MC833 - Projeto 2
Cliente e Servidor UDP e TCP

Luciano Zago - 182835
Vinicius Couto - 188115

Introdução

O projeto tem como objetivo criar uma comunicação TCP, através de sockets, entre um cliente e um servidor concorrente, e uma comunicação UDP entre um cliente e um servidor iterativo. Além disso, será realizado uma comparação entre os protocolos utilizados, através de análises de tamanho de código, confiabilidade e tempos de comunicação da aplicação. Os servidores (UDP e TCP) devem ser capazes de receber as requisições do cliente localizado em uma máquina diferente, e transmitir todas as informações disponíveis. O cliente deve poder optar entre receber as informações disponíveis via TCP ou via UDP, comunicando-se com o servidor de preferência.

Sistema

1. Descrição Geral

Para estabelecer um ambiente em que o cliente pudesse optar por utilizar protocolo UDP ou TCP, foi criado dois servidores em diferentes portas (um TCP e um UDP) de tal forma que o cliente consiga criar dois sockets (vinculados às diferentes portas dos diferentes servidores) e selecionar qual deles utilizar para a troca de mensagem.

Os servidores e o cliente estão divididos em dois arquivos: ".h" e ".c". Para ambos, o ".h" define assinaturas de funções, inclui bibliotecas necessárias, define constantes essenciais e possui *wrappers* para funções de envio e recepção tanto do protocolo TCP quanto do UDP. Em relação às constantes, deve-se ressaltar a importância de "BUFFLEN", "TCP_PORT" e "UDP_PORT". A primeira é responsável por padronizar o tamanho da mensagem, fazendo com que haja um mapeamento 1:1 entre um envio do servidor com sua recepção no cliente, e vice-versa. A segunda define a porta em que deve-se estabelecer a conexão TCP/UDP. Estas devem assumir os mesmos valores nos servidores e no cliente. Outros componentes importantes nos *headers* são os *wrappers* para as funções *send, recv, sendto* e *recvfrom*. Os wrappers, além de capturar erros das *syscalls*, garantem que o tamanho da mensagem enviada/recebida seja sempre igual ao valor de "BUFFLEN", garantindo o mapeamento 1:1 citado anteriormente. No cliente, o *wrapper* denominado *transfer* serve para criar uma interface única de mensagens UDP e TCP, possibilitando que qualquer mensagem de qualquer protocolo possa ser enviada através de uma única chamada de função.

O fluxo de execução é o mesmo do primeiro projeto, com algumas diferenças para o servidor UDP. Ambos os servidores possuem uma estrutura *switch-case* que coordena a troca de mensagens com o cliente. Uma das diferenças é que, enquanto o TCP é concorrente, o UDP é iterativo, o que implica que o UDP trabalha com uma fila de

requests de clientes enquanto o TCP apresenta um processo dedicado e paralelo para cada cliente. Outra diferença entre os servidores TCP e UDP se encontra no tratamento de erros. No caso do TCP, uma falha de comunicação implica a parada do programa, uma vez que impossibilita a troca de mensagens por este canal. No caso do UDP, erros são tolerados, uma vez que é possível a perda de mensagens neste protocolo. Consequentemente, a consistência da informação recebida não é garantida, mas, em compensação, o servidor UDP é mais flexível por não necessitar uma conexão propriamente dita e ser mais tolerante a erros.
O cliente também apresenta algumas mudanças durante o uso do protocolo UDP. Devido a possibilidade de perda de mensagens, foi definido um *timeout* utilizando a função *setsockopt* a fim de evitar que o cliente seja bloqueado pela *syscall*.
O armazenamento dos dados segue o mesmo modelo do primeiro projeto.

2. Casos de Uso

<protocolo> 1 <email> - Lista todas as informações de um perfil

O campo protocolo deve ser substituído por **u** ou **t**, a fim de indicar se o cliente deseja se comunicar via **UDP** ou **TCP**, respectivamente. O campo email é utilizado de chave para identificar o perfil desejado.

Estrutura de dados e armazenamento

Os dados do servidor estão todos armazenados dentro do diretório “server/data”. As imagens de perfil estão armazenadas no diretório “server/data/images”.
A estrutura de dados consiste em um arquivo “index.txt” para registrar as chaves (emails) cadastradas no sistema, e arquivos “[email].txt” para armazenar as informações relativas ao perfil do usuário. As fotos de perfil são armazenadas em arquivos “images/[email].jpg”.
Cada tipo de informação do perfil é armazenada consistentemente em determinada linha do arquivo de texto, como no esquema a seguir:

Linha 1	Nome
Linha 2	Sobrenome
Linha 3	Cidade
Linha 4	Curso
Linha 5	Habilidades
Linha 5+i, i>=1	Experiência nº i

Implementação do servidor UDP

O servidor foi implementado visando o armazenamento de dados de forma persistente requisições feitas iterativamente entre clientes. O servidor UDP, por ser iterativo, consiste em um único processo associado a porta “UDP_PORT” com protocolo IPv4 (AF_INET). Após a configuração das informações do servidor e da porta vinculada a ele, aloca-se a struct *cliaddr*

para armazenar as informações dos clientes possibilitando a resposta do servidor UDP [1]. Para fins de praticidade, supôs-se que todos os comandos fornecidos pelo cliente eram válidos dentro do escopo de operações do servidor. Como o protocolo UDP não garante a entrega e a ordenação das mensagens, a maior parte dos casos de erros foram tolerados, garantindo que o servidor continue executando mesmo que haja perda ou embaralhamento de informações. O servidor também pode aceitar requisições de múltiplos clientes iterativamente, atendendo as requisições por ordem de chegada e sem misturá-las, dado que ele atende um endereço de cliente por vez.

