Socket Programming

Ingrid Santana Lopes
14/0083065
Departamento de Ciência da Computação,
Universidade de Brasília
Brasília,Brasil

Marcos Paulo Cayres Rosa
14/0027131
Departamento de Ciência da Computação,
Universidade de Brasília
Brasília,Brasil

I. METODOLOGIA

Nessa sessão, está explicada a forma como cada requisição foi implementada sobre o protocolo UDP de exemplo fornecido em linguagem Python.

A. *Entrega confiável e ordem dos dados*

A entrega confiável e a ordem dos dados foi implementada de forma que: dada uma mensagem que o cliente deseja enviar, o checksum da mesma será calculado. Quando a mensagem é enviada ao servidor juntamente com o resultado desse checksum, o checksum é recalculado do lado do servidor apenas para o trecho correpondente à mensagem original para fins de comparação. Caso os dois checksums, o recebido pelo servidor e o calculado novamente com base na mensagem recebida, estiverem iguais, então não houve corrupção de dados. Existe a possibilidade de duas mensagens diferentes terem o mesmo checksum, mas a probabilidade para isso acontecer depois de uma dada mensagem ter sido corrompida é muito pequena.

Quanto a ordem, tanto o cliente quanto o servidor contém um contador que é incrementado para cada mensagem enviada ou recebida respectivamente. O cliente envia a mensagem junto com sua soma de bits e a sequência da mesma. Ao chegar no servidor, ele separa tais informações e checa se o número de sequência enviado pelo cliente confere com o número de sequência que esperaria receber. Caso não seja, ele detecta como estando fora de ordem e informa isso ao cliente para que ele possa então enviar o pacote correto de novo e continuar enviando os pacotes seguintes a partir disso. Tal situação de fora de ordem acontece quando um pacote se perde ao ser enviado do cliente ao servidor.

B. *Controle de Fluxo e Pipeline*

O Pipeline implementado foi o Go N Back, o cliente envia mensagens contínuas levando em conta sempre o tamanho da janela e a movimentando quando recebe um pacote de confirmação (ACK) do receptor e, no caso do exemplo, a mensagem em caps lock. A ideia de controle de fluxo no Go N Back está intimamente ligada à janela deslizante que representa um valor de pacotes que o servidor consegue suportar dado o buffer do mesmo.

No caso de um erro em que algum pacote se perca em qualquer um dos sentidos, a janela não irá se movimentar e, assim, o pacote que se perdeu será reenviado e todos aqueles depois dele também para que o servidor possa obte-los na ordem certa e então mandar a mensagem alterada de volta para o cliente também na ordem certa. Ou seja, se qualquer pacote for perdido ou danificado, então esse pacote e todos os pacotes a seguir na janela (mesmo se eles foram recebidos sem erro) vão ser re-enviados. Dessa forma, o pequeno trecho onde a ordem ficou errada, seria desconsiderado. Como tanto o cliente, como o servidor, notam quando um pacote falta, a mensagem de erro é indicada no terminal.

II. EXEMPLOS DE EXECUÇÃO

Foram feitas execuções fornecendo exatamente 26 mensagens no formato de strings de tamanhos variados retiradas do capítulo 2 de Harry Potter e a Pedra Filosofal. Uma dessas execuções foi registradas para ser mostradas de exemplo neste relatório.

A simulação já funciona usando o pipeline Go n Back. No iníio, não ocorre primeiramente nenhum erro, como pode ser observado na figura 1.

Figura 1. Começo da execução

Pode-se ver como uma mensagem gerada pelo cliente é recebida pelo servidor que então manda a mensagem alterada de volta que é recebida com sucesso. Como até essa etapa não ocorre nenhum erro ou problema, quando a informação chega no servidor, ela rapidamente é mandada em sua nova forma ao cliente.

Depois de algumas trocas de mensagens, ocorre um erro de perda de pacote na sequência 3 como pode ser visto na Figura 2. Ano notar isso, o servidor não manda a mensagem alterada de volta para o cliente pois, ele não a recebeu. Graças a isso, as mensagens depois dessa serão indicadas pelo servidor

como estando fora de ordem, assim como pode ser notado na figura 2. Mas logo as mensagem voltam para a ordem quando o cliente nota que algo está errado e reenvia a mensagem podendo então finalmente movimentar sua janela.

Figura 2. Erro de perda de pacote e out of order

O cliente segue movimentando a janela a cada mensagem recebida de volta do servidor depois do cliente ter enviado a sua mensagem até que ocorre um erro de corrupção na sequência 25.

Figura 3. Corrupção e final da execução

O erro de corrupção da mensagem 25 só vai ser corrigido depois que, após alguns envios de mensagens, o cliente notar que tem algo errado pois não irá conseguir movimentar a janela na qual o Go N Back se baseia para seu funcionamento.

Dessa forma, como pode ser observado na figura 3, a mensagem de número de sequência 25 será enviada novamente e o servidor poderá retorná-la em letras maiuscúlas.

Quando a sequência de 26 mensagens termina, o cliente termina de receber as mensagens que faltavam e então a conecção é encerrada e o servidor mostra quando tempo, em segundos, ele levou para executar tal simulação como pode ser visto também na figura 3.

Em relação a implementação desses erros, foram randomizadas valores que são comparados a uma probabilidade para decidir se irá corromper dados do pacote (no código do cliente antes de enviar o pacote para o servidor) ou perder o pacote (no código do servidor antes de passar os dados para a camada superior). Para detecção de erro, o cliente tenta receber os ACKs do servidor por um certo período de tempo especificado e, caso haja uma discrepância entre o momento atual e o do último ack que tenha movido a janela, será visto como um erro e assim reenviada a janela completa. Para o servidor, quando não se recebe mensagem por um certo período e já foi recebida a última mensagem, padronizada como um texto vazio, ele termina seu serviço.

III. INFORMAÇÕES DE DESEMPENHO

Como a simulação de perdas de pacote e corrupção baseia-se na geração de números randômicos que podem ou não estar dentro de uma faixa de probabilidade que determina se isso vai acontecer, a ocorrência dos mesmos se baseia nessa probabilidade. Para a nossa implementação, randomizamos valores entre zero e um e caso fossem menores do que 0.1, aplicávamos a lógica para corromper os dados ou perder pacotes, dependendo do caso.

Para detecção de erro, depende da tempo especificado como a diferença entre o atual e o último ack com número de sequência novo recebido. Assim, quando a diferença é maior que o limiar estabelecido, o cliente reenviará a janela. No caso, colocamos essa diferença como 0.01, testado a cada tempo limite de 0.001. Colocamos esses valores a partir de testes realizados para não reenviar pacotes excessivamente e permitir que o erro fosse detectado pelo cliente em um período que o servidor não ficasse com demasiada captação de erro.