Network Programming - UDP

SCOMRED, October 2017

Aplicações clientes e servidoras

No contexto do modelo cliente-servidor, um cliente de rede é uma aplicação que pretende enviar dados para serem processados por uma aplicação servidora de rede, depois de enviar o pedido fica a aguardar uma resposta.

A interação com o utilizador ocorre na aplicação cliente. Para enviar o pedido a aplicação cliente necessita de conhecer o endereço **IP do servidor** e o **número de porto do servidor**:

O **endereço IP do servidor** é normalmente fornecido pelo utilizador, eventualmente sob a forma de um nome DNS.

O **número de porto do servidor** é fixo e é do conhecimento da aplicação cliente (*hardcoded*), para cada tipo de aplicação servidora standard está estabelecido um número de porto fixo.

Por exemplo, quando se pretende utilizar o protocolo HTTP para aceder a uma página WEB, a aplicação cliente (browser) sabe que o número de porto do servidor é o 80, o endereço do servidor é fornecido pelo utilizador através do URL.

O número de porto do servidor é algo que deve estar definido no protocolo de aplicação.

Protocolo de aplicação

O protocolo de aplicação é uma especificação de todas as trocas de dados entre as aplicações de rede que utilizam esse protocolo. Inclui desde logo a indicação de qual o protocolo de transporte usado (normalmente UDP ou TCP), os números de porto fixos usados (normalmente do lado da aplicação servidora), a sequência das trocas de dados e os respetivos conteúdos.

Quando no contexto do desenvolvimento de aplicações pretendemos implementar comunicações através da rede podemos lançar mão de um protocolo de aplicação standard já existente, por exemplo o HTTP. Nesse caso temos de estudar a respetiva especificação e implementar as aplicações de acordo com ela.

A alternativa é criar um novo protocolo de aplicação, o desenvolvimento da especificação do novo protocolo de aplicação tem de ser realizado antes de se iniciar a sua implementação. A especificação do protocolo de aplicação deve ser independente do sistema operativo e linguagem de programação usada. Deste modo o protocolo poderá aplicado em qualquer tipo de sistema.

UDP e TCP

As aplicações de rede correntes utilizam dois protocolos de transporte: UDP (*User Datagram Protocol*) e/ou TCP (*Transmission Control Protocol*).

Estes protocolos operam sobre IPv4 e IPv6, isto significa que os seus dados são encapsulados em pacotes IPv4 ou IPv6. Num nó ***dual-stack*** será usado IPv4 ou IPv6 em função do tipo de endereço de destino que é especificado. Obviamente que a comunicação através de IPv4 só é possível entre dois nós que possuem IPv4 e a comunicação através de IPv6 só é possível entre dois nós que possuem IPv6.

Uma caraterística comum ao UDP e TCP é que ambos definem números de porto de 16 bits para identificar aplicações no interior de cada nó. No entanto os números de porto UDP são totalmente independentes dos números de porto TCP.

São os números de porto que tornam os protocolos UDP e TCP adequados a aplicações dos utilizador, garantem que comunicações de diferentes aplicações (possivelmente pertencentes a utilizadores diferentes) não interferem entre si.

UDP (*User Datagram Protocol*)

O UDP é classificado como não fiável e *connection-less*, os dados produzidos pela aplicação são colocados num datagrama UDP, para se proceder ao seu envio basta especificar o endereço IP de destino (IPv4 ou IPv6) e o número de porto UDP de destino.

O UDP é não fiável, não existe garantia que o datagrama UDP chegue ao destino, não existe feedback, o emissor não é informado se o datagrama UDP chegou ao destino ou não.

