Socket Programming

Bruno José Bergamaschi Kumer Reis - 14/0017666
Victor Zaffalon Marra - 13/0136760
Universidade de Brasilia - Professor Jacir L. Bordim
Email : b.reis03@yahoo.com.br , zaffalonvictor@gmail.com

***Abstract*— O artigo visa mostrar como realizar a garantia de uma entrega confiável e ordenada de dados no protocolo UDP, bem como acrescentar um controle de fluxo ao protocolo e também o pipeline. Também visa demonstrar a implementação do algoritmo go-back-n.**

I. INTRODUÇÃO

A. Algoritmo Go-back-N

O algoritmo Go-back-N consiste em um protocolo que utiliza o pipeline no envio de informações entre um servidor e um cliente. Um protocolo que utiliza Pipeline é aquele que permite que o host remetente envie múltiplos pacotes, ainda em transição, que precisam ser reconhecidos.

O host remetente pode enviar até N (valor definido pelo programador/limitações da rede) pacotes não reconhecidos em uma Pipeline. O host destinatário só envia sinais que sejam cumulativos confirmando o reconhecimento, em outras palavras, não reconhece um pacote se existe um espaço de pacote não reconhecido entre ele e o último que tenha sido reconhecido. O remetente tem um temporizador para o pacote mais antigo que não foi reconhecido, quando esse temporizador de timeout expira, o host remetente reenvia todos os pacotes não reconhecidos.

II. ANÁLISE DO ITEM (A)

A. Como garantimos a confiabilidade da entrega dos dados e de forma ordenada

A confiabilidade da entrega foi garantinda pelo uso do algoritimo de go-back-n, quando uma mensagem é recebida pelo servidor uma resposta ack é criada e enviada de volta ao cliente contendo o conteudo da mensagem e o número de sequencia daquela mensagem. No nosso programa o número de sequência é responsavel por simular se a perda de um pacote ocorreu no programa ou se os pacotes chegaram no servidor na ordem incorreta. O valor do campo sequenceNumber da classe ack enviada de volta pela conexão em socket recebe o valor -1 para o caso em que o pacote deve ser considerado como perdido no caminho.

Quando o cliente recebe um ack com o valor de sequenceNumber igual a -1 ou diferente do sequenceNumber esperado todas as mensagens contidas na janela nesse momento são reenviadas e aguardam novamente o recebimento dos acks de forma a ter certeza que os dados estão sendo corretamente recebidos pelo servidor.

O valor do sequenceNumber do ack tambem pode conter um valor diferente do valor de sequenceNumber esperado para aquela mensagem, como nosso programa guarda esse valor esperado podemos então identificar quando uma mensagem chegou em ordem incorreta no servidor pois um sequenceNumber incorreto sera atribuido ao ack. Assim como no caso da perda de ack toda a janela deve ser reenviada para garantir a entrega dos pacotes na ordem correta.

B. Como fizemos o controle de fluxo no programa

Na simulação o controle de fluxo do programa foi feito atrasando a execução do envio de mensagens utilizando um timer no caso em que o servidor esta recebendo mensagens a uma taxa maior do que a taxa maxima definida.

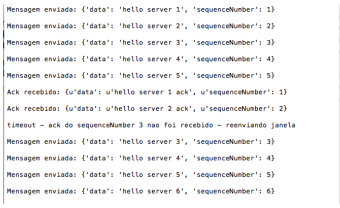
C. Como implementamos o pipeline no código

A implementação do go-back-n garante o pipeline porque o envio de varios pacotes de uma vez utilizando a janela e a consequente espera por varios acks de retorno se caracterizam como um pipeline.

III. ANÁLISE DO ITEM (B)

A. Exemplos de execução onde há perda de dados

Execução do cliente e servidor respectivamente:



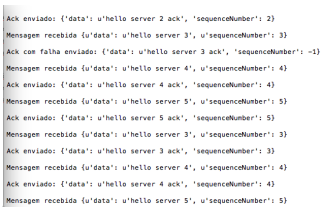
```
Mensagem enviada: {'data': 'hello server 1', 'sequenceNumber': 1}
Mensagem enviada: {'data': 'hello server 2', 'sequenceNumber': 2}
Mensagem enviada: {'data': 'hello server 3', 'sequenceNumber': 3}
Mensagem enviada: {'data': 'hello server 4', 'sequenceNumber': 4}
Mensagem enviada: {'data': 'hello server 5', 'sequenceNumber': 5}
Ack recebido: {u'data': u'hello server 1 ack', u'sequenceNumber': 1}
Ack recebido: {u'data': u'hello server 2 ack', u'sequenceNumber': 2}
timeout - ack do sequenceNumber 3 nao foi recebido - reenviando janela
Mensagem enviada: {'data': 'hello server 3', 'sequenceNumber': 3}
Mensagem enviada: {'data': 'hello server 4', 'sequenceNumber': 4}
Mensagem enviada: {'data': 'hello server 5', 'sequenceNumber': 5}
Mensagem enviada: {'data': 'hello server 6', 'sequenceNumber': 6}
```

```
Mensagem recebida {u'data': u'hello server 1', u'sequenceNumber': 1}
Ack enviado: {'data': u'hello server 1 ack', 'sequenceNumber': 1}
Mensagem recebida {u'data': u'hello server 2', u'sequenceNumber': 2}
Ack enviado: {'data': u'hello server 2 ack', 'sequenceNumber': 2}
Mensagem recebida {u'data': u'hello server 3', u'sequenceNumber': 3}
Ack com falha enviado: {'data': u'hello server 3 ack', 'sequenceNumber': -1}
Mensagem recebida {u'data': u'hello server 4', u'sequenceNumber': 4}
Ack enviado: {'data': u'hello server 4 ack', 'sequenceNumber': 4}
Mensagem recebida {u'data': u'hello server 5', u'sequenceNumber': 5}
Ack enviado: {'data': u'hello server 5 ack', 'sequenceNumber': 5}
Mensagem recebida {u'data': u'hello server 3', u'sequenceNumber': 3}
Ack enviado: {'data': u'hello server 3 ack', 'sequenceNumber': 3}
```

B. Exemplos de execução onde há dados corrompidos

Execução do cliente e servidor respectivamente:

```
Mensagem enviada: {'data': 'hello server 4', 'sequenceNumber': 4}
Mensagem enviada: {'data': 'hello server 5', 'sequenceNumber': 5}
Mensagem enviada: {'data': 'hello server 6', 'sequenceNumber': 6}
Mensagem enviada: {'data': 'hello server 7', 'sequenceNumber': 7}
Ack recebido: {u'data': u'hello server 3 ack', u'sequenceNumber': 3}
Ack recebido: {u'data': u'hello server 4 ack', u'sequenceNumber': 4}
numero de sequencia errado - reenviando janela
Mensagem enviada: {'data': 'hello server 5', 'sequenceNumber': 5}
Mensagem enviada: {'data': 'hello server 6', 'sequenceNumber': 6}
Mensagem enviada: {'data': 'hello server 7', 'sequenceNumber': 7}
Mensagem enviada: {'data': 'hello server 8', 'sequenceNumber': 8}
Mensagem enviada: {'data': 'hello server 9', 'sequenceNumber': 9}
```



```
Ack enviado: {'data': u'hello server 2 ack', 'sequenceNumber': 2}
Mensagem recebida {u'data': u'hello server 3', u'sequenceNumber': 3}
Ack com falha enviado: {'data': u'hello server 3 ack', 'sequenceNumber': -1}
Mensagem recebida {u'data': u'hello server 4', u'sequenceNumber': 4}
Ack enviado: {'data': u'hello server 4 ack', 'sequenceNumber': 4}
Mensagem recebida {u'data': u'hello server 5', u'sequenceNumber': 5}
Ack enviado: {'data': u'hello server 5 ack', 'sequenceNumber': 5}
Mensagem recebida {u'data': u'hello server 3', u'sequenceNumber': 3}
Ack enviado: {'data': u'hello server 3 ack', 'sequenceNumber': 3}
Mensagem recebida {u'data': u'hello server 4', u'sequenceNumber': 4}
Ack enviado: {'data': u'hello server 4 ack', 'sequenceNumber': 4}
Mensagem recebida {u'data': u'hello server 5', u'sequenceNumber': 5}
```

IV. ANÁLISE DO ITEM (C)

A. Informações de desempenho do programa em termos de vazão

Logo com uma internet de 1GB obtemos uma vazão de 30kbps.

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

V. CONCLUSÃO

Podemos perceber que a implementação do go-back-n em um modelo que usa o protocolo UDP como sua forma de conexão permite fornecer ao UDP caracteristicas do protocolo TCP como confiabilidade, entrega em ordem dos pacotes e controle de fluxo.

REFERENCES

[1] Problem Solving with Algorithms and Data Structures using Python. Disponível em : http://interactivepython.org/runestone/static/pythonds/index.html. Acesso feito no dia 29 de março de 2017.

[2] Kurose, James & Ross, Keith et. al. Redes de Computadores e a Internet (Uma Abordagem Top Down). 3ª Edição. Addison-Wesley;