Na perspectiva do cliente, as adaptações necessárias para a interface de comunicação TCP, são similares às do servidor UDP: os *wrappers* para mensagens UDP do cliente são mais tolerantes a erros garantindo que, ao invés de interromper a execução do cliente, o cliente possa retomar o fluxo de execução permitindo novas *requests* mesmo que uma troca de mensagens tenha sido inutilizada. Além disso, devido a possibilidade de perda de mensagens, foi definido um *timeout* utilizando a função *setsockopt* a fim de evitar que o cliente seja bloqueado pela *syscall*. Nota-se que, no caso de *timeout*, a troca de mensagens é interrompida e, consequentemente, as informações recebidas tornam-se incompletas e inconsistentes, mas tanto o cliente quanto o servidor continuam a execução.

Resultados

Os resultados correspondem à consultas realizadas em máquinas diferentes em uma rede local.

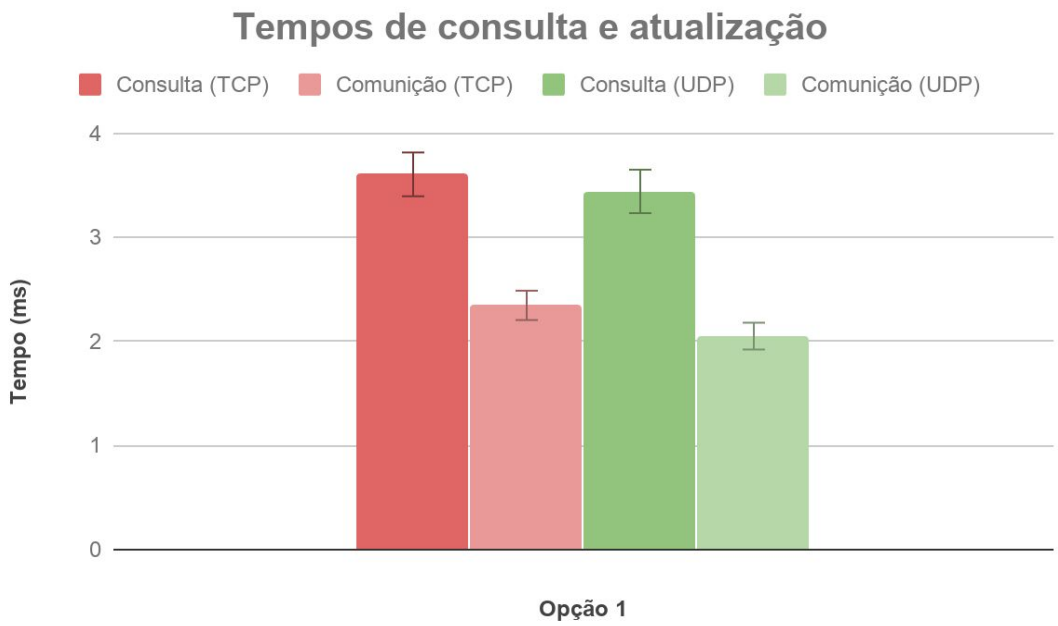
O cálculo do erro com intervalo de confiança de 95% foi realizado como $\sigma_M = 1.96 \times \frac{\sigma}{\sqrt{N}}$ sendo σ o desvio padrão e N o número total de iterações. O valor 1.96 é a constante que representa o intervalo de confiança de 95% [2].

Descrevendo as legendas dos tempos apresentados no gráfico e na tabela, temos:

- Operação: tempo de processamento de uma *request* pelo servidor.
- Consulta: tempo que que o cliente leva enviar e terminar de receber a resposta de uma *request*.
- Comunicação: diferença entre os tempos de consulta e operação que resulta em uma média que representa o tempo gasto em troca de mensagens pela rede.

