Universidade Estadual de Campinas

MC833

Edmundo Madeira

Uso de Sockets para comunicação UDP

Autores:	*RA:*
Nathália Harumi	175188
Túlio Martins	177761

26 de Abril de 2018

1 Introdução

Neste projeto, foi explorado o uso e a eficiência da comunicação entre duas máquinas via sockets que se comunicam pelo protocolo UDP. Para a implementação dos sockets foram usadas múltiplas bibliotecas da "sys/*"para sockets, para o uso do "select"e para a captação de tempo, além disso foram usadas as bibliotecas da arpa/inet pra estabelecimento de conexão e resolvimento de nomes . Para a verificação da eficiência do UDP foram medidos os tempos através de funções da biblioteca "sys/time.h". O programa utilizado para fazer tais medições é uma simulação de um banco de dados de disciplinas com acesso às informações das mesmas . O objetivo do sistema é enviar múltiplos datagramas diferentes entre cliente e servidor pra medir o tempo de envio dos mesmos no protocolo UDP em uma rede LAN entre duas máquinas diferentes. Com isto, outro objetivo também é comparar os tempos obtidos neste experimento com os obtidos no protocolo TCP e fazer uma comparação razoável.

2 Sistema: descrição geral e casos de uso

2.1 Descrição geral do sistema

O sistema implementado consiste de dois agentes: o servidor e o cliente. O servidor contém todas as informações das disciplinas, a implementação das estruturas de dados e possui um socket ouvindo uma porta em protocolo UDP. Por ser UDP o servidor naturalmente aguenta múltiplos clientes, uma vz que não existe um sistema de handshake, entretanto o teste foi feito somente com um cliente ativo.

Já o cliente possui todas as chamadas possíveis que o servidor suporta, a mensagem enviada para o servidor é um datagrama que é padrão para todas as mensagens e a mesma é então traduzida somente no lado do servidor. A única função do cliente após enviar requisição é imprimir a resposta do servidor que também é um datagrama padronizado independente da função e de uma resposta bem sucedida ou não, em caso de timeout atingido(ou seja, o servidor não enviou a resposta a tempo) o cliente imprime uma mensagem de timeout e pode tentar novamente.

O cliente consiste de apenas um socket fazendo requisições que serão tratados por um único socket no lado do servidor. O mesmo recebe mensagens únicas não estruturadas, ou seja, ele não está ciente da estrutura de dados e como eles são armazenados no lado do servidor, somente o tamanho da mensagem recebida é de conhecimento do cliente.

Para mantermos a persistência nos dados foram usados arquivos. Como apenas os comentários da última aula são sobrescritos, ou seja, são alterados por clientes, apenas os mesmos estão guardados em memória permanente em disco. Esta memória permanente reside no diretório raiz do programa do servidor e o comentário da disciplina está escrito em um arquivo com o nome da mesma disciplina.

2.2 Casos de uso

Para implementação e teste do sistema foram levados em considerações os seguintes casos de uso:

1. Um cliente pede ao sistema listar todas as disciplinas.
2. Um cliente pede para o sistema todas as informações de uma disciplina que o mesmo escolheu pelo id.
3. Um cliente pede a ementa de uma disciplina para o sistema, dado que o aluno forneceu o id da disciplina.
4. Um cliente pede todas as informações de todas as disciplinas ao sistema.
5. Um cliente professor que leciona uma dada disciplina pede requisição para escrever um comentário sobre a próxima aula desta disciplina

6. Um cliente fornece um código de uma disciplina e recebe o comentário sobre a última aula desta disciplina.

Portanto, baseado nestes casos de uso foram criadas as funcionalidades do sistema servidor e do cliente. O cliente tem estas 6 opções para escolher e sabe como deve ser formatada sua mensagem para entendimento do servidor, que então só processa a requisição e envia um outro datagrama de resposta. Os testes foram realizados entre duas máquinas diferentes em LAN.

