DCC023: Redes de Computadores

Caio Augusto Ferreira Godoy - 2011048987 - Turma Dorgival
Lucas Furtini Veado - 2013007609 - Turma Italo

RMTLOG: Um Sistema de Log Remoto

Introdução

"Um log server remoto é nada mais do que um sistema configurado para prover espaço em disco para outros sistemas registrarem os seus logs." - Eric Hines. Os dados enviados para o log server poderão ser utilizados para reconstruir os eventos registrados na rede monitorada além de armazenar de forma segura e confiável estes dados obtidos.

Neste trabalho implementamos o RMTLOG, uma biblioteca para criação, transmissão e armazenamento de logs. A biblioteca permite que uma aplicação executando em um dispositivo D1 gere mensagens de log e as envie para um coletor executado no dispositivo D2. A biblioteca realiza a codificação e transmissão confiável de mensagens entre aplicações e um coletor de logs.

Desenvolvimento

Foram desenvolvidos dois programas neste trabalho, o cliente e o servidor. Ambos se comunicam através do protocolo de rede UDP e implementam uma janela deslizante com retransmissão seletiva. Ambos os programas recebem como parâmetro um valor de probabilidade de erro, que deve ser levado em conta na hora de criar os pacotes, que simula o envio de pacotes corrompidos. Os detalhes de cada programa será descrito a seguir.

Cliente

O Cliente é responsável por ler um arquivo de entrada contendo as mensagens de log, criar um pacote para cada mensagem e enviá-las ao servidor utilizando uma janela deslizante para manter controle dos itens enviados e confirmados.

O programa recebe de entrada cinco parâmetros, são eles:

- **‘Arquivo’ (string):** nome do arquivo que contém as mensagens de log
- **‘IP:Porta’ (string):** endereço e porta que o servidor irá receber as mensagens
- **‘WTX’ (int):** o tamanho da janela deslizante de transmissão
- **‘TOUT’ (int):** tempo de timeout para reenvio de mensagens perdidas
- **‘PERROR’ (float):** probabilidade de gerar erro no pacote

Sendo assim, o programa é executado da seguinte maneira (exemplo):

$ python3 cliente.py inputFile 127.0.0.1:5000 5 10 0.5

Como existem dois fluxos que precisam ser executados simultaneamente (envio de pacotes e recebimento e verificação de ACKs), foram criadas duas Threads de execução: uma para lidar com o envio dos pacotes e a outra para o controle dos ACKs recebidos. A primeira thread é responsável por verificar os índices da janela criar e enviar os pacotes que estiverem dentro da janela; ela também é responsável por marcar o tempo e re-enviar um pacote caso o TOUT tenha sido atingido; durante a criação dos pacotes, o valor do PERROR é levado em conta e pacotes corrompidos são gerados à partir de seu valor. A segunda thread é responsável por receber os ACKs do servidor, que

possuem tamanho máximo de 36 bytes, verificá-los, para o caso de estarem corrompidos, e fazer o ajuste da janela e dos pacotes a enviar conforme os ACKs são confirmados.

Ao final da execução o programa imprime na tela e finaliza a execução. A linha impressa contém o número de mensagens de log distintas, o número de mensagens de log enviadas (incluindo retransmissão), o número de mensagens enviadas com o MD5 incorreto e o tempo total de execução.

Um detalhes que vale a pena destacar é que, conforme foi dito em sala, o programa cliente não faz gerenciamento de memória. Sendo assim, para facilitar a implementação e focar no desenvolvimento do protocolo, lemos e salvamos todo o conteúdo do arquivo de log na memória em uma lista antes de começar o envio dos dados.

Servidor

O servidor é responsável por receber as mensagens recebidas dos clientes, enviar mensagens de ACK para cada pacote recebido com sucesso e salvar essas mensagens em um único arquivo para todos os clientes. As mensagens são escritas no arquivo respeitando a ordem de chegada de cada cliente, porém as mensagens podem estar intercaladas entre clientes.

