Universidade Federal de Minas Gerais
TP2 – Redes de Computadores
Alunos: Enzo Roiz e Juan Oliveira

1 – Introdução

Este trabalho visa a implementação de um serviço de transmissão confiável utilizando UDP como protocolo de transporte. Porém como é sabido, UDP não oferece serviço confiável, podendo não entregar pacotes em ordem, cabendo a nós fazê-lo. Para isto será implementado um protocolo de janela deslizante oferecendo três tipos de transmissão: *Stop and Wait, Go Back N e Selective ACK*, fazendo com que a transmissão se dê de forma confiável e ordenada. Para a transmissão o cliente enviará o conteúdo de um arquivo, passado como parâmetro, para o servidor que receberá este arquivo e armazenará no disco. Detalhes sobre cada tipo de transmissão serão abordados nos tópicos a seguir.

2 – Desenvolvimento

2.1 – Descrição do protocolo

2.1.1 – Formato dos pacotes

Visando não ultrapassar o MTU de uma rede Ethernet estabeleceu-se tamanhos de pacotes com no máximo 1500 bytes cada. Sendo assim, cada mensagem não deve conter mais do que 1460 bytes de dados, uma vez que os cabeçalhos IP e UDP contém 20 bytes cada. Além dos cabeçalhos IP e UDP foi também implementado um cabeçalho próprio do protocolo a fim de auxiliar no recebimento de mensagens em ordem, na verificação de erros durante o transporte da mensagem, por meio de checksum, e no reenvio de mensagens que por ventura não tenha chegado de forma correta. Sendo assim, os pacotes têm o seguinte formato:

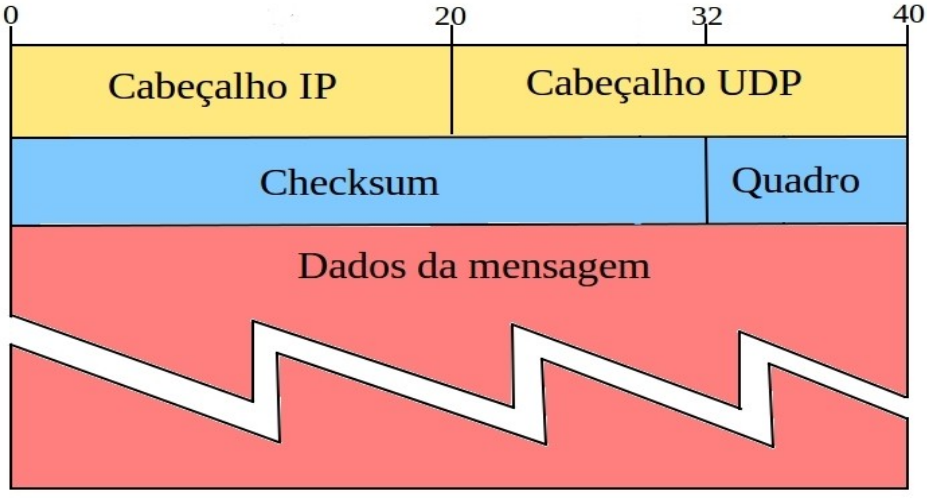

Figura 1 – Cabeçalho do pacote

O cabeçalho do pacote é formado por Checksum (32 bytes) e Quadro (8 bytes). No campo de checksum tem-se uma string de 32 caracteres que são a representação sob a forma hexadecimal do resultado gerado pelo algoritmo MD5 para a mensagem que está sendo enviada. Já no campo quadro, tem-se uma string de 8 caracteres representando o número do quadro ao qual a mensagem se refere. Após o cabeçalho, no caso do cliente, vêm até 1420 bytes do conteúdo do arquivo transmitido, completando os 1500 bytes do pacote Ethernet. Já no caso do servidor, não há dados a serem enviados além do próprio cabeçalho.

2.1.2 – Funcionamento

2.1.2.1 – Cliente

No lado do cliente existem duas threads funcionando sob a forma “produtor/consumidor”. A thread responsável pela leitura do arquivo é a produtora, enquanto a thread responsável pelo envio das mensagens ao servidor é a consumidora. Cada thread tem a sua “vez” de executar, pois ambas acessam um mesmo recurso, neste caso um buffer circular utilizado para implementar a janela deslizante, onde são colocadas partes lidas do arquivo. Como visto anteriormente, as mensagens deste protocolo transportam até 1420 bytes de dados, logo cada parte do buffer, que tem tamanho igual ao da janela deslizante, armazena uma parte do arquivo com 1420 bytes. Mais a frente será discutido como foram feitos os cálculos para se obter o tamanho da janela deslizante.

A thread de leitura do arquivo começa com a “vez” de execução. Ela verifica o buffer e, enquanto este tiver espaço, ou seja, sua primeira posição estiver vazia, a thread lê 1420 bytes do arquivo e armazena no buffer, retirando a primeira posição vazia e acrescentando uma posição ao fim do buffer, contendo os dados lidos. Quando o buffer enche, ou seja, primeira posição dele não é mais vazia, a thread passa a vez para a thread de envio e espera para ser chamada novamente.

Ao receber a vez de execução, pela primeira vez, a thread de envio manda uma mensagem inicial ao servidor, contendo as informações do checksum do arquivo que está sendo enviado e do tipo de transmissão que será feito, *Stop and Wait, Go Back N* ou *Selective ACK*. Após receber a mensagem do servidor confirmando a transmissão, o cliente envia todas as mensagens contidas no buffer, ou seja, no primeiro envio o cliente envia toda a janela sem esperar por nenhuma resposta do servidor. Cada tipo de transmissão trata de modo diferente a liberação de espaço no buffer à medida que as respostas vão sendo recebidas, como será visto no próximo tópico, mas o funcionamento da thread de envio é igual para todos. Ao se receber a resposta do servidor confirmando o recebimento do primeiro quadro da janela, é liberado espaço, colocando a primeira posição do buffer circular como vazia. Quando a thread de envio vê que a primeira posição está vazia ela então retorna a vez da execução para a thread de leitura, para que esta encha o buffer, e espera para ser chamada novamente. O processo ocorre sucessivamente até que todo o arquivo seja enviado.

2.1.2.2 – Servidor

No lado do servidor a ideia é a mesma do lado do cliente. Existem duas threads funcionando como produtor/consumidor. A thread que recebe as mensagens é a produtora, enquanto a thread que escreve os dados no arquivo é a consumidora. De modo análogo, como as threads acessam um mesmo recurso compartilhado, um buffer circular de tamanho igual ao da janela deslizante, o acesso a este recurso tem que ser coordenado e assim cada thread executa quando for a sua vez.