3 Armazenamento e Estrutura de Dados do Servidor

O sistema armazena todos os dados da disciplina em uma struct definida especificamente pelo lado do servidor, ou seja em memória principal no "runtime"da aplicação, do qual o cliente não necessita de conhecer. Esta struct pode ser definida conforme a figura a seguir. Como visto, a única informação guardada em memória permanente (disco) é o comentário sobre a última aula. Na struct ele é definido como um ponteiro para o arquivo de nome igual ao da disciplina.

```
typedef struct {
  char id[6];             /*Formato : MCXXX\0*/

  char titulo[LINESIZE];  /*Formato : linha de texto*/
  char ementa[TEXTSIZE];  /*Formato : texto*/
  char sala_de_aula[5];   /*Formato : CC02\0 (?)*/
  char horario[LINESIZE]; /*Formato : DIA_DA_SEMANA HH:mm a
                                  HH:mm;DIA_DA_SEMANA...\0*/

  char comentario_ultima_aula[TEXTSIZE]; /*Formato : Nome_do_arquivo*/

  char usuario[LINESIZE];
  char senha[LINESIZE];

} Disciplina;
```

Figura 1: Configuração da struct da Disciplina.

O funcionamento das informações nas structs é simples. As mesmas são armazenadas como variáveis locais com os 8 campos descritos conforma a figura. Existem 10 disciplinas armazenadas no banco, sendo elas MC833; MC102; MC536; MC750; MC358; MC458; MC558; MC658; MC346; MC886. Cada disciplina possui intrinsecamente o "id"que age como identificador para encontrar a disciplina, e os campos "titulo", "ementa", "sala_de_aula"e "horário"são strings que descrevem conforme o nome indica. O campo de "comentario_ultima_aula"indica o nome do arquivo que possui o comentário. Os últimos dois campos são strings usadas para autenticação do professor que irá escrever o comentário.

O arquivo é armazenado no mesmo diretório do servidor. Toda vez que um professor deseja escrever um comentário novo sobre a próxima aula o mesmo sobrescreve o comentário anterior. Os arquivos são gerados automaticamente caso ainda não exista nenhum comentário. Se nenhum comentário foi escrito ainda, é enviada para o cliente que o conteúdo do comentário é "N/A".

4 Detalhes de implementação

Para a implementação deste serviço foram utilizadas mensagens de tamanhos padrões, isto é, o cliente sabe o tamanho da mensagem que espera do servidor em qualquer momento do processo. Somente uma mensagem é enviada e uma é recebida por função em cada lado do processo, ou seja o cliente envia uma requisição e recebe a resposta, enquanto no lado do servidor o mesmo recebe a requisição e envia a resposta. Essa tomada de decisão foi feita baseada no formato das funções "send"e "receive"das bibliotecas de programação em socket utilizadas. Como independente de como as mensagens seriam enviadas o tamanho delas deveria ser conhecida foi optado por essa abordagem.

Outra decisão importante foi a mudança do número de mensagens enviadas. No servidor em TCP uma única função enviava múltiplas mensagens de tamanhos menores. Devido a não garantia de envio das mensagens para garantir melhor consistência do sistema em UDP foi decidido que todas as funções enviariam apenas uma mensagem contendo todas as informações necessárias. Assim o servidor possui somente um estado, que é esperando por requisição, e a ocorrência de um timeout é tratada simplesmente por tentar enviar a requisição de novo caso fosse necessário.

Como a concorrência não foi trabalhada nesse experimento os arquivos poderiam ser substituídos por memória principal em "runtime", entretanto, para manter consistência com o servidor TCP, foi mantido o uso de arquivos. Além disso os arquivos permitem manter consistência de dados para diferentes iterações do servidor. Assim, caso o servidor fosse reiniciado os comentários se manteriam persistentes.

Foi utilizada função "select"que está inclusa na biblioteca "sys/select"para computar o timeout quando ocorresse perda de mensagem. Para que ela funcione o cliente precisa inicializar os sockets que esperam leitura e os adiconar em um conjunto propriamente definido pela biblioteca. A função "select"então causará um bloqueio no "receive"se o timeout for atingido antes de existir uma mensagem para ser lida no socket.