Tempos de consulta, atualização e comunicação (milisegundos)						
Iterações	**TCP**			**UDP**		
-	**Operação**	**Consulta**	**Comunicação**	**Operação**	**Consulta**	**Comunicação**
1	3.964	5.360	1.396	1.615	3.721	2.106
2	1.185	3.365	2.180	1.276	3.111	1.835
3	1.329	3.725	2.396	1.541	3.491	1.950
4	1.044	3.397	2.353	3.008	4.837	1.829
5	1.129	3.978	2.849	1.378	3.672	2.294

6	1.105	3.390	2.285	1.204	3.702	2.498
7	1.209	3.572	2.363	1.215	2.986	1.771
8	1.060	3.348	2.288	1.258	2.777	1.519
9	1.122	3.461	2.339	1.367	3.547	2.180
10	1.049	3.376	2.327	0.694	3.389	2.695
11	1.015	3.301	2.286	1.340	3.291	1.951
12	1.046	3.164	2.118	2.180	4.122	1.942
13	0.838	3.376	2.538	1.357	3.392	2.035
14	1.004	3.399	2.395	1.253	3.498	2.245
15	0.956	3.355	2.399	1.483	3.326	1.843
16	1.009	4.142	3.133	1.156	3.244	2.088
17	1.189	3.494	2.305	1.236	3.683	2.447
18	1.691	3.868	2.177	1.349	3.390	2.041
19	1.080	3.419	2.339	1.363	2.965	1.602
20	1.137	3.617	2.480	0.526	2.682	2.156
Média (ms)	**1.258**	**3.605**	**2.347**	**1.390**	**3.441**	**2.051**
Erro (95%)	**0.289**	**0.210**	**0.141**	**0.219**	**0.209**	**0.129**



Analisando os resultados, nota-se que o tempo de comunicação corresponde a uma parte significativa do tempo de consulta, sendo responsável cerca de 60% do tempo de consulta.
Nota-se que o tempo de comunicação do TCP é um pouco mais demorado que o UDP. O TCP garante tanto a ordem quando a entrega das mensagens, portanto espera-se que este seja mais demorado, uma vez que ele precisa executar mais passos para fornecer essas garantias. Um exemplo clássico é o uso de *ACKs* para garantir que uma mensagem chegou ao destinatário. Em contrapartida o UDP apenas enviar a mensagem sem realizar nenhum outro procedimento, já que não garante entrega nem ordenação de mensagens.
Nota-se, também, que não houve perda de mensagens nos testes feitos e as diferenças entre os tempos de comunicação, apesar de significativa, foi pequena. Justifica-se essa observação pelo fato que os testes foram feitos entres máquinas diferentes mas na mesma rede local. Tanto a perda de mensagens quanto as discrepâncias dos tempos de comunicação são fortemente amenizadas neste cenário. Para obter resultados mais realistas, teriam de ser realizados testes entre máquinas em diferentes redes, o que causaria um aumento na latência do TCP e demonstraria a perda de mensagens UDP. Tal cenário possibilitaria a atuação significativa de fatores como o controle de congestionamento e uso de *ACKs* para o TCP, a perda de mensagens e a ausência de *overheads* para o UDP.
Acrescenta-se que, apesar de que o tempo de operação (vide tabela) seja, em média, diferente para os dois servidores, o erro deles apresenta uma intersecção significativa, implicando que, na prática, ambos os servidores realizam as operações em tempos similares.

Comparação TCP vs. UDP

Comparando os códigos fonte dos servidores, nota-se que o servidor UDP iterativo requer menos passos para se estabelecer a comunicação. Além de não usar *fork* para paralelizar o atendimento de clientes, o UDP também dispensa a necessidade de aceitar conexões usando o *accept*, tornando-o a opção mais fácil e rápida de ser implementada.
Todavia, a praticidade do servidor UDP resulta em um *tradeoff* com confiabilidade. A principal falha de comunicação entre cliente-servidor foi perda de mensagens, com alguns raros casos de embaralhamento de dados. Apesar de que a ocorrência de erros foi rara, deve-se considerar que os testes foram feitos entre máquinas na mesma rede: escalando o teste para redes diferentes e longas distâncias, a confiabilidade das transmissões seria reduzida consideravelmente.
Em relação aos tempos obtidos, o TCP foi, em média, mais lento que o UDP; todavia, a diferença obtida foi muito pequena, principalmente ao se considerar o erro dos tempos. Justifica-se essa observação pelo fato o teste foi feito em rede local dedicada, como descrito na análise dos resultados.

Conclusão

O projeto criou um cliente que se comunica com servidores UDP (iterativo) e TCP (concorrente) e que, através da medição do tempo em cada operação, permite comparar a performance de ambos protocolos.

A análise dos resultados mostrou que o resultado obtido coincidiu com o esperado: a comunicação UDP foi mais rápida que a comunicação TCP. Vale notar, porém, que por ser usado apenas máquinas em uma rede local, não houve perda de mensagens nos testes feitos e as diferenças entre os tempos de comunicação, apesar de significativa, foi pequena.
Portanto, o projeto atingiu o objetivo esperado, estabelecendo dois servidores de diferentes protocolos e um cliente que possa comunicar-se com ambos, todavia, seria necessário testar em grande escala para que as diferenças de tempo entres os protocolos o TCP e UDP fossem mais maiores.

Referências

1. Beej's Guide to Network Programming (http://beej.us/guide/bgnet/html/single/bgnet.html)
2. Confidence Interval on the Mean (http://onlinestatbook.com/2/estimation/mean.html)