A thread de recebimento de mensagens é a que começa executando, enquanto a thread de escrita no arquivo espera ser chamada. Ela espera então por mensagens que chegam numa porta determinada. A primeira mensagem, como já visto, traz informações sobre o checksum do arquivo enviado e o tipo de transmissão a ser realizado. O servidor armazena estas informações e envia resposta confirmando o início da transmissão. Após isto feito as mensagens que chegam ao servidor são as mensagens contendo partes lidas do arquivo. A cada mensagem que chega o servidor a armazena no buffer e de forma igual ao que acontece na thread de envio do lado do cliente, enquanto o buffer estiver vazio, ou seja, sua primeira posição for vazia, ele continua a armazenar mensagens. Quando o buffer enche, tem a sua primeira posição com dados do arquivo enviado, a thread de recebimento passa a vez para a thread de escrita do arquivo.

A thread de escrita, ao executar pela primeira vez, cria o arquivo de saída caso este não exista ou limpa o arquivo caso já exista. Enquanto houver dados a serem escritos, ou seja, enquanto a primeira posição do buffer não for vazia, a thread escreve o dado da primeira posição no arquivo, o retira do buffer e acrescenta uma posição vazia ao final do buffer. Assim, após escrever todo o buffer, a thread de escrita irá se deparar com uma posição de buffer vazia. Quando isto ocorre a

thread de escrita repassa a vez da execução para a thread de recebimento de mensagens, para que esta encha o buffer, e espera para que seja chamada novamente. Este processo ocorre até que todo o arquivo tenha sido escrito.

2.1.3 – Temporização

Como este trabalho é executado sobre uma rede não confiável, que perde pacotes, o protocolo implementado faz uso de temporizações, no lado do cliente, para verificar se houve perda de algum pacote durante a troca de mensagens entre cliente e servidor. Quando a temporização dispara, ou seja, quando é percebida a perda de um pacote, o cliente reenvia a(s) mensagem(ns) de acordo com o tipo de transmissão que está sendo feita.

O *time* out, tempo para disparar a temporização, adotado para este protocolo foi de 2 segundos. Como a maior latência, na topologia utilizada neste trabalho, para a transmissão é de cerca de 500 milissegundos, este tempo de time out garante que a temporização não irá disparar no meio de uma transmissão correta e, para caso de estudo, nos permite ver o atraso que ocorre em caso de perdas de pacotes, mas sem afetar consideravelmente o desempenho do protocolo.

2.1.4 – Tipos de transmissão

2.1.4.1 – Stop And Wait

Na transmissão Stop And Wait a cada mensagem enviada pelo cliente ao servidor, ele espera a confirmação do recebimento desta mensagem para enviar a próxima mensagem. Com isto a janela de transmissão tem apenas uma posição tanto do lado do cliente, quanto do servidor.

2.1.4.2 – Go Back N

A transmissão Go Back N é mais sofisticada que a Stop And Wait por ter tamanhos de janela maiores do que uma posição no lado do cliente. Assim, são enviadas várias mensagens sem que se precise esperar a resposta do servidor. Porém do lado do servidor a janela de recepção tem apenas uma posição, ou seja, neste tipo de transmissão o servidor somente aceita receber a mensagem se ela for o próximo quadro esperado. Caso não seja, o receptor entende como um quadro não esperado e não recebe o pacote.

2.1.4.3 – Selective ACK

Assim como Go Back N, Selective ACK tem tamanhos de janela maiores do que uma posição no lado do cliente. Porém a diferença entre estes dois tipos de transmissão é que no lado do servidor a janela de recepção pode receber todos os quadros enviados pela janela de transmissão. Isto faz com que independentemente da ordem enviada pelo cliente, o receptor possa armazenar os quadros recebidos mesmo que estes não sejam os quadros esperados, ao contrário do que ocorre em Go Back N, onde os quadros que não são esperados são simplesmente descartados.

2.1.5 – Liberação de espaço no buffer de envio

2.1.5.1 – Stop And Wait

Quando o cliente recebe a confirmação do recebimento da mensagem esta é necessariamente a confirmação esperada, pois neste tipo de transmissão envia-se um quadro por vez. Assim, o buffer, de uma posição, é esvaziado e caso ainda tenha informações a serem lidas do arquivo, estas são lidas e armazenadas no buffer para serem enviadas ao servidor.

2.1.5.2 – Go Back N e Selective ACK

Como em Go Back N e Selective ACK a janela de transmissão tem mais de uma posição e UDP pode enviar pacotes fora de ordem, podem ocorrer casos em que o ACK recebido não é necessariamente o ACK esperado. Então quando o cliente recebe um ACK ele olha em que posição do buffer está a mensagem que este confirma o recebimento e libera o espaço daquela posição. Após isto, verifica se o ACK recebido é o ACK esperado. Se sim, caso tenha informações a serem lidas do arquivo, estas são lidas e armazenadas no buffer até que este encha novamente para que seja feito o envio das mensagens ao servidor.

2.1.6 – Reenvio de pacotes

2.1.6.1 – Stop And Wait

Quando a temporização dispara, como a janela de transmissão tem apenas uma posição, a mensagem cujo recebimento não foi confirmado é reenviada.

2.1.6.2 – Go Back N

Como neste caso a janela de transmissão é maior do que uma posição e a janela de recepção tem apenas uma posição, ou seja, não armazena quadros recebidos, quando a temporização dispara o cliente reenvia todos os pacotes contidos na janela de transmissão. Com isto, além do pacote perdido na transmissão, o cliente reenvia todos os demais pacotes da janela.

2.1.6.3 – Selective ACK

Como dito anteriormente a diferença entre Selective ACK e Go Back N é que o receptor é capaz de armazenar os pacotes que chegam fora de ordem. Assim quando a temporização dispara o cliente reenvia apenas o pacote perdido na transmissão, sem reenviar todos os demais pacotes contidos na janela, uma vez que estes podem ter sido corretamente recebidos pelo servidor.

2.1.6.4 – Servidor

No servidor pode ocorrer casos em que o pacote enviado pelo cliente chega corretamente, porém o ACK enviado pelo servidor é perdido e o cliente não sabe que a mensagem chegou sem erros e reenvia a mensagem. Neste caso como o servidor já possui os dados da mensagem não há necessidade destes serem armazenados novamente, então o ACK não recebido pelo cliente é reenviado e a transmissão continua de maneira normal.