Durante a implementação foi decidido também utilizar as funções em "time.h"e "sys/time.h"para obtenção das medidas de tempo com maior precisão possível sem uso de bibliotecas fora do system. Assim, para cada uma das operações foi medido o tempo total das mesmas, e dependendo do número de "send"e "receive"utilizados pelos sockets foi realizado uma media para obtenção dos tempos das mensagens.

5 Gráficos e Interpretações

A partir das medições foram obtidos os seguintes tempos médios para cada operação como enumerada na seção de Casos de Uso pela visão do servidor:

1. $$t = (205,41 \pm 42)\mu s$$
2. $$t = (189,55 \pm 40)\mu s$$
3. $$t = (147,99 \pm 19)\mu s$$
4. $$t = (149,01 \pm 29)\mu s$$
5. $$t = (171,77 \pm 84)\mu s$$
6. $$t = (176,50 \pm 64)\mu s$$

Com isto podemos ver que naturalmente o UDP é um protocolo rápido. Houve pouco desvio entre os tempos entre as funções e, como não há necessidade de reenvio de mensagens ou de ordenação das mesmas, isto era esperado.

Assim, a partir destes tempos foi possível fazer um gráfico com todos os tempos de envio de mensagens e recepções para que fosse obtida uma média de quanto tempo leva para uma mensagem sair do cliente e ir para o servidor e vice-versa. Assim o gráfico abaixo representa o tempo de envio destas mensagens e o tempo médio e o seu desvio padrão são dados por :

$$t = 171 \pm 54\mu s$$

que estão dentro do intervalo de confiança esperado.

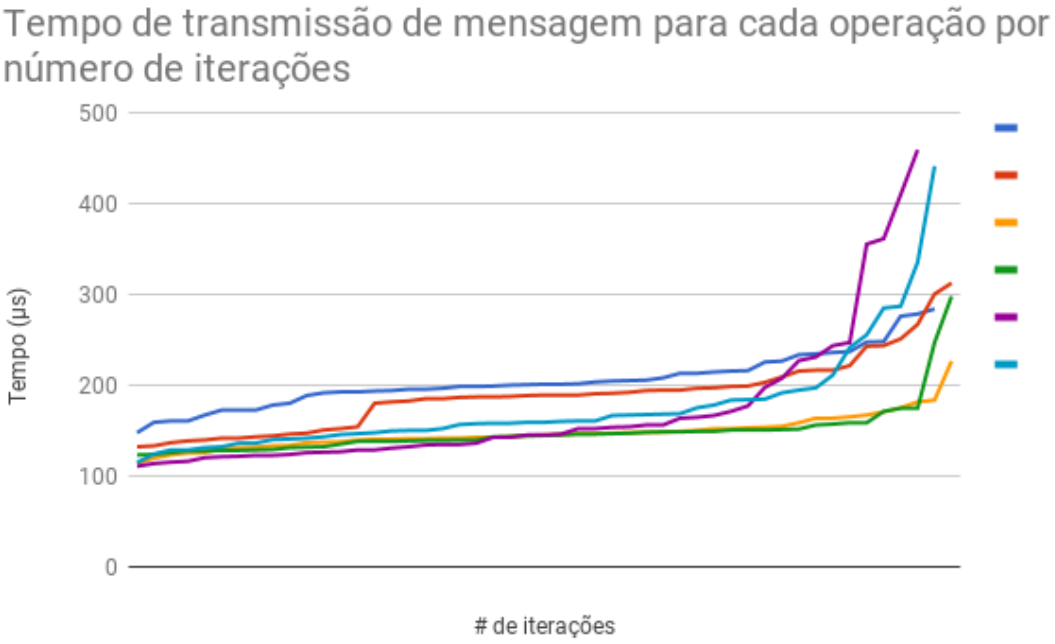

Figura 2: Gráfico do tempo(eixo y) por iteração(eixo x) da conexão cliente-servidor.Legenda: Em azul escuro, listar disciplinas; em vermelho, apresentar ementa da disciplina; em laranja, informação da disciplina; em verde, todas as informações de todas as disciplinas; em roxo, escrever comentário; em azul claro, apresentar comentário sobre a próxima aula.