O programa recebe de entrada quatro parâmetros, são eles:

- **‘Arquivo’ (string):** nome do arquivo onde serão salvas as mensagens
- **‘Porta’ (int):** porta na qual o servidor irá receber as mensagens
- **‘WRX’(int):** o tamanho da janela deslizante de recepção
- **‘PERROR’ (float):** probabilidade de gerar erro no pacote

Sendo assim, o programa é executado da seguinte maneira (exemplo):

$ python3 servidor.py outputFile 5000 5 0.3

Cada cliente distinto possui a sua própria janela deslizante que é inicializada quando o servidor recebe o primeiro pacote do respectivo cliente. Ao receber um pacote, o servidor verifica se já existem mensagens recebidas daquele cliente, caso positivo, ele checa o MD5 da mensagem para confirmar sua corretude, se o pacote estiver correto ele então verifica o número de sequência do pacote e verifica se esse número está dentro da janela esperada. Se o número for igual ao primeiro valor esperado (NFE), ele envia um ACK para o cliente, salva as mensagens recebidas em sequência até o momento no arquivo e reajusta os índices da janela; se o número do pacote for maior que o primeiro esperado, mas estiver dentro da janela esperada, ele salva o pacote na janela e envia um ACK daquele pacote ao cliente mas não ajusta os índices da janela; o mesmo vale para mensagens recebidas cujo pacote estiver abaixo do primeiro valor esperado.

O servidor só escreve no arquivo quando recebe um pacote cujo número de sequência é igual ao valor do próximo quadro esperado (NFE), nesse caso ele percorre a janela e escreve todos os pacotes que foram recebidos e que estão em sequência até aquele momento.

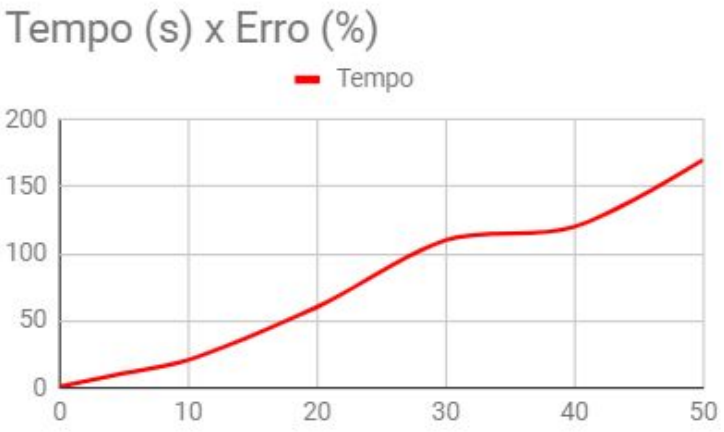
Quando o servidor recebe uma mensagem com o mesmo número de sequência e hashes corretos, mas com o conteúdo diferente (caso anômalo), ele executará os passos descrito e, se o número de sequência for igual ao primeiro valor esperado ele salva a mensagem no arquivo e anda com a janela, ao receber o segundo quadro com o mesmo número de sequência, ele irá reenviar o ACK daquele pacote, pois o valor do número de sequência recebido será menor que o valor mínimo da janela, salvando no arquivo o conteúdo do **primeiro** pacote com mesmo número de sequência recebido.. Se o número de sequência não for o menor esperado, mas estiver dentro da janela, o servidor salvará o quadro na janela e enviará o ACK, quando o segundo pacote com mesmo número de sequência chegar, o servidor sobrescreverá o pacote já salvo na janela e enviará um ACK, sendo assim ele irá salvar no arquivo o conteúdo do **último** pacote de mesmo número de sequência recebido.

Pacotes são descartados caso o MD5 não esteja correto ou o valor de sequência do pacote estiver acima do maior valor esperado na janela. Para finalizar o programa servidor, deve-se entrar com o comando CTRL+C no prompt de execução.

Testes e Desempenho

Para testar o correto funcionamento dos nossos programas, realizamos diversos testes. Como nós implementamos cada um dos programas com um parâmetro que gera erro na mensagem enviada, conseguimos testar com mais facilidade o código. Abaixo discutimos alguns desses testes e os resultados levantados. Os testes foram executados num computador notebook Acer ASPIRE com processador Intel Core i7 - 2670QM 2.60GHz, 4GB de memória. Cada teste foi rodado 3 vezes e o valor mostrado é a média dos valores observados.

Mantendo fixo o tamanho da janela de transmissão e recebimento (tamanhos 5 e 10 respectivamente), e também o tamanho do arquivo de entrada (o arquivo contém 10 linhas para envio), mas variando a probabilidade de erro, observamos que o número de retransmissões cresce muito rápido, conforme mostrado no gráfico abaixo. O TOUT também foi mantido o mesmo (no valor de 10) em todos os testes. Neste teste, variamos apenas a probabilidade de erro do cliente.



Conclusão

O desenvolvimento do projeto ocorreu sem maiores problemas. O maior desafio encontrado foi em relação à implementação da janela deslizante e seu controle. Na teoria seu funcionamento é bem simples, mas quando fomos implementar encontramos dificuldades em manter o sincronismo e movimentar a janela de acordo. Mas no final tudo ficou implementado corretamente.