2.2 – Decisões de implementação

2.2.1 – Início e fim de conexão

Como citado anteriormente, o início da conexão se dá quando a thread responsável pelo envio manda uma mensagem inicial ao servidor contendo o checksum do arquivo que irá ser enviado e o tipo de transmissão que será feita. Após receber a resposta do servidor, o cliente envia então os dados do arquivo.

No lado do cliente para encerrar a conexão foi criado um método que verifica se o envio está sendo realizado ou não. Quando é enviado o último pacote para o servidor, este método retorna que o envio foi finalizado. A thread de leitura então é finalizada pois não tem que ler dados do arquivo, após isto a thread de envio é finalizada e a conexão encerrada.

No lado do servidor, de modo análogo, um método foi criado para verificar se este estava

recebendo mensagens, porém isto foi feito associado com temporização. A temporização adotada foi de 4 segundos, maior do que o time out adotado pelo cliente. Com isto, após 4 segundos de inatividade, ou seja, sem que o cliente envie nenhum pacote, o servidor “entende” que a conexão foi encerrada. Assim o método implementado retorna que o recebimento de pacotes foi finalizado. A thread que recebe pacotes é finalizada e após isto a thread que escreve no arquivo também é finalizada, encerrando a conexão em ambos os lados.

2.2.2 – Implementação de time out

Antes, neste trabalho, foi falado sobre temporizações, porém em *Python* isto pode ser feito de várias formas. Seja utilizando *select*, sinais ou o time out do próprio *socket* através da função *settimeout(timeout).*

Neste trabalho optou-se por utilizar o parâmetro de temporização do select. E para o seu uso basta passar três listas , neste caso, de sockets para serem avaliados pelo select da seguinte forma.

entrada, saida, excecao = select.select([socket], [], [], timeout)

A primeira lista verifica o recebimento de dados a serem lidos, a segunda lista contém objetos que receberão dados e a terceira lista recebe objetos que possam gerar erros. O quarto parâmetro é o time out, descrito anteriormente. Este método retorna três novas listas. Como estamos interessados apenas nas mensagens a serem recebidas do servidor, passamos a lista com o socket do cliente apenas para o primeiro parâmetro e manipularemos apenas a lista “entrada”, que contém os dados recebidos através do socket.

Com isto basta apenas verificar se “entrada” é vazia ou não, da seguinte forma:

```
if entrada:
        # Existem dados enviados pelo servidor.
        # Manipula os dados recebidos.
else:
        # Dado não foi recebido dentro do tempo estipulado. Temporização dispara.
        # Reenvio de dados ao servidor.
```

Logo se existirem dados em “entrada” estes são manipulados. Porém se entrada não contiver dados, a temporização dispara e é feito reenvio de dados.

2.2.3 – Simulação de perda de pacotes

A “perda de pacotes” neste trabalho é simulada tanto pelo cliente, quanto pelo servidor. Isto é feito por uma função que, após o cliente ou servidor receber uma mensagem, sorteia de acordo com a taxa de perda de pacotes estipulada se aquela mensagem “realmente chegou” ou se ela “se perdeu” no meio da transmissão.

2.2.4 – Checksum

A fim de garantir que as mensagens recebidas chegam com os mesmos dados de quando são transmitidas, foi adotado o uso de checksum. O checksum é uma string de 32 bytes, sob a forma hexadecimal, do resultado gerado pelo algortitmo MD5 para determinado dado. Neste protocolo se faz uso do checksum para duas finalidades diferentes. Na primeira delas o MD5 é gerado no lado do cliente para o arquivo a ser enviado como um todo e enviado junto à mensagem inicial de conexão. Assim no fim da transmissão é possível comparar o checksum do arquivo original, no lado do cliente, com o que foi recebido pelo servidor. Para a outra finalidade, cada mensagem leva consigo um checksum dos dados enviados. E de forma semelhante ao que acontece com o arquivo enviado,

quem recebe a mensagem faz a comparação do checksum que vai junto com a mensagem com o checksum gerado pelo algoritmo MD5 para os dados da mensagem.

2.2.5 – Uso de buffers e auxiliares

2.2.5.1 – Cliente

Para a implementação da janela deslizante, no lado do cliente além do buffer que contém os dados lidos do arquivo, outros dois buffers circulares auxiliam na transmissão correta dos dados para o servidor.

Um deles, *ACKsRecebidos*, faz o controle das respostas recebidas pelo cliente verificando o *LAR*, último quadro recebido. Ele tem duas vezes o tamanho da janela deslizante e mantém o número dos quadros das mensagens cujo recebimento foi confirmado pelo servidor. Como ACKs podem ser recebidos fora de ordem devido ao uso de UDP como protocolo de transporte, um tamanho de buffer duas vezes maior que o da janela deslizante garante que o *LAR* esteja sempre neste buffer, garantindo um controle correto dos ACKs recebidos.

O outro deles, *mensagensEnviadas*, guarda uma cópia das mensagens enviadas ao servidor e tem o mesmo tamanho da janela deslizante. Quando há a necessidade de se fazer o reenvio de uma mensagem, é então enviada a cópia desta mensagem que está armazenada neste buffer.

2.2.5.2 – Servidor

Do lado do servidor, além do buffer utilizado para armazenar as mensagens recebidas do cliente, outro buffer circular ajuda no recebimento das mensagens.

Este buffer, *mensagensRecebidas*, controla as mensagens recebidas pelo servidor observando o *NFE*, próximo quadro esperado. Ele tem também duas vezes o tamanho da janela deslizante, e de forma análoga ao *ACKsRecebidos*, do cliente, mantém o número dos quadros recebidos pelo servidor e garante que o NFE sempre estará neste buffer. Assim se tem também um controle correto das mensagens recebidas.

2.2.6 – Recebimento dos dados em ordem

Como UDP não oferece serviço confiável, podendo entregar pacotes fora de ordem, coube a nós garantirmos o recebimento dos pacotes de forma ordenada.

Para transmissões Stop And Wait e Go Back N como a janela de recepção tem apenas uma posição e só é recebido o próximo quadro esperado, a recepção sempre se dá de forma ordenada.