Ademais, foi possível obter o tempo que o servidor demora para realizar cada operação, desde a requisição do cliente até a obtenção da mensagem esperada como resposta. Esses dados são visíveis em ambos gráficos abaixo:

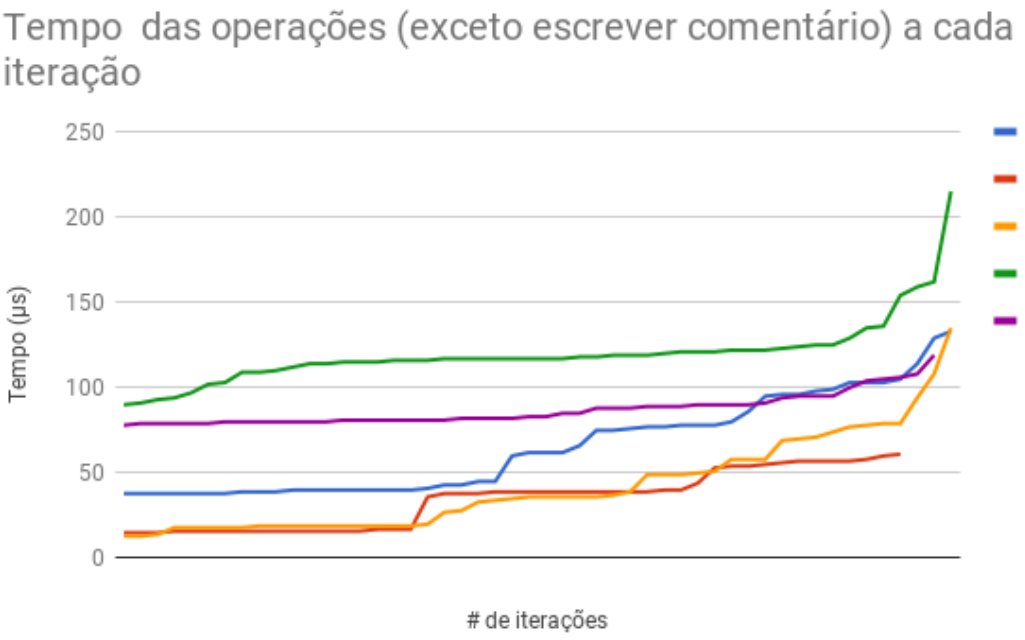

Figura 3: Gráfico do tempo(eixo y) por iteração(eixo x) de cada operação (exceto a que escreve comentário). Legenda: Em azul, listar disciplinas; em vermelho, apresentar ementa da disciplina; em laranja, informação da disciplina; em verde, todas as informações de todas as disciplinas; em roxo, apresentar comentário sobre a próxima aula.

Figura 4: Gráfico do tempo(eixo y) de escrever comentário por iteração(eixo x).

Por último foram observadas as ocorrências de timeout. Considerando que foram realizadas 50 medidas para cada uma das 6 funções tivemos no total 300 medidas. Destas, apenas 4 resultaram em timeout, como representa o gráfico.

