PÓS-GRADUAÇÃO ALFA

FACULDADES ALFA ALVES ARIA

A MELHOR ESCOLA DE NEGÓCIOS DO CENTRO-OESTE

Diego Am érico Guedes

PROGRAMAÇ ÃO COM FRAMEWORKS E COMPONENTES

GRUPO JOSÉ ALVES

Protocolo UDP

Características do protocolo UDP

- Datagramas são emitidos entre processos sem a existência de confirmações ou novas tentativas de transmissão
- Se ocorrer uma falha, a mensagem pode não chegar
- Datagrama carrega o endereço de destino do processo remetente
- O processo destino deve especificar um vetor de bytes para receber as mensagens (datagrama IP pode ter até 64KB)

Características do protocolo UDP

- Bloqueio: send não bloqueante e receive bloqueante
  - A operação send retornará assim que a mensagem for copiada para o buffer da máquina
  - Ao chegar, a mensagem é copiada para a fila de recepção do socket associado ao processo destino
  - A mensagem é recuperada dessa fila quando a operação receive é executada
  - Receive bloqueia a execução do processo até que um datagrama seja recebido

Características do protocolo UDP

- Timeout
  - Em algumas situações não é desejado que um processo espere indefinidamente por uma mensagem
  - Timeout (limites temporais) podem ser configurados nos sockets.
  - O timeout deve ser bem grande, em comparação com o tempo exigido para transmitir uma mensagem.

Características do protocolo UDP

- Modelo de Falhas
  - Falhas por omissão
    - Mensagens podem ser descartadas (*buffer* de origem, destino ou canal) por erros de *checksum* e espaço disponível no *buffer*
  - Ordenamento
    - Mensagens podem ser entregues fora de ordem
- **É possível criar um serviço confiável, desde que as falhas sejam tratadas na camada de aplicação**

Características do protocolo UDP

- Exemplos de Serviços que utilizam UDP
  - DNS
  - VOIP

- Classes Principais
  - InetAddress: representa um endereço IP.
    - GetByName (String): Obtem uma instância InetAddress a partir de um nome de domínio ou endereço IP.
  - DatagramPacket: representa um datagrama UDP
    - DatagramPacket(mensagem, tamanho,IP, porta)
    - DatagramPacket(mensagem, tamanho)
    - getData: recupera a mensagem
    - getPort: recupera a porta
    - getAddress: recupera o IP

- Classes Principais
  - DatagramSocket: representa o socket UDP
    - DatagramSocket(): cria um socket no cliente
    - DatagramSocket(porta): cria um socket no Servidor
    - send(datagrama): envia um DatagramPacket contendo mensagem e endereço de destino
    - receive(datagrama): recebe um DatagramPacket
    - setSoTimeout(tempo): configura timeout para o socket

UDP client sends a message to the server and gets a reply

```
import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(String args[]){
            // args give message contents and server hostname
            DatagramSocket aSocket = null;
              try {
                        aSocket = new DatagramSocket();
                        byte [] m = args[0].getBytes();
                        InetAddress aHost = InetAddress.getByName(args[1]);
                        int serverPort = 6789;
                        DatagramPacket request = new DatagramPacket(m,   args[0].length(), aHost, serverPort);
                        aSocket.send(request);
                        byte[] buffer = new byte[1000];
                        DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                        aSocket.receive(reply);
                        System.out.println("Reply: " + new String(reply.getData()));
              }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
              }catch (IOException e){System.out.println("IO: " + e.getMessage());}
            finally {if(aSocket != null) aSocket.close();}
    }
}
```

UDP server repeatedly receives a request and sends it back to the client

```
import java.net.*;
import java.io.*;
public class UDPServer{
        public static void main(String args[]){
        DatagramSocket aSocket = null;
            try{
                    aSocket = new DatagramSocket(6789);
                    byte[] buffer = new byte[1000];
                    while(true){
                        DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                       aSocket.receive(request);
                       DatagramPacket reply = new DatagramPacket(request.getData(),
                                request.getLength(), request.getAddress(), request.getPort());
                       aSocket.send(reply);
                    }
             }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
            }catch (IOException e) {System.out.println("IO: " + e.getMessage());}
        finally {if(aSocket != null) aSocket.close();}
    }
}
```