Para transmissões Selective ACK isto não ocorre, uma vez que, caso o quadro recebido não seja o quadro esperado, o servidor armazena este quadro. Neste caso o que foi feito para se manter o recebimento de forma ordenada foi inserir o quadro recebido na posição correta considerando o número do quadro recebido e o NFE, próximo quadro esperado. Em um pseudo-código teríamos:

```
# Enquanto servidor receber mensagens ...

posicaoNoBuffer = numQuadro - NFE
insereNoBuffer(posicao, mensagem) # Coloca a mensagem no buffer

atualizaNFE() # Verifica qual é o próximo quadro esperado
```

A seguir uma simulação do recebimento dos quadros 1, 3 e 6 pelo servidor para uma janela deslizante com 5 posições.

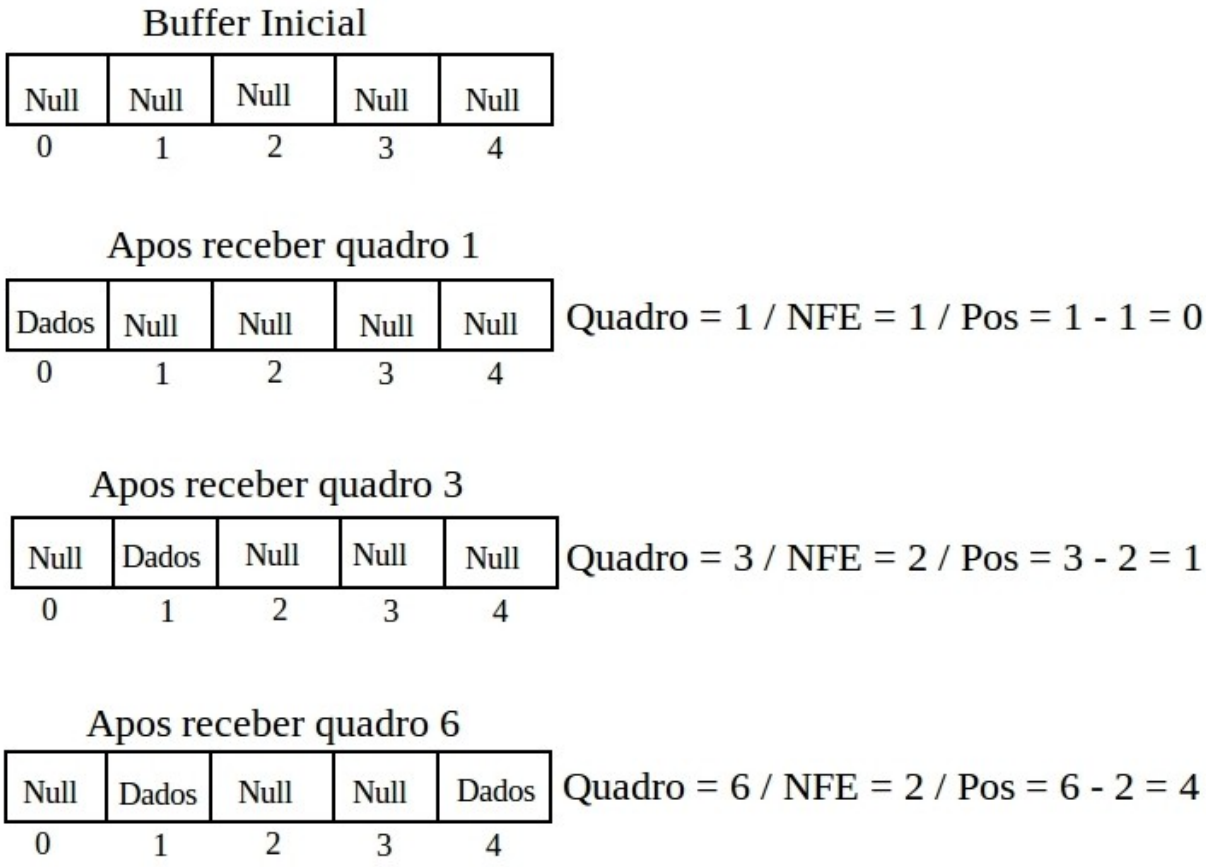

Figura 2 – Simulação de recebimento de mensagens

Relembrando que quando a primeira posição é preenchida a thread de escrita escreve os dados no arquivo e esvazia as posições escritas do buffer. Assim, independente do quadro recebido ele sempre será inserido em uma posição vazia no buffer e na sua posição correta, garantindo o recebimento das mensagens e consequentemente a escrita no arquivo de saída de forma ordenada.

3 – Experimentos

Para este trabalho será utilizada uma topologia com dois hosts e dois switches entre estes hosts. O link dos hosts para os switches terão 1 Mbps de banda e atraso de 10 ms. Já o link entre os dois switches terá uma banda menor, de 100 Kbps. Os testes serão realizados variando o atraso entre o link dos switches de 10 ms a 500 ms. Além destes testes se manterá o link entre os switches com atraso de 100 ms, variando o tamanho da janela deslizante. Será medido o tempo gasto para o envio de um arquivo de cerca de 1MB, bem como o throughput obtido.

3.1 – Tamanho da janela deslizante

Os tamanhos das janela deslizantes foram estimados de modo que pudessem armazenar um número considerável de pacotes a serem transmitidos. Foi observado também a capacidade da rede, levando-se em conta o atraso e a banda apresentados pela topologia utilizada. Com isto, para os experimentos os tamanhos de janela vão até 40 posições, contendo até cerca de 60KB de dados do arquivo, 40 * 1500 B (tamanho do pacote).

3.2 – Experimentos comparativos

3.2.1 Tamanho de janela fixo

Para estes experimentos, fixou-se o tamanho da janela deslizante em 20 posições, cerca de 30KB por janela, variando-se o atraso do link entre os switches de 10ms a 500ms. Primeiro serão considerados envios sem perdas de pacote e após, envios com perdas de pacotes de 5% no cliente e no servidor.

3.2.1.1 Tempos de envios sem perdas de pacotes

O gráfico abaixo foi gerado a partir dos tempos coletados para envios sem perdas de pacotes. Através dele podemos perceber que o atraso influenciou apenas no tipo de transmissão Stop And Wait, que variou de cerca de 84 segundos, para atraso de 10 ms no link entre switches, a 754 segundos, para atraso de 500 ms no link entre switches. Para melhor visibilidade do gráfico este último tempo, de 754 segundos, foi ocultado.

