Relembrando . . .

- Processos duma aplicação distribuída comunicam via mensagens.
  - O formato e o significado destas mensagens depende do protocolo usado.
- As mensagens são transportadas entre processos através de redes de computadores.
- Para uma aplicação distribuída a rede pode ser abstraída como um canal de comunicação, com determinadas propriedades.
- Na Internet, os protocolos de transporte normalmente usados são TCP e UDP.

Resumo das Propriedades de UDP e de TCP

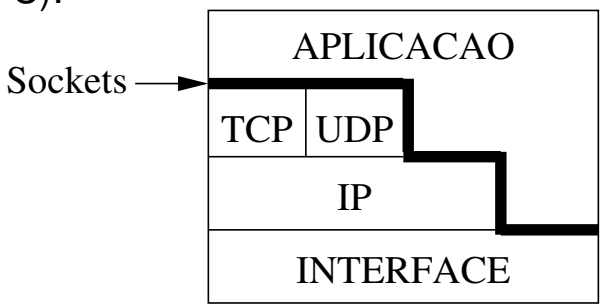
Propriedade	UDP	TCP
Abstracção	Mens.	*Stream*
Baseado em Conexão	N	S
Fiabilidade (perda & duplicação)	N	S
Ordem	N	S
Controlo de Fluxo	N	S
Número de Receptores	n	1

Sockets: Sumário

- Conceito (e história).
- *Sockets* UDP
  - API de Java
  - API da biblioteca C.
  - (Comunicação *multicast*.)

Sockets

- Origem no (UNIX) BSD 4.1c (1982), mas adoptada (com variações) por quase todos sistemas operativos (incluindo Windows).
- Fornece uma interface para qualquer das camadas da Internet (em C):



- Programação com *sockets* é muito semelhante à programação de protocolos:
  - *sockets* expõem a rede ao programador – ausência total de transparência.

Conceito de *socket*

socket: é um objecto (do sistema operativo) que representa o ponto de acesso a um canal de comunicação entre processos.

- Os passos para comunicação com *sockets* dependem do tipo de canal usado:
  - Sem conexão (UDP)
  - Com conexão (TCP)
- Para que um *processo remoto* possa estabelecer um canal com um *socket*, é necessário atribuir um *nome* ao *socket*:

  uma ficha telefónica não é suficiente para estabelecer uma chamada telefónica.

Passos para Comunicação com *Sockets* UDP

1. Criar um *socket*:
2. Atribuir-lhe um *nome* (*opcional*)
   - Se um *socket* só receber mensagens de *sockets* a que enviou mensagens, não é necessário atribuir-lhe explicitamente um nome.
3. Transferir informação
   - Enviar e receber mensagens.

Sockets: Sumário

- Conceito.
- *Sockets* UDP
  - **API de Java**
  - API da biblioteca C.
  - (Comunicação *multicast*.)

Comunicação *Sem* Conexão em Java

- Java define 2 classes especificamente para comunicação sem conexão:

  `DatagramSocket` representa um *socket* UDP;

  `DatagramPacket` representa um *datagrama* UDP.

- Tipicamente, uma aplicação:
  - Cria um `DatagramSocket`.
  - Envia e recebe `DatagramPackets` através do `DatagramSocket`
- Para usar qualquer destas classes é conveniente conhecer a classe:

  `InetAddress` representa um endereço IP;

  e as suas subclasses `Inet4Address` e `Inet6Address` que representam os endereços IPv4 e IPv6, respectivamente.

Classe `InetAddress`

- Suporta um conjunto de métodos estáticos que permitem obter objectos `InetAddress` associados a um nome DNS ou a um endereço IP:

  `static InetAddress getByAddress(byte[] addr)`: o endereço deve estar na *ordem da rede* (MSB em `addr[0]`)

  `static InetAddress getByName(String host)`

  `static InetAddress getLocalHost()`

- Inclui ainda um conjunto de métodos utilitários, p.ex.:

  `String getHostName(InetAddress addr)`

  `String toString()`

- As subclasses `Inet4Address` herdam estes métodos e incluem ainda alguns métodos específicos.

Classe `DatagramPacket`

- Suporta um conjunto de construtores:

  `DatagramPacket( byte[] buf, int length)`: constrói um datagrama para receber/enviar datagramas com até `length` *bytes*;

  `DatagramPacket( byte[] buf, int length, InetAddress address, int port)`: constrói um datagrama e inicializa o endereço e o porto (destino).

- Oferece um conjunto de operações como p.ex.:

  `InetAddress getAddress()` extrai o endereço do *socket* remoto;

  `int getPort()` extrai o porto do *socket* remoto;

  `byte[] getData()` extrai os dados do datagrama;

  `void setData(byte[] buf)` inicializa os dados do datagrama.

Classe `DatagramSocket` (1/2)

- Suporta um conjunto de construtores, alguns dos quais permitem:
  - Atribuir um nome ao *socket* UDP criado.

    IMP.- Só é necessário atribuir o nome a um *socket*, se se pretender que receba mensagens de *sockets* remotos a que não enviou mensagens previamente.
- Suporta operações para:
  - Enviar (quer *unicast* quer *broadcast*) datagramas.
  - Receber datagramas.
  - Configurar diferentes parâmetros dos *sockets*, como p.ex.:
    * tamanho de *buffers*;
    * valores de temporização.

Classe `DatagramSocket` (2/2)

- Construtores:

  `DatagramSocket()` Cria um *socket* UDP sendo o seu nome determinado pelo sistema.

  `DatagramSocket(int port)` Cria um *socket* UDP associado ao porto especificado.

  `DatagramSocket(int port, InetAddress addr)` Cria um *socket* UDP com o nome especificado.

- Métodos:

  `void receive(DatagramPacket p)` recebe um datagrama UDP que é copiado para o seu argumento.

  `void send(DatagramPacket p)` envia o datagrama UDP que lhe é passado como argumento;

  `int getLocalPort()` retorna o porto local associado ao *socket*;

Comunicação *Sem* Conexão em Java: Exemplo (1/2)

```
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException{
        if (args.length != 2) {
             System.out.println("Usage: java Echo <hostname> <string t
             return;
        }
        // send request
        DatagramSocket socket = new DatagramSocket();
        byte[] sbuf = args[1].getBytes();
        InetAddress address = InetAddress.getByName(args[0]);
        DatagramPacket packet = new DatagramPacket(sbuf, sbuf.length,
                                                   address, 4445);
        socket.send(packet);
```


Comunicação *Sem* Conexão em Java: Exemplo (2/2)

```
        // get response
        byte[] rbuf = new byte[sbuf.length];
        packet = new DatagramPacket(rbuf, rbuf.length);
        socket.receive(packet);

        // display response
        String received = new String(packet.getData());
        System.out.println("Echoed Message: " + received);

        socket.close();
    }
}
```