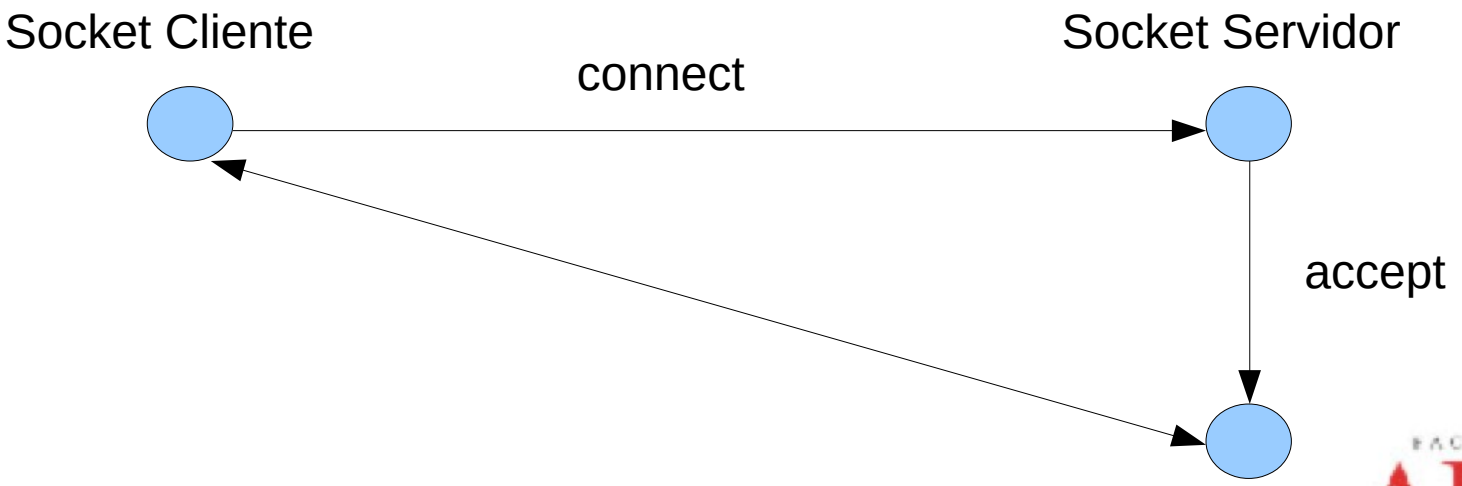
Protocolo TCP

Características do protocolo TCP

- Tamanho das mensagens é ilimitado
- Usa esquema de confirmação de mensagem que permite retransmissão em caso de perda
- Usa esquema de controle de fluxo que bloqueia remetentes rápidos
- Usa identificadores de mensagens, permitindo detectar e rejeitar pacotes duplicados e ordenar pacotes que chegam fora de ordem

Características do protocolo TCP

- Dois processos estabelecem uma conexão antes da troca de mensagem
- Com a conexão, os processos lêem e escrevem fluxos, sem necessidade de usar IP e porta
- Conexão



Características do protocolo TCP

- Bloqueio
  - Um processo pode ficar bloqueado ao executar uma operação receive e a fila estiver vazia
  - Um processo pode ficar bloqueado ao executar uma operação send e a fila do receptor estiver cheia
- Threads
  - Quando um servidor aceita uma conexão, geralmente ele cria uma nova *thread* para se comunicar com o cliente

Características do protocolo TCP

- Modelo de Falhas
  - Fahas por omissão:
    - Se a perda de pacotes ultrapassar um limite
    - Se a rede for rompida
    - Se a rede estiver congestionada
  - O protocolo TCP não fornece comunicação confiável, pois não garante a entrega diante de todas as dificuldades possíveis.
- Uso do TCP
  - HTTP, FTP, Telnet, SMTP

API de Java para Programação com socket

- Classes
  - ServerSocket: representa o socket do servidor, no qual ele espera requisições connect
    - Accept(): recupera um pedido da fila do socket ou bloqueia se fila estiver vazia. Retorna uma instância de Socket
  - Socket: classe usada pelos dois processos para ler ou escrever fluxos
    - Socket(host,porta): cria socket no cliente e o connecta ao servidor

TCP client makes connection to server, sends request and receives reply

```
import java.net.*;
import java.io.*;
public class TCPClient {
        public static void main (String args[]) {
        // arguments supply message and hostname of destination
        Socket s = null;
          try{
                int serverPort = 7896;
                s = new Socket(args[1], serverPort);
                DataInputStream in = new DataInputStream( s.getInputStream());
                DataOutputStream out =
                        new DataOutputStream( s.getOutputStream());
                out.writeUTF(args[0]);          // UTF is a string encoding see Sn 4.3
                String data = in.readUTF();
                System.out.println("Received: "+ data) ;
          }catch (UnknownHostException e){
                        System.out.println("Sock:"+e.getMessage());
          }catch (EOFException e){System.out.println("EOF:"+e.getMessage());
          }catch (IOException e){System.out.println("IO:"+e.getMessage());}
        finally {if(s!=null) try {s.close();}catch (IOException e){System.out.println("close:"+e.getMessage());}}
        }
}
```

TCP server makes a connection for each client and then echoes the client's request

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
            try{
                    int serverPort = 7896;
                    ServerSocket listenSocket = new ServerSocket(serverPort);
                    while(true) {
                            Socket clientSocket = listenSocket.accept();
                            Connection c = new Connection(clientSocket);
                    }
            } catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
    }
}

// this figure continues on the next slide
```

continued

```
class Connection extends Thread {
        DataInputStream in;
        DataOutputStream out;
        Socket clientSocket;
        public Connection (Socket aClientSocket) {
            try {
                    clientSocket = aClientSocket;
                    in = new DataInputStream( clientSocket.getInputStream());
                    out =new DataOutputStream( clientSocket.getOutputStream());
                    this.start();
             } catch(IOException e)  {System.out.println("Connection:"+e.getMessage());}
        }
        public void run(){
            try {                                       // an echo server
                    String data = in.readUTF();
                    out.writeUTF(data);
            } catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
            } catch(IOException e) {System.out.println("IO:"+e.getMessage());}
           finally{ try {clientSocket.close();}catch (IOException e){/*close failed*/}}
        }
}
```