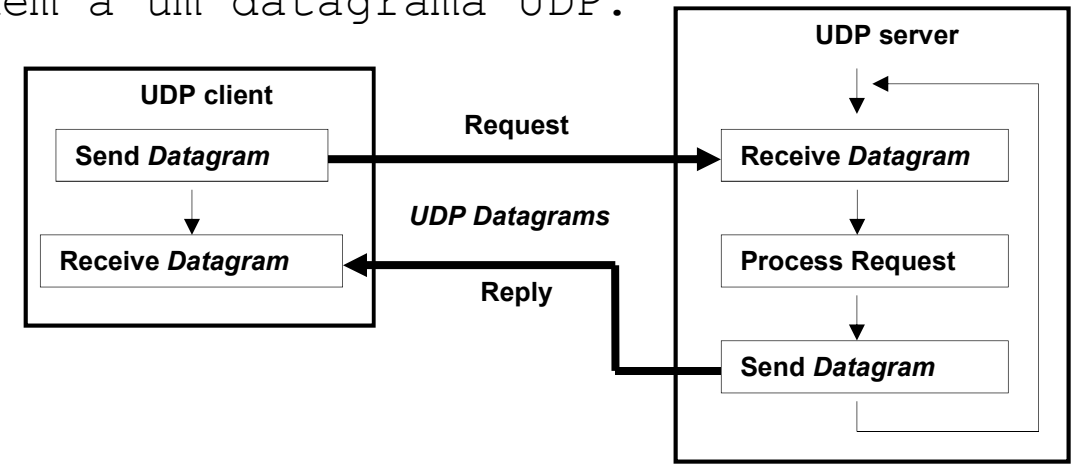
O volume de dados colocados pela aplicação em cada datagrama UDP **não deve exceder os 512 bytes**, caso contrário isso poderá ser um motivo para o datagrama não chegar ao destino.

Não existe ligação entre as aplicações (*connection-less)*, cada datagrama é enviado isoladamente especificando um destino (endereço IP e número de porto UDP). O facto de não haver ligação permite o envio para endereços multicast, e broadcast no caso do IPv4.

Clientes e servidores UDP

Dado que o UDP é *connection-less*, normalmente tanto o pedido como a resposta correspondem a um datagrama UDP.

O cliente sabe qual é o número de porto do servidor (protocolo de aplicação), recebe do utilizador o endereço do servidor e o conteúdo do pedido.



A aplicação cliente envia ao servidor o pedido e de seguida fica a aguardar por uma resposta, quando a resposta chega apresenta-a ao utilizador.

Sendo o UDP não fiável pode nunca chegar nenhuma resposta, o cliente deve prever essa possibilidade.

Se o volume do pedido ou resposta exceder os 512 bytes, poderá ser dividido em vários datagramas, mas o protocolo de aplicação torna-se mais complexo. Um alternativa é usar TCP em lugar de UDP.

TCP (*Transmission Control Protocol*)

O TCP é classificado como fiável e *connection-oriented*, antes de se puderem transferir dados é necessário criar uma ligação lógica entre duas aplicações chamada conexão TCP.

A conexão TCP é criada usando o modelo cliente-servidor, a aplicação cliente envia um pedido de ligação TCP ao servidor, para isso necessita de conhecer o endereço IP do servidor e o número de porto TCP do servidor.

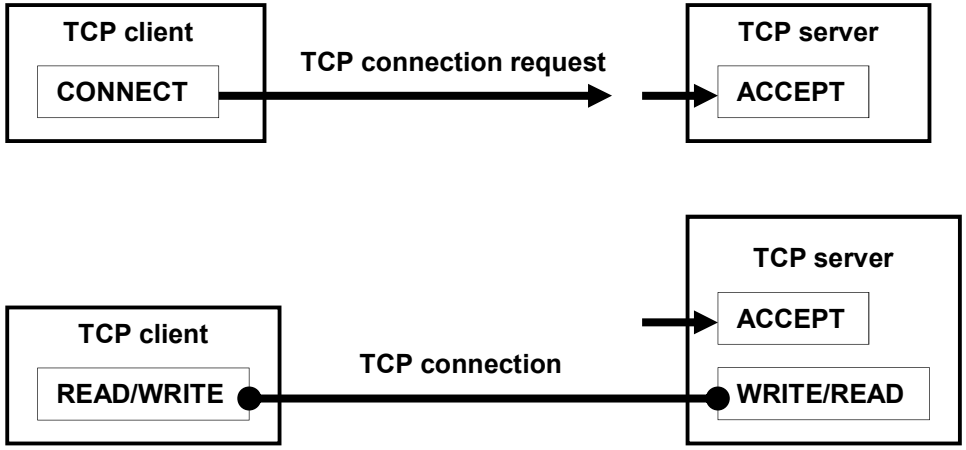
Criada a conexão TCP entre as duas aplicações, o envio e receção de dados faz-se como se as aplicações estivessem a escrever e a ler de ficheiros locais. O TCP garante a fiabilidade e a ordem dos bytes escritos, qualquer perda de dados é automaticamente corrigida pelo TCP através da retransmissão.

