MAC 448 – Programação em Redes de Computadores
BCC – 1º Semestre de 2009
Prova 1 – 24Abr09
Nome:_____Assinatura:_____NUSP:_____

1. Um arquivo de F bits é enviado do computador A para o computador B. Entre eles há um roteador e, portanto 2 links de R bits/segundo cada um. O computador A fragmenta o arquivo em segmentos de S bits e adiciona C bits de cabeçalho em cada segmento, formanto pacotes de (C+S) bits. Cada pacote é recebido totalmente pelo roteador antes de ser transmitido. Ignore atrasos de fila, de processamento e de propagação.

a. Quanto tempo leva para o primeiro pacote chegar em B?

2*(C+S)/R segundos.

b. Quanto tempo leva para transmitir o arquivo inteiro?

Após o primeiro pacote, a cada (C+S)/R segundos chega um novo pacote em B.
O total de pacotes restantes é de (F/S) – 1.

Portanto:
2*(C+S)/R + (F/S – 1)*(C+S)/R = (1+F/S)*(C+S)/R

c. E se houvesse Q links entre A e B?

Q*(C+S)/R + (F/S – 1)*(C+S)/R = (Q – 1+F/S)*(C+S)/R

2. Uma página HTML contendo 3 pequenos objetos é requisitada por um browser. A página e os objetos etão no mesmo servidor WEB. Supondo que o RTT (round-trip-time) para este servidor seja R segundos e desprezando-se os tempos de transmissão quanto tempo demora a carga da página completa quando se usa:

a) HTTP com conexão não persistente sem conexões paralelas?

2*R para a página HTML

2*R para cada um dos 3 objetos – uma conexão para cada objeto – só inicia após a chegada do anterior

Total = 8*R

b) HTTP com conexão não persistente com conexões paralelas?

2*R para a página HTML

2*R para os 3 objetos – uma conexão para cada objeto – solicitadas em paralelo

Total = 4*R

c) HTTP com conexão persistente com paralelismo?

2*R para a página HTML

R para os 3 objetos – são solicitados em paralelo na mesma conexão

Total = 3*R

3. Explique como um servidor de cache WEB numa rede local reduz o atraso num objeto solicitado por um cliente desta rede.

O servidor de cachê armazena todas as páginas consultadas recentemente.
Quando uma nova página é requisitada à internet, essa requisição vai ao servidor cache. O servidor faz a consulta à Internet com o parâmetro `If-modified-since: <data-hora>`. A `<data-hora>` é exatamente a da versão presente no cache.
A Internet devolve a nova página se a mesma foi modificada posteriormente, senão devolve o status `304 Not Modified`.
Assim, quando não há modificação a página não trafega novamente no links de acesso reduzindo o tempo de resposta.

4. Quais as características que tornam o UDP mais eficiente que o TCP?

a) Não há o estabelecimento e da conexão (mais eficiente)

b) O cabeçalho de cada segmento é menor. O TCP tem 20 bytes enquanto o UDP tem 12. Portanto menos bytes são transmitidos.
c) Não faz a recuperação de erros, ou seja, não retransmite pacotes que não tiveram confirmação de chegada.
d) Não faz controle de fluxo. O receptor não pode pedir ao emissor para enviar menos dados.
e) Não faz controle de congestionamento. O emissor não verifica o comportamento da rede para enviar menos dados ao receptor.

5. O TCP precisa estimar um valor de time-out, isto é, o tempo de espera para concluir que a confirmação de entrega de determinado pacote não vem mais.

a) Porque é importante que essa estimativa seja bem feita, ou seja, nem muito grande nem muito pequena?

Se for muito pequeno, o TCP pode retransmitir pacotes desnecessariamente, pois a confirmação pode chegar após pacotes serem retransmitidos.
Se for muito grande, perde-se tempo, pois a retransmissão de pacotes perdidos demora a ocorrer.

b) Como o TCP faz esta estimativa?

O TCP mantém um valor estimado dinamicamente, baseado nas amostras do tempo de confirmação de pacotes.
O RTT (Round Trip Time) é recalculado a cada resposta e a estimativa é feita privilegiando as amostras recentes:

```
novo RTT = 0,8*(RTT atual) + 0,2*(última amostra de RTT).
```

O time-out é estimado a partir do RTT:

```
timeout = RTT + 4*desvio
novo desvio = 0,8*(desvio atual) + 0,2* |última amostra de RTT-novo RTT|
```

6. Explique como é feito o controle de congestionamento no TCP?

O controle de congestionamento é uma iniciativa do lado transmissor.
Esse lado monitora a ocorrência de 2 eventos que podem significar que esteja havendo perda de dados na rede devido ao excesso de dados transmitidos. Os 2 eventos monitorados são:
Recepção de 3 ACKs repetidos
Ocorrência de Time-Out

A cada conexão, o TCP tem 2 fases:
Slow-start (partida lenta) que vai dobrando o tamanho da janela a partir de 1 até certo limite pré-estabelecido.
Congestion-avoidance (evitar colisões) que vai incrementando a janela em 1.
Quando ocorre triplo ACK o incremento da janela passa a ser de 1 e o limite é reavaliado (médio).
Quando ocorre time-out (grave) recomeça o processo na fase slow-start e o limite é reavaliado também.

7. Um cliente envia um arquivo a cada segundo a um servidor (**`servidor.com`**). O arquivo é enviado linha a linha com o número da linha na frente. A cada linha enviada recebe “OK” ou “NOK”. Se receber “NOK” envia a linha novamente. A última linha do arquivo é vazia.