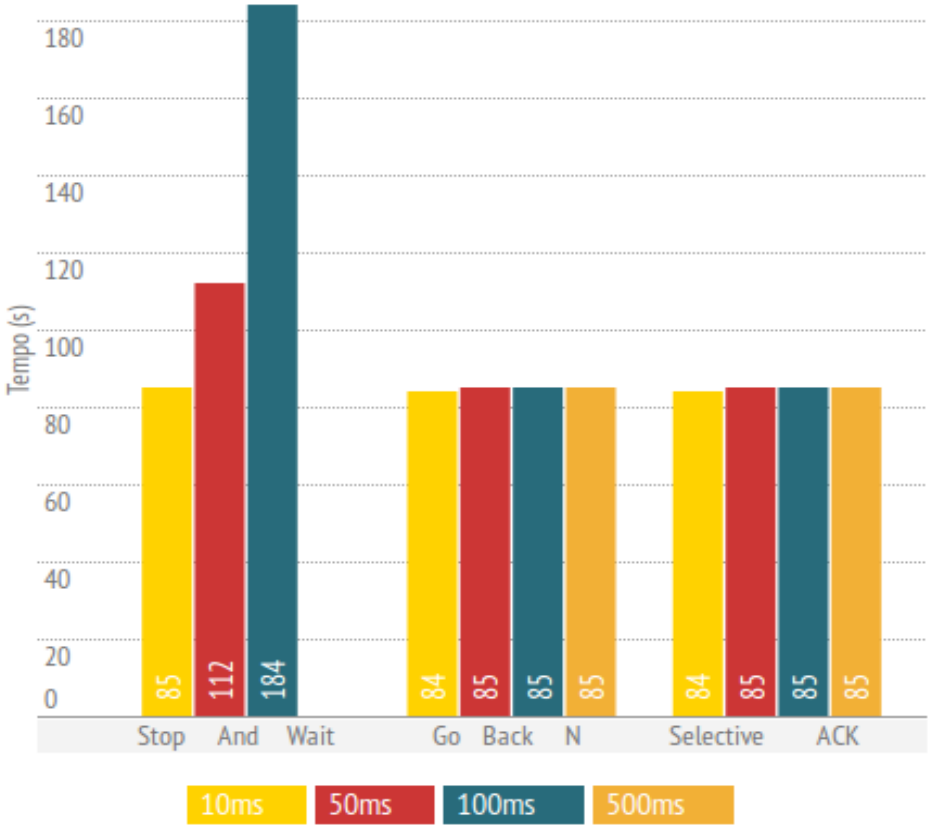

Figura 3 – Tempos para envios sem perdas de pacote

O fato de o tempo variar apenas para Stop And Wait é devido a neste tipo de transmissão o canal só ser ocupado pela mensagem enviada pelo cliente ou pelo ACK enviado pelo servidor, pois o transmissor tem que esperar pelo ACK do receptor para poder enviar nova mensagem. Nos outros tipos de transmissão o canal é ocupado por mais mensagens, não ficando ocioso e consequentemente o aproveitando de maneira mais efetiva, uma vez que várias mensagens do cliente e/ou do servidor estão trafegando ao mesmo tempo. Com isto pôde-se ver que apesar da variação no atraso, o tempo de transmissão sofreu leve variação. Podemos observar também o comportamento semelhante entre Go Back N e Selective ACK, pois, como não ocorreram perdas de pacotes e, neste caso, o recebimento de pacotes se deu de forma ordenada, estes dois tipos de transmissão se comportaram da mesma maneira.

3.2.1.2 Throughput de envios sem perdas de pacotes

A seguir o gráfico mostra o thoughput (em Kbps) obtido para o experimento sem perdas de pacotes. E como visto ao analisar os tempos de transmissão, a maior variação se deu para Stop And Wait, que teve variações de 10 Kbps a 94 Kbps. Isto ocorre pois, para o envio de um mesmo arquivo o tempo de transmissão aumentou, e como consequência o throughput diminui, pois menos dados são enviados por unidade de tempo.

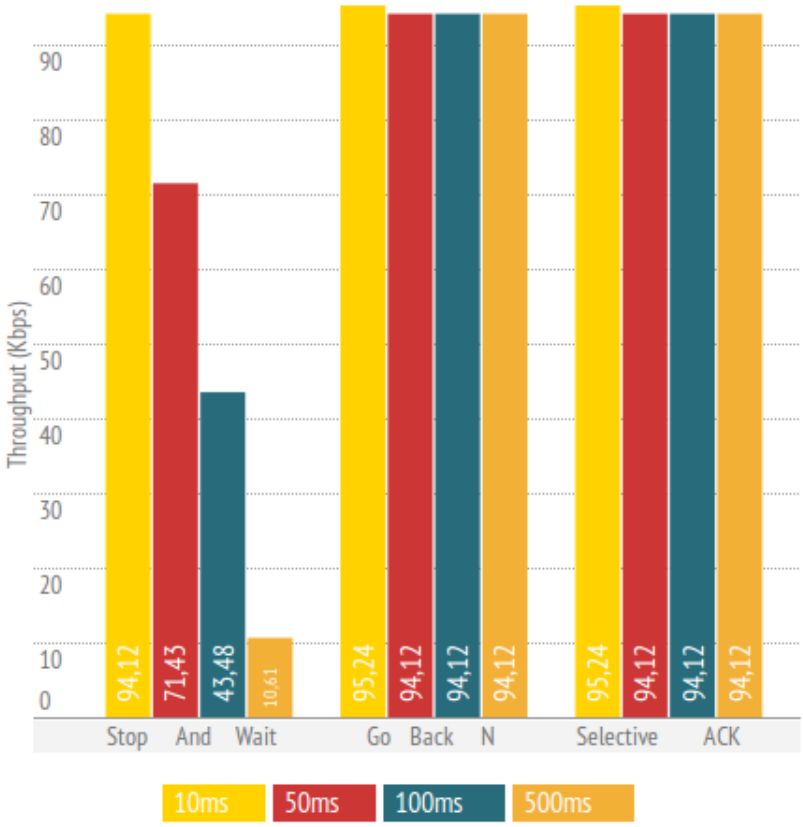

Figura 4 – Throughput para envios sem perdas de pacotes

3.2.1.3 Tempos de envios com perdas de pacotes

O gráfico a seguir foi gerado a partir dos tempos coletados para envios com perdas de pacotes de 5% no cliente e 5% no servidor. Foram ocultados dos gráficos os tempos para atraso no link entre switches de 500 ms para melhor visibilidade, já que estes tempos chegaram a 14 minutos.