Enquanto no UDP as operações de envio e receção são orientadas por datagramas, no UDP são orientadas por bytes. Se um protocolo de aplicação UDP tem apenas de garantir o sincronismo de datagramas (emissão/receção), no TCP é necessário garantir o sincronismos de bytes (write/read).

Clientes e servidores TCP

Antes de um cliente TCP poder enviar pedidos tem de estabelecer a conexão TCP com o servidor. Do lado da aplicação cliente isso é relativamente simples, basta especificar o endereço IP e número de porto do servidor.

Para que a conexão seja estabelecida com sucesso, o servidor tem de a aceitar (accept), temos então uma conexão TCP que tem duas extremidades, uma do lado do cliente e outra do lado do servidor:



O problema é que o servidor deve continuar a aceitar clientes (accept) e para cada cliente terá mais um conexão TCP através da qual o respetivo cliente pode enviar um pedido a qualquer momento.

Claramente o servidor tem várias tarefas a realizar ao mesmo tempo, assim em Java a solução típica é lançar um novo thread para cada novo cliente (accept).

Sockets

Sob o ponto de vista da API (*Application Programming Interface*) a utilização da rede é conseguida através do conceito de ***socket***. Um ***socket*** representa um acesso da aplicação à pilha de protocolos de rede do sistema operativo. No caso do Java um *socket* é um objeto (obtido por instanciação de classes Socket).

Dependendo do tipo de classe o socket poderá estar associado ao UDP (Classe **DatagramSocket**) ou ao TCP (Classes **Socket** e **ServerSocket**). Um *socket* está **implicitamente associado aos endereços IP locais do nó** onde a aplicação está a ser executada.

Embora não seja obrigatório, normalmente quando o *socket* é criado (instanciação) é desde logo **associado a um número de porto** fornecido ao construtor. No caso de uma aplicação servidora, o número de porto é fixo (faz parte do protocolo de aplicação), se já estiver em uso a instanciação vai gerar uma exceção. Uma aplicação cliente não necessita de um número de porto fixo, não o fornecendo ao construtor será associado ao novo *socket* um qualquer número de porto disponível no nó.

DatagramSocket – exemplos de instanciação

Se pretendemos implementar uma **aplicação servidora UDP**, o *socket* deverá ficar associado a um número de porto UDP fixo para que os pedidos do cliente enviados para esse porto sejam recebidos:

```
DatagramSocket mySocket = new DatagramSocket(9999);
```

O cliente UDP correspondente deverá enviar os pedidos para o número de porto UDP 9999 do servidor, mas o *socket* usado pela **aplicação cliente UDP** pode ficar associado a um qualquer número de porto que esteja livre no nó onde a aplicação é executada:

```
DatagramSocket mySocket = new DatagramSocket();
```

Note-se que tanto no UDP como no TCP os números de porto inferiores a 1024 são reservados e apenas o administrador do sistema operativo os pode usar.

Classe *InetAddress*

A classe **InetAddress** permite manusear endereços IP (IPv4 e IPv6).

O método estático **getByName()** recebe como argumento um *string* contendo um endereço IPv4, ou endereço IPv6, ou um nome DNS e devolve uma instancia da classe contendo o endereço fornecido.

Tendo uma instancia de InetAddress com o endereço definido, ela pode ser usada para definir o endereço IP de destino de um datagrama UDP ou de uma conexão TCP.

O método **getHostAddress()** devolve um *string* contendo a representação legível do endereço IP contido numa instancia da classe InetAddress.

O método getByName será tipicamente utilizado em aplicações cliente para converter o endereço do servidor fornecido pelo utilizador num instancia de InetAddress.

O método getHostAddress() pode ser usado para apresentar o endereço de origem de um datagrama ou uma conexão TCP após a sua receção ou aceitação.

Classe *DatagramPacket*

A classe **DatagramPacket** destina-se a criar datagramas UDP antes de serem emitidos e armazenar datagramas UDP quando são recebidos. Um objeto DatagramPacket contém vários campos que podem ser manuseados através de métodos apropriados:

Data - Dados a serem enviados ou que foram recebidos:

```
void setData(byte[] buf, int offset, int length);
byte[] getData();
```

Length – número de bytes a serem enviados ou que foram recebidos:

```
void setLength(int length);
int getLength();
```

Address – **endereço IP remoto** (para onde se pretende enviar ou de onde foi recebido):

```
void setAddress(InetAddress addr);
InetAddress getAddress();
```

Classe *DatagramPacket*

Port – número de porto UDP remoto (para onde se pretende enviar ou de onde foi recebido):

```
void setPort(int port);

int getPort();
```

A classe **DatagramPacket** possui vários construtores, de entre eles os mais utilizados são:

```
DatagramPacket(byte[] buf, int length);

DatagramPacket(byte[] buf, int length, InetAddress address, int port);
```

O primeiro é normalmente usado para criar um DatagramPacket destinado a receber um datagrama UDP, nesse caso os únicos campos relevantes são os relativos ao local onde será guardado o conteúdo do datagrama, o endereço IP de origem e número de porto de origem estarão disponíveis após a receção.