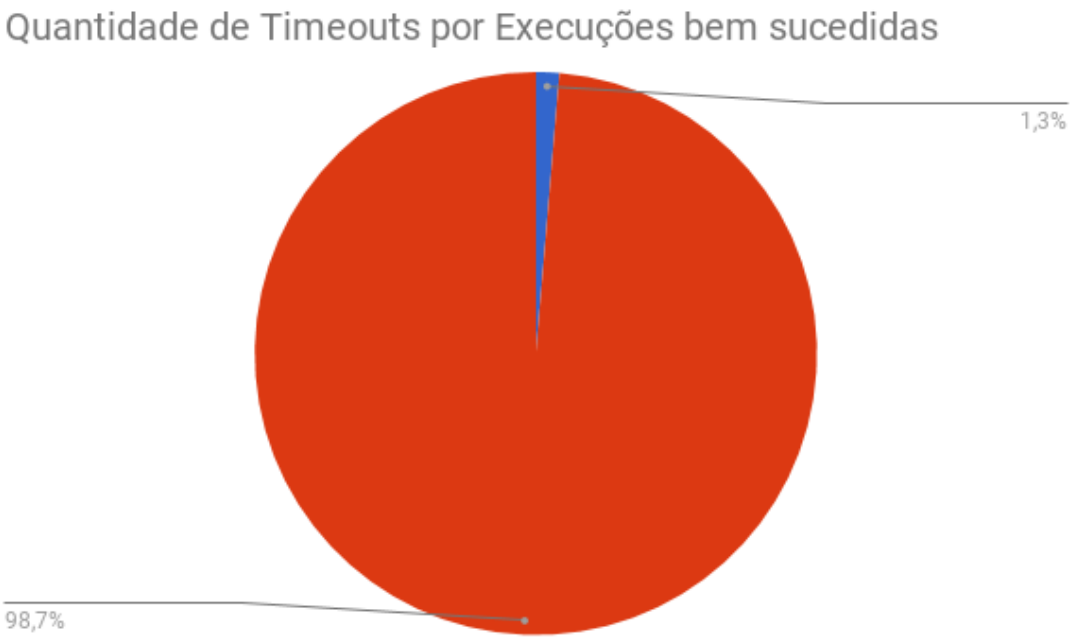

Figura 5: Gráfico a frequência de ocorrência de timeouts. Em azul ocorrência de timeout, em vermelho recepção normal de mensagem.

A taxa resultante é de 1,33% de chances de ocorrência. Considerando que 2 das ocorrências aconteceram na função de escrita de comentário, que é a função que mais tomou tempo, foi fácil observar que caso o timeouts fosse aumentado estas ocorrências poderiam ter sido diminuídas. Ainda sim, o percentual é muito pequeno fazendo com que o UDP mesmo não sendo naturalmente seguro por não lidar bem com falhas, pode em geral ser confiável para enviar pelo menos 99% dos dados com confiança.

6 Comparação entre TCP e UDP

Considerando os valores obtidos de tempo pelo TCP anteriormente, conseguimos fazer as comparações devidas dos tempos de envio e de recepção de mensagens. Inicialmente, devemos fazer uma adaptação considerando que no servidor TCP eram enviadas múltiplas mensagens durante uma mesma operação, e no servidor UDP isto foi simplificado para um datagrama conter toda a informação necessária para o cliente para uma dada função.

Assim foi possível comparar os tempos de envios de mensagens em cada uma das funções e o tempo médio:

$$t_{TCP} = 2946 \pm 420\mu s$$

$$t_{UDP} = 171 \pm 54\mu s$$

.

Também podemos reforçar os resultados médios a partir do gráfico a seguir, que apresenta uma comparação de TCP e UDP para cara operação implementada.

