Redes de Computadores - UFRJ - 2022.2
Lista de Exercícios - Programação com Sockets

Filipe Castelo Branco de Souza
DRE: 119066195

06/10/22

Fiz esse trabalho sem ajuda de colegas e consultei o livro-texto do Kurose versão americana e alguns sites na internet. A versão final do trabalho foi feita por mim de forma independente.

1)

Como o servidor UDP não é finalizado após receber uma mensagem do cliente (fica aberto indefinidamente), não precisaremos modificá-lo. Precisaremos modificar apenas o cliente para que não seja fechado após o envio da mensagem. Para isso, utilizaremos um while e um if.

2)

Nesse caso o servidor encerra após rodar uma vez, então precisamos mudar tanto o cliente quanto o servidor.

Como o servidor deixa uma conexão TCP aberta depois de enviar uma resposta, os requests e respostas seguintes entre o mesmo cliente e servidor podem ser enviados na mesma conexão, reduzindo o congestionamento da rede. Também não precisa alocar tantos buffers de TCP nem guardar tantas variáveis de TCP no cliente e servidor. Além disso, não tem tanto delay de RTT como nas conexões não persistentes, já que só ocorrerá uma vez. Os objetos são enviados de uma vez, sem precisar esperar por todas as respostas dos requests pendentes.

Os recursos são liberados imediatamente após o uso, e como é feita uma conexão para cada request, não tem o problema de ficar segurando todos os arquivos enquanto a conexão não for encerrada.

Ao executar vários clientes em paralelo, o servidor dá conta apenas de um, então ele retorna na mesma ordem em que foram feitas as conexões, como se os clientes estivessem em uma fila. Ou seja, o segundo cliente não poderá receber sua mensagem enquanto o primeiro não tiver encerrado a conexão.

4)

A diferença do UDP é que ele cria uma conexão para cada cliente, enquanto o TCP utiliza apenas uma conexão persistente. É possível ver essa diferença pelos prints, já que o servidor UDP imprime cada número em uma linha, enquanto o TCP imprime todos os números na mesma linha. Além disso, o UDP não garante que a mensagem será entregue em ordem, diferente do TCP.

6)

Vantagens do stateful: protocolos stateful mantêm informação da conexão, e por isso entregam performance superior; eles são mais intuitivos pois mantêm dados no servidor entre as duas requisições; podem aumentar a performance quando há necessidade de recuperar dados apenas uma vez.

Desvantagens do stateful: precisa alocar memória para guardar dados; caso haja uma manutenção ineficiente da sessão pode haver redução na performance; é necessário gerenciamento contínuo; normalmente protocolos stateful precisam de espaço de armazenamento.

Vantagens do stateless: é mais fácil se recuperar de falhas partciais já que nenhum dado é mantido; o servidor não precisa armazenar a seção entre as requisições, então a escalabilidade é maior já que é possível deployar o serviço para qualquer número de servidores; precisa de uma quantidade pequena de recursos porque o sistema não precisa guardar informação da sessão.

Desvantagens do stateless: pode ser necessário incluir informações adicionais em cada requisição; talvez diminua a performance da rede ao aumentar a quantidade repetida de dados enviados nas requisições, já que não podem ser salvos e reutilizados.

7) Problema 29

a)

É necessário utilizar um número de sequência aleatório, pois se fosse sempre o mesmo número (por exemplo, começando de 1) seria facilmente alvo

de ataques de syn flood.

b)

Não, o invasor não consegue criar conexões apenas enviando ACKs para o alvo. Conexões meio abertas não são possíveis, já que um servidor usando SYN cookies não mantém variáveis de conexão e buffers nas conexões antes que conexões completas sejam estabelecidas. Em relação a conexões completas, um invasor teria que saber o número de sequência inicial para fazê-lo, então como ele não tem como saber, não tem como criar uma conexão completa.

c)

Não, o servidor pode adicionar um timestamp ao computar esses números de sequência iniciais e escolher um tempo de utilização, descartando os números expirados caso o invasor reuse-os.

Extra

Ao usar o comando *bind*, é necessário escolher um socket e porta livres, pois o *bind* liga o socket ao endereço. Ao fazer isso a combinação de IP e porta ficam abertas para algum cliente conectar nelas.

8) Problema 30

a)

Se valores de timeout forem fixos, aumentar o tamanho do buffer aumentaria a chance de ele ser retransmitido por atingir o tempo do timeout (estaria na fila mas não teria sido perdido) então os emissores poderiam acabar dando um timeout precoce. Ou seja, alguns pacotes seriam retransmitidos mesmo se não forem perdidos. O roteador teria usado melhor a capacidade de transmissão se em vez disso tivesse enviado um pacote diferente.

b)

Se valores de timeout forem estimados (como os do TCP), então aumentar o tamanho do buffer com certeza ajudaria a aumentar o throughput de determinado roteador. Mas pode acabar tendo um problema, já que o atraso de fila pode acabar ficando muito grande.

Extra

Conexões múltiplas melhoram o throughput, pois o throughput total é a soma de todos os throughputs de todas as conexões.

9) Problema 31

Sabemos que as fórmulas para DevRTT, EstimatedRTT e TimeoutInterval são as abaixo:

$$DevRTT = (1 - \beta) \cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT|$$
$$EstimatedRTT = (1 - \alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$$
$$TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$$

Agora vamos fazer os cálculos. Depois de obter o primeiro SampleRTT $106ms$:

$$DevRTT = 0.75 \cdot 5 + 0.25 \cdot |106 - 100| = 5.25ms$$
$$EstimatedRTT = 0.875 \cdot 100 + 0.125 \cdot 106 = 100.75ms$$
$$TimeoutInterval = 100.75 + 4 \cdot 5.25 = 121.75ms$$

Depois de obter $120ms$:

$$DevRTT = 0.75 \cdot 5.25 + 0.25 \cdot |120 - 100.75| = 8.75ms$$
$$EstimatedRTT = 0.875 \cdot 100.75 + 0.125 \cdot 120 = 103.16ms$$
$$TimeoutInterval = 103.16 + 4 \cdot 8.75 = 138.16ms$$

Depois de obter $140ms$:

$$DevRTT = 0.75 \cdot 8.75 + 0.25 \cdot |140 - 103.16| = 15.77ms$$
$$EstimatedRTT = 0.875 \cdot 103.16 + 0.125 \cdot 140 = 107.76ms$$
$$TimeoutInterval = 107.76 + 4 \cdot 15.77 = 170.84ms$$

Depois de obter $90ms$:

$$DevRTT = 0.75 \cdot 15.77 + 0.25 \cdot |90 - 107.76| = 16.27ms$$
$$EstimatedRTT = 0.875 \cdot 107.76 + 0.125 \cdot 90 = 105.54ms$$
$$TimeoutInterval = 105.54 + 4 \cdot 16.27 = 170.62ms$$

Depois de obter $115ms$:

$$DevRTT = 0.75 \cdot 16.27 + 0.25 \cdot |115 - 105.54| = 14.57ms$$
$$EstimatedRTT = 0.875 \cdot 105.54 + 0.125 \cdot 115 = 106.72ms$$
$$TimeoutInterval = 106.72 + 4 \cdot 14.57 = 165ms$$

Extra

A flag $MSG\_WAITALL$ é usada para evitar um loop em uma operação de receive até que todos os dados tenham chegado. Permite que a aplicação espere que o buffer inteiro na operação seja recebido antes de satisfazer uma operação bloqueada.