Envio de datagramas UDP

Para enviar um datagrama UDP é necessário:

1º Criar um socket UDP.

2º Criar um datagrama UDP, definindo os dados e o destino.

3º Invocar o método send() do socket UDP fornecendo-lhe o datagrama UDP como argumento.

Exemplo:

```
String frase="Teste";
DatagramSocket mySocket = new DatagramSocket();
DatagramPacket udpPacket = new DatagramPacket(frase.getBytes(), frase.length,
        InetAddress.getByName("192.168.10.1"), 9999);
mySocket.send(udpPacket);
mySocket.close();
```

Este exemplo envia um datagrama contendo o string "Teste" para o número de porto UDP 9999 do nó de rede que possui o endereço IPv4 192.168.10.1, se a aplicação pretender terminar, antes de o fazer deve fechar o socket (close).

Receção de datagramas UDP

Para receber um datagrama UDP é necessário:

1º Criar um socket UDP associado a um número de porto fixo.

2º Criar um datagrama UDP, definindo o local de armazenamento dos dados a receber e tamanho máximo.

3º Invocar o método receive() do socket UDP fornecendo-lhe o datagrama UDP como argumento. Note-se que o método receive() **bloqueia a aplicação até que seja recebido um datagrama.**

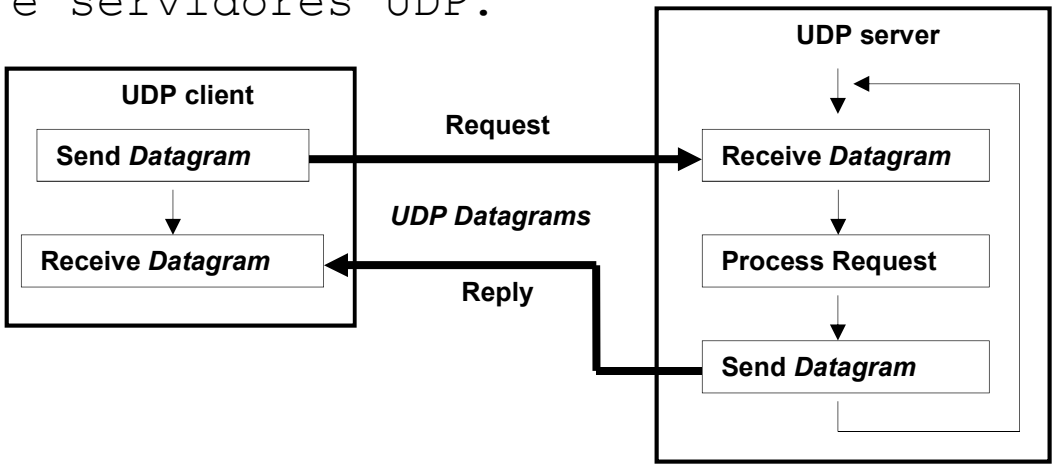
Exemplo:

```
byte[] data = new byte[30];
DatagramSocket mySocket = new DatagramSocket(9999);
DatagramPacket udpPacket = new DatagramPacket(data, 30);
mySocket.receive(udpPacket);
String frase = new String( udpPacket.getData(), 0, udpPacket.getLength());
System.out.println(frase + " received from IP: " +
         udpPacket.getAddress().getHostAddress() + " port: " +
         udpPacket.getPort());
mySocket.close();
```

Clientes UDP – deteção de falhas

Sabendo com se podem enviar e receber datagramas UDP podemos agora implementar clientes e servidores UDP.

Não podemos esquecer que o UDP não é fiável. O servidor UDP é uma aplicação que recebe um pedido, processa-o, envia a resposta e passa para o pedido do cliente seguinte.