Novamente podemos perceber a disparidade entre os tempos para Stop And Wait, com tempos de 293 a 477 segundos. Do mesmo modo do citado anteriormente, isto é devido ao fato de ter que esperar o ACK do servidor para enviar outra mensagem. Para Go Back N e Selective ACK houve variações entre os tempos de acordo com o atraso, mas estas não foram tão grandes quanto para Stop And Wait. Para Go Back N esta variação foi de cerca de 5 segundos e para Selective ACK, de aproximadamente 30 segundos. Também podemos observar através do gráfico, comparando com os resultados de envio sem perda de pacotes, como a perda de pacotes afeta significativamente o tempo de transmissão. Isto é devido ao disparo de temporizações para alertar das perdas de pacotes que ocorreram e ao tempo de se reenviar os pacotes e receber o seu ACK.

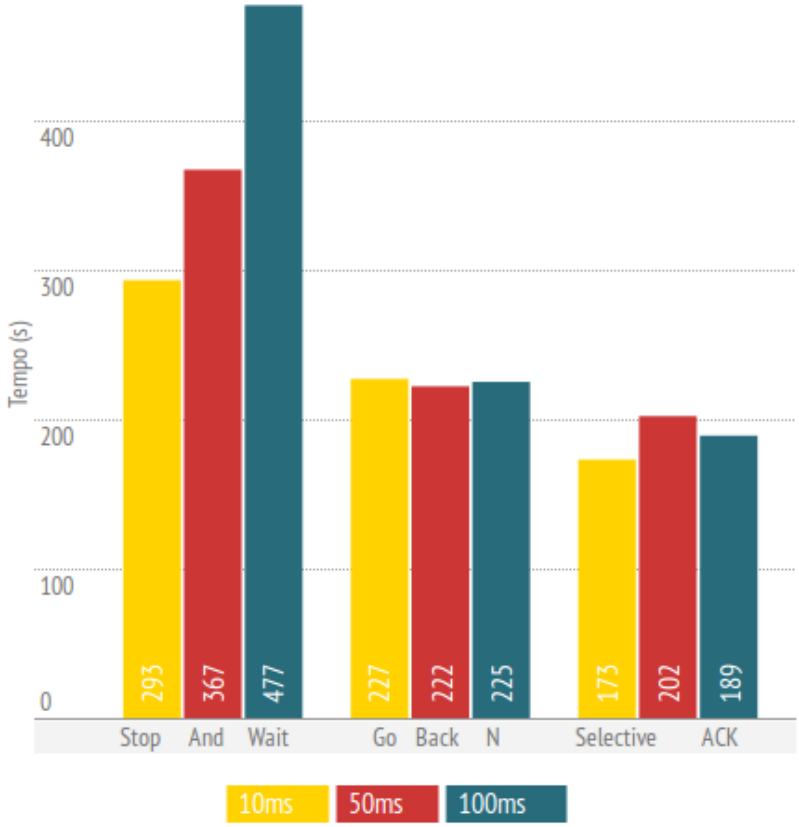

Figura 5 – Tempos para envios com perdas de pacotes

Analogamente ao que ocorreu no envio sem perdas de pacotes, aqui o atraso entre os links tem maior influência sobre Stop And Wait, já que para os outros tipos de transmissão o canal será utilizado de forma mais efetiva, com trânsito vários pacotes ao mesmo tempo, com isso o tempo de transmissão não é tão sensível ao atraso entre os links. Abaixo tabelas de quadro reenviados para cada tipo de transmissão, tanto pelo cliente, quanto pelo servidor.

Pacotes reenviados - Cliente			
	10ms	**50ms**	**100ms**
Stop And Wait	79	83	94
Go Back N	1060	1020	1040
Selective ACK	72	74	60

Tabela 1- Pacotes reenviados pelo cliente

Pacotes reenviados – Servidor			
	10ms	**50ms**	**100ms**
Stop And Wait	40	39	46
Go Back N	305	314	316
Selective ACK	28	28	35

Tabela 2 – Pacotes reenviados pelo servidor

Pelas tabelas de quadros reenviados podemos perceber que para Stop And Wait e Selective ACK o número de quadros reenviados pelo cliente foi parecido, com cerca de 85 quadros para Stop And Wait e 69 para Selective ACK em média. O mesmo aconteceu do lado do servidor, em que Stop and Wait necessitou em média de reenviar 41 ACKs, e Selective ACK reenviou cerca de 30 ACKs. Já para Go Back N o cliente reenviou 1040 quadros em média, enquanto o servidor precisou reenviar 312 ACKs em média.

Analisando cada tipo de transmissão separadamente, temos que Go Back N tem tantos reenvios devido a cada vez que a temporização dispara o cliente reenviar todos os pacotes contidos na janela. Assim a cada pacote com erro, considerando a janela com 20 posições, 20 pacotes são reenviados pelo cliente. No lado do servidor o número alto de ACKs retransmitidos acontece porque o servidor recebeu corretamente os pacotes enviados pelo cliente, porém o cliente não recebe o ACK enviado pelo servidor e reenvia toda a janela. Quando o servidor recebe pacotes que ele já contém, ele entende que o cliente não recebeu a confirmação dos mesmos e reenvia os ACKs destes pacotes. Já Stop And Wait e Selective ACK tem números menores de quadros reenviados, para ambos, cliente e servidor, por reenviarem apenas os quadros para os quais a temporização disparou, ou seja, para os quadros cujo recebimento pelo servidor não foi confirmado.

3.2.1.4 Throughput de envios com perdas de pacotes

O gráfico a seguir mostra o throughput para o experimento com perdas de pacotes. Nele podemos ver que novamente o throughput para Stop And Wait é decrescente, pois são enviados menos dados por unidade de tempo. Já para Selective ACK e Go Back N, o throughput se mantém em um nível parecido independente do atraso no link entre os switches.

Figura 6 – Throughput para envios com perdas de pacotes

3.2.2 Tamanho de janela variando

Para estes experimentos, fixou-se o atraso no link entre os switches em 100 ms, variando-se os tamanhos de janela deslizante em 5, 10, 20 e 40 posições, podendo armazenar respectivamente cerca de 7.5KB, 15KB, 30KB e 60KB. Serão considerados apenas envios com perdas de pacotes de 5% no cliente e no servidor. Além disto o tipo de transmissão Stop And Wait será desconsiderado, uma vez que sua “janela” possui apenas uma posição.

3.2.2.1 Tempos de envios com perdas de pacotes e tamanho de janela variando

O gráfico abaixo mostra os tempos obtidos para Go Back N e Selective ACK para diferentes tamanhos de janela, variando de 5 a 40 posições.

