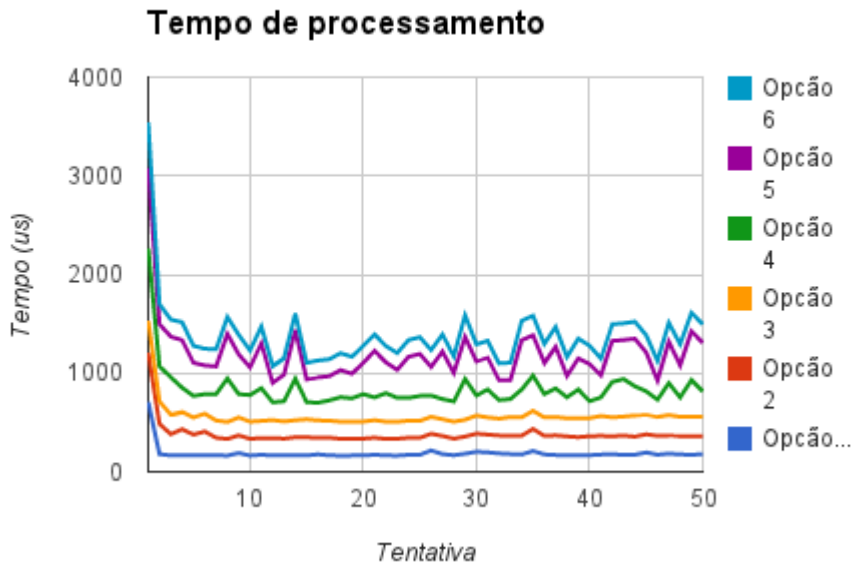
Tempo de processamento (us)

	Opção 1	Opção 2	Opção 3	Opção 4	Opção 5	Opção 6
1	709	500	323	734	813	463
2	179	306	227	357	427	203
3	166	215	198	379	412	173
4	170	261	175	246	482	182
5	166	211	173	219	336	172
6	166	242	181	198	291	169
7	167	178	175	267	281	175
8	164	169	172	441	451	172

9	190	179	182	236	402	199
10	164	170	174	270	281	171
11	173	169	174	332	457	168
12	165	177	182	180	198	172
13	166	170	174	205	269	168
14	166	184	173	420	492	168
15	166	182	184	174	231	168
16	177	170	174	179	257	171
17	166	180	172	206	245	172
18	164	170	174	248	275	171
19	165	169	175	236	249	172
20	165	171	173	278	317	172
21	173	172	178	231	472	167
22	166	168	174	289	313	165
23	164	171	172	243	283	171
24	172	173	173	231	417	169
25	172	172	172	253	426	170
26	216	169	172	216	290	172
27	179	184	170	204	482	173
28	168	168	172	205	289	169
29	183	174	171	413	425	220
30	203	183	184	204	345	173
31	197	180	174	282	327	168
32	186	182	171	188	200	172
33	180	185	190	182	192	177
34	175	191	186	282	500	197
35	211	223	184	357	409	198
36	176	188	187	237	317	192
37	170	199	186	290	423	200
38	167	192	184	212	217	192
39	168	183	190	293	314	202
40	168	193	183	171	376	187
41	175	189	202	188	224	170
42	176	184	192	359	416	169
43	174	193	193	380	396	169
44	17	183	214	297	482	173

45	196	184	201	235	394	175
46	173	191	192	175	194	200
47	184	189	202	324	419	192
48	176	185	196	197	326	213
49	173	189	196	372	495	187
50	178	184	194	258	492	188

Desvio Padrão	76.5	50.7	23.3	98.0	115.0	42.4



Conclusão

Poderemos comparar os tempos do TCP e UDP considerando algumas ressalvas:

- O banco de dados usado no UDP era maior, deixando o tempo de processamento e de comunicação ligeiramente maiores.
- A máquina do servidor UDP era mais fraca que a do servidor TCP. Podemos confirmar isso nos tempos de processamento, são maiores no no UDP para realizar as mesmas funções;

- A rede em que foi realizado o teste com o UDP era uma rede caseira, enquanto a rede dos testes de TCP foi uma rede institucional, então provavelmente a rede institucional tem uma velocidade maior (horário não congestionado). Podemos confirmar isso com os maiores tempos de comunicação do client com o servidor e também na maior variação entre eles, visto no desvio padrão e nos gráficos;

Apesar de todas as probabilidades serem a favor do TCP e contra o UDP, os tempos do UDP foram pouca coisa maior, considerando a grandeza dos tempos e que podem conter erros na medida, a diferença foi muito pequena. Também como a quantidade de pacotes é pequena, o número de pacotes adicionais que o TCP troca para o estabelecimento das conexões se torna um número considerável. Isso confirma a maior velocidade do UDP, mesmo com condições mais adversas.
Quanto a confiabilidade de transferência não podemos julgar pois não houve perda visível de pacotes. Talvez num teste mais apurado com condições mais adversas poderíamos ter um parâmetro melhor para comparação, por exemplo comunicando-se com um servidor que não está na mesma rede, num horário congestionado e o cliente num wireless público.

Referências Bibliográficas

http://www.cplusplus.com
http://www.cs.ucsb.edu/~almeroth/classes/W01.176B/hw2/examples/