Falhas no UDP não afetam o servidor. Para o cliente o cenário é diferente, porque após enviar o pedido fica bloqueado à espera da resposta do servidor. Se algo corre mal e a resposta nunca chega, a aplicação cliente fica eternamente bloqueada.

A solução é o cliente definir um tempo máximo para a resposta chegar, para isso pode associar um timeout ao seu socket através do método **setSoTimeout(int milliseconds)**. Se o **receive()** demora mais do que este valor, desbloqueia gerando a exceção **SocketTimeoutException**.

Clientes UDP – broadcast

Algo que é possível fazer com o UDP é enviar datagramas para endereços multicast e no caso do IPv4 para o endereço de broadcast. Por omissão, um DatagramSocket não permite a emissão em broadcast, isso pode ser alterado através do método **setBroadcast(boolean on).**

Um cliente pode usar a emissão em broadcast para chegar a um servidor mesmo sem conhecer o seu endereço IP, como a rede entrega uma cópia do datagrama em todos os nós da rede, se um deles for um servidor apropriado, vai receber o pedido.

É claro que isto só funciona dentro dos limites de uma rede IP, os routers não transferem estes pacotes entre redes IP.

Outro aspeto a ter em consideração é que se existirem vários servidores na rede o cliente não vai obter apenas uma resposta, obtém um número de respostas igual ao número de servidores.

O endereço de broadcast que as aplicações devem usar é **255.255.255.255,** este endereço de broadcast genérico na rede local funciona em qualquer rede IP.

Exercícios práticos – Cliente e servidor UDP

Protocolo de aplicação:

- Cliente-servidor UDP
- O cliente envia uma frase num datagrama UDP (pedido)
- O servidor recebe o datagrama, inverte a frase e devolve-a ao cliente também num datagrama UDP (resposta).
- O servidor recebe o pedido seguinte (volta ao estado anterior).

Requisitos da aplicação cliente:

- Aplicação de consola (seu GUI).
- A aplicação solicita ao utilizador o endereço do servidor e valida o endereço.
- A aplicação solicita ao utilizador uma frase a enviar, se for **sair** termina, caso contrário envia-a ao servidor e aguarda pela resposta.
- Quando a resposta chega imprime-a e volta ao ponto anterior.

Solução base fornecida

No Moodle são disponibilizados dois ficheiros de texto com código fonte Java: **UdpCli.java** e **UdpSrv.java**.

Correspondem respetivamente à implementação do cliente UDP e servidor UDP de acordo com os requisitos enunciados.

No **NetBeans** crie dois novos projetos do tipo ***Java/Java Application***, defina os nomes de cada projeto como respetivamente **UdpCli** e **UdpSrv**, escolha a pasta apropriada para os alojar.

No projeto **UdpCli** substitua o conteúdo do ficheiro **UdpCli.java** pelo conteúdo correspondente fornecido. Proceda de forma equivalente para o projeto **UdpSrv**.

Proceda a uma análise do código fonte fornecido.

As classes UdpSrv e UdpCli definem o método main() pelo que podem ser executadas. Embora não fosse um requisito enunciado, o servidor quando recebe um pedido apresenta na consola sua origem (endereço IP e número de porto).

Teste 1 – servidor e cliente no mesmo nó

A - Colocar em funcionamento no posto de trabalho a aplicação servidora.

B - Executar a aplicação cliente no posto de trabalho:

- Uma vez que o servidor se encontra em funcionamento no mesmo nó de rede do cliente, podemos fornecer como endereço IP do servidor o **endereço de *loopback*** (**127.0.0.1**), também pode especificar este endereço através do nome ***localhost***.
- Enviar frases ao servidor e verificar os resultados devolvidos por ele. Na consola do servidor observar os endereços de origem dos pedidos.
- Para terminar a aplicação cliente escreva **sair**.
- A aplicação servidora não termina. Se o pretender terá de forçar a sua paragem (stop).

Teste 2 – servidor e cliente em nós diferentes

Para a sua aplicação servidor receber pedidos provenientes da rede, esse tráfego terá de ser permitido no firewall do posto de trabalho. Nas regras ***inbound*** (tráfego de entrada) terá de permitir tráfego UDP destinado ao número de porto 9999.

Os clientes necessitam de conhecer o seu endereço IP, pode ser obtido acedendo através de um browser ao URL **https://rede.dei.isep.ipp.pt/myip**. Forneça-o ao colega que vai comunicar com o seu servidor.