Através dele podemos perceber que para uma janela de até 20 posições os tempos de transmissão são semelhantes ficando em torno de 200 segundos. Para 40 posições ambos os tempos, tanto de Selective ACK quanto de Go Back N, aumentam, porém este último aumenta de maneira considerável em relação ao primeiro.

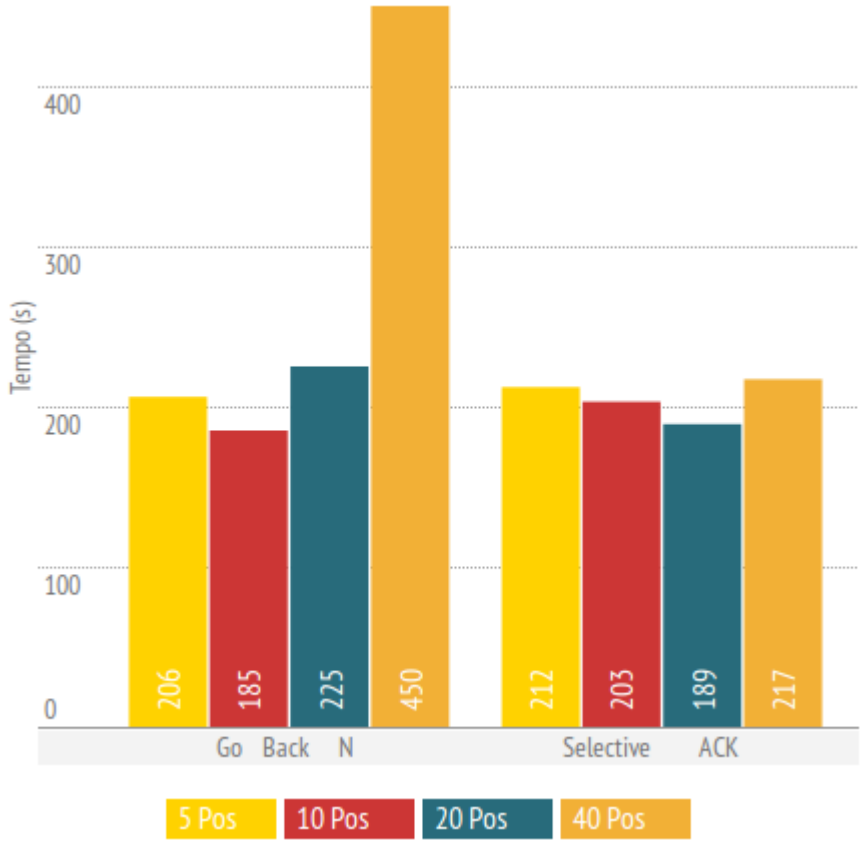

Figura 7 – Tempos para envios com perdas de pacotes e tamanhos de janelas variando

Isto porque quando um quadro é perdido, toda a janela é retransmitida e com o aumento da janela, maior é o número de quadros reenviados pelo cliente e maior a chance de perda destes pacotes reenviados. Para Selective ACK independente dos tamanhos de janela apresentados, os tempos de transmissão giram em torno de 200 segundos, mostrando que o fato de só retransmistir os pacotes cujo recebimento não foi confirmado garante a este tipo de transmissão um comportamento semelhante sob qualquer tamanho de janela. Para este experimento também foram contados os quadros reenviados por ambos, cliente e servidor. Os dados estão nas tabelas abaixo:

Pacotes reenviados – Cliente

	5 Pos	10 Pos	20 Pos	40 Pos
Go Back N	345	540	1040	3360
Selective ACK	70	67	60	76

Tabela 3- Pacotes reenviados pelo cliente

Pacotes reenviados – Servidor

	5 Pos	10 Pos	20 Pos	40 Pos
Go Back N	142	212	316	1713
Selective ACK	35	40	35	36

Tabela 4 – Pacotes reenviados pelo servidor

Através desta tabela podemos confirmar que quanto maior o número de posições na janela deslizante, maior é o número de quadros reenviados para Go Back N. Indo, no lado do cliente, de 345 quadros reenviados para uma janela de 5 posições, até 3360 quadros reenviados, para uma janela de 40 posições. O mesmo ocorre do lado do servidor, onde de 142 a 1713 quadros foram

reenviados. Já para Selective ACK, independente do tamanho da janela deslizante, o número de quadros reenviados fica entre 60 e 76 quadros no lado do cliente e entre 35 e 40 quadros no lado do servidor.

3.2.2.2 Throughput de envios com perdas de pacotes e tamanho de janela variando

A seguir o gráfico que mostra o throughput obtido para Go Back N e Selective ACK para diferentes tamanhos de janela, variando de 5 a 40 posições.

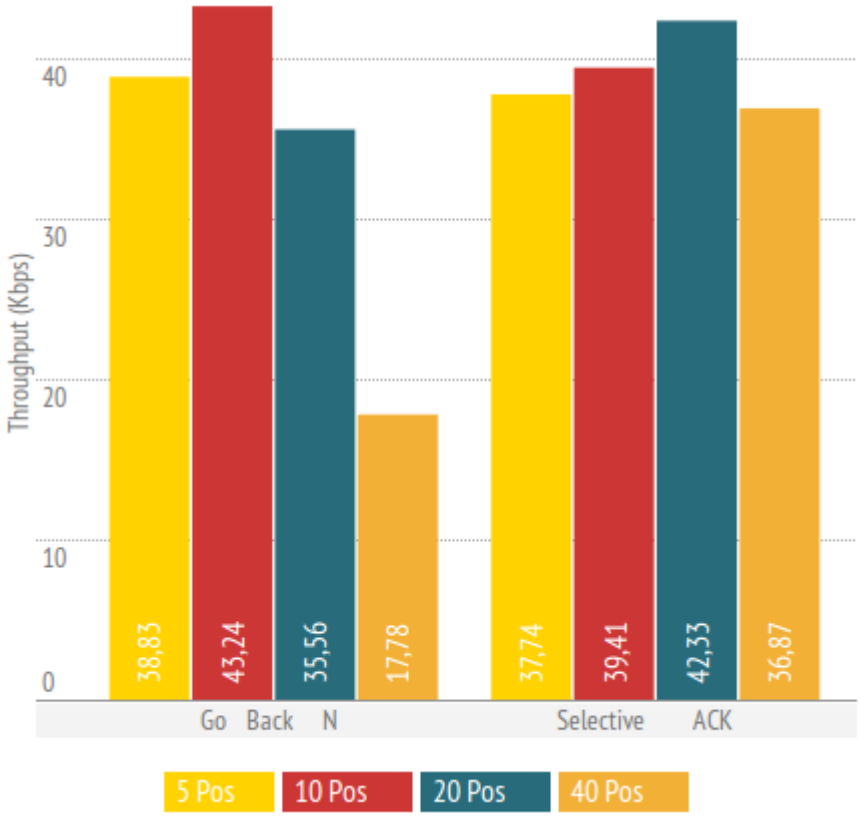

Figura 8 – Throughput para envios com perdas de pacotes e tamanhos de janelas variando