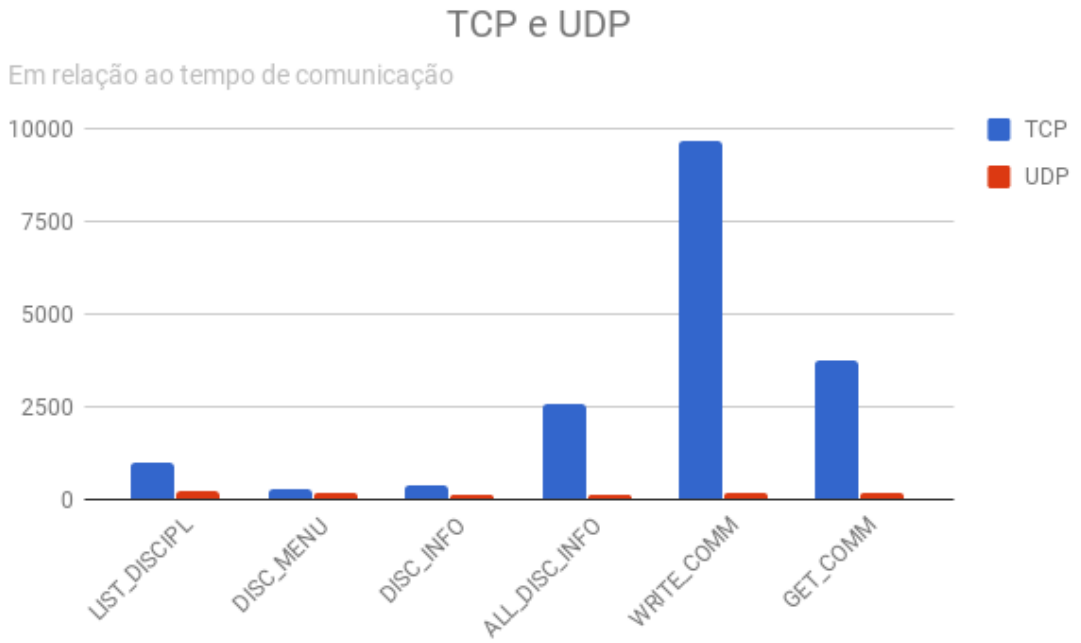

Figura 6: Gráfico de comparação dos tempos de comunicação de cada operação para o TCP e para o UDP.

Assim podemos concluir que o envio de mensagens pelo protocolo UDP é muito mais rápido que o TCP. Isto se deve uma vez que o TCP deve garantir o envio de todas as mensagens, fazendo reenvio quando necessário, também garante a prdem das

mesmas e faz controle de tráfico. Devidas a estas medidas tomadas pelo protocolo é de se esperar que o seu tempo médio seja maior, pŕincipalmente quando se leva em conta que os tempo medidos de timeout em UDP não foram considerados.

1.
$$t_{UDP} = (205,41 \pm 42)\mu s$$
$$t_{TCP} = (1010,06 \pm 170,25)\mu s$$

2.
$$t_{UDP} = (189,55 \pm 40)\mu s$$
$$t_{TCP} = (266,68 \pm 179,07)\mu s$$

3.
$$t_{UDP} = (147,99 \pm 19)\mu s$$
$$t_{TCP} = (362,17 \pm 102)\mu s$$

4.
$$t_{UDP} = (149,01 \pm 29)\mu s$$
$$t_{TCP} = (2573,66 \pm 678,21)\mu s$$

5.
$$t_{UDP} = (171,77 \pm 84)\mu s$$
$$t_{TCP} = (9708,2 \pm 7047,68)\mu s$$

6.
$$t_{UDP} = (176,50 \pm 64)\mu s$$
$$t_{TCP} = (3760,83 \pm 6035,27)\mu s$$

Os dados refletem o que é visto no gráfico acima, que os tempos médios de nenhuma das operações de TCP foi inferior a do UDP. Considerando também as diferenças dos tempos em cada uma das operações

7 Conclusão

A partir dos dados extraídos foi possível concluir a eficiência da execução das operações que não necessitavam o uso de arquivos. Além disso foi observado o tempo de comunicação do UDP como sendo 171us, que por si só, já é um tempo esperado de mensagens em LAN. Comparando-o com o tempo do TCP que é 2964us foi observado que o tempo tomado pelo reenvio de mensagens e quaisquer medidas tomadas pelo protocolo TCP para garantir segurança e ordenação atrasam bastante o tempo médio de conexão, fazendo do UDP mais rápido mesmo que menos seguro.

Referências

[1] Stevens, R. *UNIX Network Programming. Vol. 1 "The Sockets Networking API"*. 3rd Edition, Addison-Wesley, 2003. BIMECC 005.43St47u.

[2] Beej's Guide to Network Programming, `http://beej.us/guide/bgnet/`