A - Colocar em funcionamento no posto de trabalho a aplicação servidora (ou manter em funcionamento se já estava em execução).

B - Executar a aplicação cliente no posto de trabalho:

- Fornecer à aplicação o endereço IP indicado pelo colega cujo servidor pretende utilizar.
- Repita os procedimentos do teste anterior, verificando os endereços de origem na consola do servidor.

Teste 3 – falhas do UDP

O UDP é não fiável, tanto o pedido como a resposta podem perder-se, isto tem impacto no funcionamento do cliente. A forma mais simples criarmos essa situação é forçar a paragem do servidor.

A - Colocar em funcionamento no posto de trabalho a aplicação servidora (ou manter em funcionamento se já estava em execução).

B - Executar a aplicação cliente no posto de trabalho:

- Fornecer à aplicação cliente o endereço do servidor local (***localhost***).
- Enviar a primeira frase ao servidor e aguardar a resposta.
- **Forçar a paragem do servidor (STOP).**
- Enviar uma segunda frase ao servidor.

A aplicação cliente ficou bloqueada porque está à espera de uma resposta (método **receive()**)que nunca irá chegar.

Terá de forçar a sua paragem (STOP).

Implementação de deteção de falhas no cliente UDP

Efetue as alterações necessárias no cliente UDP (UdpCli) para evitar este problema.

Em vez de o cliente ficar para sempre à espera de uma resposta do servidor, deve definir um tempo máximo para a resposta chegar.

Se a resposta não chega no tempo máximo estabelecido (***timeout***), a aplicação deve avisar o utilizador que o servidor não respondeu e continuar a funcionar normalmente.

O método **setSoTimeout()** das classes *Socket* (incluindo *DatagramSocket*) permite definir o tempo máximo em milissegundos para as operações de emissão e receção, sem o definir, as operações de receção bloqueiam até que cheguem dados.

Esgotado o *timeout* definido a operação de receção vai gerar uma exceção **SocketTimeoutException.**

Depois de concluir a implementação repita o **Teste 3** comprovando que a solução atinge o objetivo proposto.

Utilização de broadcast pelo cliente UDP

Quando um pacote UDP é enviado para o endereço de broadcast IPv4 (255.255.255.255), a rede deve entregar uma cópia do pacote a todos os nós que pertencem à mesma rede IP do nó emissor. O tráfego em broadcast não atravessa ou routers.

Muitos protocolos de aplicação utilizam estes envios para atingirem servidores quando não conhecem o respetivo endereço. É usado por exemplo pelo protocolo de aplicação DHCP para permitir ao cliente encontrar um servidor DHCP na rede local.

Pretende-se fazer exatamente o mesmo com o nosso cliente UDP:

- Em vez de pedir ao utilizador que indique o endereço do servidor, vamos enviar o primeiro pedido em broadcast.
- Quando for recebida a resposta ficamos a conhecer o endereço do servidor.
- Os pedidos seguinte além do primeiro já não devem ser enviados em broadcast, mas sim para o endereço do servidor que passou a ser conhecido.

Efetue as alterações necessárias no cliente UDP (UdpCli) para implementar estas novas funcionalidades.

UdpCliBcast project

Uma vez que esta aplicação será substancialmente diferente do UdpCli anterior, crie um novo projeto no NetBeans com o nome UdpCliBcast.

Copie integralmente o código atual do UdpCli.java para o UdpCliBcast.java do novo projeto. No UdpCliBcast.java corrija o nome do package para **udpclibcast** e o nome da classe para **UdpCliBcast** para corresponderem ao novo projeto.

Implemente as alterações necessárias no novo projeto para atingir os objetivos pretendidos.

Não esquecer que a permissão se emissão em broadcast tem de ser ativada usando o método **setBroadcast()** da classe DatagramSocket.

Teste a nova aplicação UdpCliBcast.

Problemas decorrentes da utilização de broadcast

Algumas infraestruturas de rede wireless não permitem a passagem de tráfego em broadcast com era suposto acontecer.

Dependendo do sucesso nos testes da aplicação UdpCliBcast os utilizadores poderão ter deparado com uma anomalia grave:

Depois de tudo correr bem com o primeiro pedido enviado, aparentemente as respostas aos pedidos seguintes são repetições da primeira resposta.

Para resolver o problema temos em primeiro lugar de perceber o que se está a passar.

Entendido o problema vamos procurar uma solução que se possa implementar para o resolver…