Podemos ver então que com o aumento do tamanho da janela o throughput para Go Back N é cada vez menor, pelo mesmo motivo explicado anteriormente, o aumento do tempo de transmissão. Já para Selective ACK este throughput continua praticamente o mesmo para todos os tamanhos de janela considerados.

3.3 – Análise dos resultados dos experimentos comparativos

Observando os experimentos comparativos pudemos observar uma série de fatos sobre cada tipo de transmissão, suas vantagens e desvantagens. Com isto podemos analisar cada um separadamente e observar o melhor caso em que cada um pode ser implementado.

Stop And Wait é o modo de transmissão mais simples dos observados, envia uma mensagem e só envia a próxima caso saiba a anterior ter chegado corretamente. É um método de transmissão muito simples, porém não utiliza a máxima capacidade do meio em que transmite, o que implica em envios lentos para transmissões com atrasos grandes e transmissões com perdas de pacotes, tendo um consequente throughput baixo nestas situações.

Go Back N é mais sofisticado do que Stop And Wait pois implementa janela deslizante. Com isto resolve-se o problema de baixa utilização do meio de transmissão, já que pode se ter vários pacotes em trânsito na rede ao mesmo tempo, diminuindo assim os tempos de transmissão e aumentando o throughput. Porém, neste modo de transmissão, no lado do servidor não se armazena

pacotes, pois só se recebe o pacote caso ele seja o pacote esperado, já que a janela de recepção tem apenas uma posição. Além disto caso o reenvio se faça necessário é reenviada toda a janela de transmissão e não só o pacote não recebido. Por um lado, apesar de parecer uma desvantagem, caso pacotes dentro de uma mesma janela de transmissão se percam, com um reenvio, que será de toda a janela, é possível corrigir o erro de mais de um pacote não recebido. Através dos resultados pôde-se observar também que para tamanhos grandes de janela para a rede na qual se opera, este modo de transmissao não é tão eficiente quanto para tamanhos menores, reenviando muitos quadros, tendo envio lento e throughput baixo. Fazendo ser necessário observar um tamanho adequado de janela de transmissão para que o envio se dê de forma ótima. Go Back N é então mais sofisticado e difícil de implementar do que Stop And Wait para o lado do cliente por ter janela deslizante, porém o lado do servidor continua parecido, pois para ambos a janela de recepção possui uma posição. Vale ressaltar também que ambos os modos de transmissão a recepção dos pacotes pelo servidor se dá de forma ordenada, já que sempre recebem os quadros esperados. Tendo um código para o recebimento de mensagens menos complexo.

Selective ACK assim como Go Back N tem janela de transmissão no lado do cliente, o que diferencia estes dois modos de transmissão é a janela de recepção, que para Selective ACK tem mais de uma posição. Com isto pode se receber pacotes fora de ordem, pois o servidor trata este tipo de situação. Quando o envio se dá de forma ordenada e sem perda de pacotes, Selective ACK e Go Back N tem desempenhos semelhantes, como mostrado nos gráficos de Tempo e Througput para envio sem perdas de pacotes. Ao contrário de Go Back N e assim como em Stop And Wait, caso a temporização seja disparada, pois ocorreu perda de pacotes, só é reenviado o pacote não confirmado e não a janela toda. Isto faz com que o número de pacotes reenviados seja consideravelmente menor. Além disto, pudemos notar que para todos os tamanhos de janela testados na análise comparativa, o desempenho e o número de pacotes reenviados se mantiveram semelhantes, mostrando a transmissão ter a mesma eficiência para qualquer das situações abordadadas, o que não acontece em Go Back N. Analisando desempenho, através dos gráficos é possível perceber que Selective ACK é o modo de transmissão com melhor desempenho. Ele apresenta os menores tempos e maiores throughputs tanto transmissões sem perdas de pacotes, neste caso juntamente com Go Back N, quanto para transmissões em que perdas de pacotes ocorrem. Porém Selective ACK tem código mais complexo e requer armazenamento de dados por parte do servidor devido ao fato de permitir o recebimento de mensagens fora de ordem. Assim quando um pacote que não é o esperado chega, o receptor o armazena da devida forma para manter a ordem, e espera pela próxima mensagem.

4 – Conclusão

Foi um trabalho extremamente desafiante tanto pela dificuldades encontradas na implementação dos algoritmos dos diferentes tipos de envio e recepção, quanto pelos detalhes a serem levados em consideração para que os envios e recepções se dessem de maneira correta.

Através dele pudemos aprimorar nossos conhecimentos sobre a forma que se dá a troca de mensagens entre cliente e servidor, sobre o funcionamento da janela deslizante, bem como entender de forma mais aprofundada como se dá cada um dos modos de transmissão Stop And Wait, Go Back N e Selective ACK.

Pelos experimentos e análise dos experimentos realizados pudemos ver as vantagens e desvantagens de cada um dos tipos de transmissão aprendidos e relacioná-las entre eles levando em conta fatores como perda ou não de pacotes pela rede, atraso da rede e tamanhos de janelas.

5 – Referências

Sockets em Python -
http://www.python.org.br/wiki/SocketBasico

Análise de Desempenho / Janela Deslizante -
Larry L. Peterson and Bruce S. Davie (5th Edition). *Computer Networks: A Systems Approach.*

Geração de gráficos -
http://www.infogr.am
Uso de select -
http://pymotw.com/2/select/

Uso de threads Produtor / Consumidor -
http://agiliq.com/blog/2013/10/producer-consumer-problem-in-python/

MD5 Checksum
http://stackoverflow.com/questions/16874598/how-do-i-calculate-the-md5-checksum-of-a-file-in-python

Passagem de parâmetros -
http://www.saltycrane.com/blog/2009/09/python-optparse-example/