Algoritmo do Cliente:

```
while (true) {
  sleep(1000);
  NLIN=1;
  do {x = LeLinha(); //le linha NLIN do arquivo
      Envie NLIN+x ao servidor; Espere OK do servidor;
      Se vier NOK repita o comando acima;
      NLIN++;
  }
  while (x!=””); // fim de arquivo
}
```

Usando sockets TCP em Java, escreva uma aplicação para o lado cliente.
A porta do servidor é 8888.
Supor que exista a função **`LeLinha()`** que lê a próxima linha do arquivo.

```
import java.io.*;
import java.net.*;
class TCPClient {
   public static void main(String argv[]) {
     String x; int NLIN;
```

```
      // ativa a cada 1000 ms e envia arquivo
      while (true) {
        Thread.sleep(1000);
        // cria socket e pede conexão ao host hostname na porta 8888
        Socket cliente = new Socket("servidor.com", 8888);
        // cria dutos de entrada e saida com o servidor
        DataOutputStream saida = new DataOutputStream(cliente.getOutputStream());
        BufferedReader entrada = new BufferedReader(new InputStreamReader(
          cliente.getInputStream()));
        // envia arquivo ao servidor
        NLIN=1;
        do {x = LeLinha(NLIN);
            saída.writeBytes(NLIN+ ' ' + x);
            while (true) {
              y = entrada.readLine();
              if (y.equals ("OK")) break;
              saída.writeBytes(NLIN+ ' ' + x); // envia novamente
            }
        }
        while (!x.equals(""))
        // fecha o socket e os dutos do cliente com o servidor
        cliente.close();
        saida.close();
        entrada.close();
      }
    }
}
```

8. O Servidor correspondente deve receber a conexão de um cliente e criar uma thread que vai receber o arquivo deste cliente, pois podem ser atendidos vários clientes ao mesmo tempo.

Algoritmo do servidor `main()`:

```
while (true) {
   Espere cliente se conectar e lança thread para este cliente;
}
```

Thread:

```
N=1;
do {Receba linha z;
    Separe NLIN;
    if(NLIN==N) {
       GravaLinha(z sem NLIN);
       Responda OK;
       N++;
    }
    else Responda NOK;
}
while (z != “”);
```

Usando sockets TCP em Java, escreva uma aplicação para o lado servidor.
Supor que exista a função **`GravaLinha(z)`** que grava uma linha no arquivo.

```
// Servidor.java
import java.io.*;
import java.net.*;
import java.util.*;
public class Servidor extends Thread {
  public static void main(String args[]) {

    // criando um socket que fica escutando a porta 8888.
    ServerSocket s = new ServerSocket(8888);
    // Loop principal
    while (true) {
      // aguarda algum cliente se conectar.
      System.out.print(“Esperando alguem se conectar...”);
      Socket conexao = s.accept();
      System.out.println(“ Conectou!”);
      // cria uma nova thread para tratar essa conexão
      Thread t = new Servidor(conexao);
      t.start();
      // voltando ao loop, esperando mais alguém se conectar.
    }
  }

  // socket de cada instancia deste cliente
  private Socket c;
  // variáveis de cada instancia deste cliente
  private String X;
  private String r = “ok”;
  private N=1;

  // construtor que recebe o socket deste cliente
  public Servidor(Socket s) {
    c = s;
  }

  // execução da thread
  public void run() {
    // abre dutos de comunicação com o cliente
    BufferedReader entrada = new BufferedReader(new
       InputStreamReader(c.getInputStream()));
    DataOutputStream saida = new DataOutputStream(c.getOutputStream());
    // recebe o arquivo do cliente
    N=1;
    do {
         x = entrada.readLine();
         // separa os elementos da linha
```

```
            StringTokenizer TL = new StringTokenizer(x);
            // transforma primeiro elemento para int
            int NLIN = Integer.parseInt(TL.nextToken());
            // consistência
            if(NLIN==N) {
               // retira NLIN grava e responde “OK” ao cliente
               int k = x.indexOf(” “);
               x = x.substring(k);
               GravaLinha(x);
               saida.println(“OK”);
               N++;
            }
            else saida.println(“NOK”);
      } while (x != “”);
      c.close();
   }
}
```

Sockets TCP - resumo

1) Abrir um socket cliente

```
Socket C = new Socket("nome da maquina", porta);
```

2) Abrir um socket servidor

```
ServerSocket Servidor = new ServerSocket(porta);
```

3) Para o servidor esperar por uma conexão de algum cliente

```
Socket S = Servidor.accept();
```

4) Para o cliente criar o duto de entrada para receber dados do servidor

```
DataInputStream entrada = new
        DataInputStream(C.getInputStream());
```

5) Para o servidor criar o duto de entrada para receber dados do cliente

```
DataInputStream entrada = new
     DataInputStream(S.getInputStream());
```

6) Para o ler os dados (cliente ou servidor)

```
entrada.readLine()
```

7) Para o cliente criar o duto de saida para enviar dados ao servidor

Pode ser usada a classe `PrintStream` or `DataOutputStream`

```
PrintStream saida = new PrintStream(C.getOutputStream());
```

ou então

```
DataOutputStream output = new DataOutputStream(C.getOutputStream());
```

8) Para o servidor criar o duto de saida para enviar dados ao cliente

Também podem ser usadas `PrintStream` ou `DataOutputStream.`

```
PrintStream saida = new PrintStream(S.getOutputStream());
```

ou então

```
DataOutputStream saida = new DataOutputStream(S.getOutputStream());
```

9) Para enviar os dados (cliente ou servidor)

A classe `PrintStream` tem algumas funções para enviar os dados:

```
saida.Write()
saida.println()
```

Idem para a classe `DataOutputStream.`

```
saida.writebytes()
```

10) Para fechar o lado cliente

```
saida.close();
entrada.close();
C.close();
```

11) Para fechar o lado servidor

```
 saida.close();
 entrada.close();
 S.close();
 Servidor.close();
```