

# UDP Client

## Multicast

Corso Informatica classe 4<sup>a</sup>

Ambiente: .NET 2.0+

Anno 2016/2017

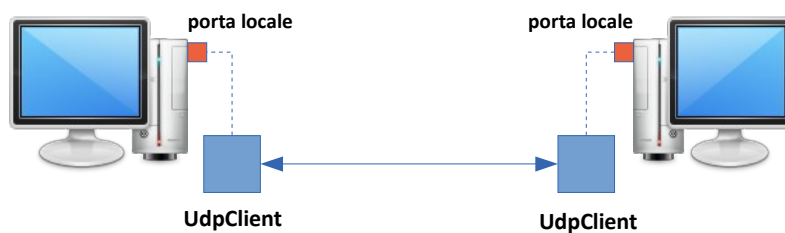
## Indice generale

<b>1</b>	<b>Comunicazione via <i>Udp</i></b>	<b>3</b>
1.1	Comunicazione “client-server”	3
1.2	Comunicazione “multicast”	3
<b>2</b>	<b>Introduzione a <i>UdpClient</i></b>	<b>4</b>
2.1	Creazione di un oggetto <i>UdpClient</i>	4
2.1.1	Specificare l'end point locale	4
2.2	Invio di datagrammi	4
2.3	Ricezione di datagrammi	4
2.4	Ricezione in un thread separato	4
<b>3</b>	<b>Comunicazione mediante “<i>multicast</i>”</b>	<b>6</b>
3.1	Registrazione su un indirizzo multicast	6
3.2	Invio di datagrammi ad un indirizzo multicast	7

# 1 Comunicazione via Udp

Il protocollo UDP implementa un tipo di comunicazione “non affidabile”; il protocollo non garantisce che i pacchetti (**datagrammi**) arrivino a destinazione e/o vengano ricevuti nello stesso ordine con il quale sono stati inviati. Non esiste il concetto di “connessione”, né di flusso dei dati (*stream*). La comunicazione avviene mediante singoli datagrammi, i quali sono memorizzati in vettori di byte.

L'invio e la ricezione di datagrammi sono realizzati mediante un oggetto di tipo `UdpClient`. Questo è in grado di ricevere datagrammi “mettendosi in ascolto” su una determinata porta (**porta locale**). Lo stesso oggetto può inviare datagrammi a un host remoto specificando il suo *endpoint* (indirizzo/nome host remoto + porta remota).



## 1.1 Comunicazione “client-server”

Data la natura del protocollo, non siamo di fronte al classico concetto di “client-server”, poiché un oggetto `UdpClient` può fungere sia da server che da client. Vale la regola: un oggetto che funziona da server (attende richieste e fornisce risposte), deve connettersi a una porta locale ben definita e dunque conosciuta dai client che vogliono comunicare con esso. Un oggetto che funziona da client può connettersi a una porta locale qualsiasi, con il vincolo che la stessa porta non può essere utilizzata da più oggetti contemporaneamente.

## 1.2 Comunicazione “multicast”

Il protocollo prevede due forme di comunicazione: **unicast** e **multicast**. Nella prima vi sono soltanto due soggetti coinvolti; nella seconda, i datagrammi possono essere spediti da un mittente e ricevuti contemporaneamente da più riceventi.

## 2 Introduzione a UdpClient

### 2.1 Creazione di un oggetto UdpClient

Perché un oggetto `UdpClient` possa inviare e/o ricevere messaggi è necessario che si colleghi a una porta locale. Se non ne viene fornita una, l'oggetto viene connesso a una porta locale scelta a caso dal SO e tenta di comunicare utilizzando la scheda di rete predefinita.

#### 2.1.1 Specificare la porta locale

È possibile specificare la porta locale direttamente nel costruttore:

```
UdpClient cli = new UdpClient(10000); // -> porta locale: 10000
```

Nel caso in cui l'host abbia più schede di rete, l'oggetto comunicherà attraverso la scheda di rete predefinita.

#### Host con più schede di rete

Se un host ha più schede di rete e si desidera ricevere i datagrammi da una qualsiasi di esse è necessario utilizzare un metodo diverso, specificando l'*end point* locale (indirizzo+porta) al quale collegare l'oggetto:

```
IPEndPoint epLocal = new IPEndPoint(IPAddress.Any, 10000);  
UdpClient cli = new UdpClient(epLocal); -> qualunque scheda, porta: 10000
```

`IPAddress.Any` sta per "qualsiasi indirizzo" e fa sì che l'oggetto possa ricevere i datagrammi attraverso una scheda di rete qualsiasi del computer locale.

### 2.2 Invio di datagrammi

L'invio di datagrammi richiede che venga specificato l'*endpoint* dell'host di destinazione (*end point* remoto). Un datagramma è rappresentato da un vettore di byte e viene spedito mediante il metodo `Send()`. Il seguente esempio invia la stringa "Hello!":

```
IPEndPoint epRemoto = new IPEndPoint(IPAddress.Parse("192.168.50.1"), 10000);  
byte[] data = Encoding.UTF8.GetBytes("Hello!");  
cli.Send(data, data.Length, epRemoto);
```

Nota bene: la stringa deve essere convertita in un vettore di byte; a questo scopo viene usato il metodo `GetBytes()` e la codifica UTF8.

Alternativamente, è possibile specificare l'*endpoint* remoto indicando separatamente il nome dell'host e la porta:

```
byte[] data = Encoding.UTF8.GetBytes("Hello!");  
cli.Send(data, data.Length, "WKS-2D2-01", 10000);
```

Oppure il suo indirizzo e la porta:

```
cli.Send(data, data.Length, "192.168.50.1", 10000);
```

## 2.3 Ricezione di datagrammi

Il metodo `Receive()` si mette in attesa di ricevere un datagramma e lo restituisce in un vettore di byte. Il metodo è bloccante e dunque l'esecuzione sarà sospesa fino a quando non viene ricevuto il pacchetto. (Oppure fino a quando non scade il timeout di lettura, se questo è stato impostato.) Il metodo richiede come argomento un oggetto `IEndPoint`, che valorizzerà con l'*endpoint* del mittente.

```
IPEndPoint ipe = null; // è necessario inizializzarlo (qualunque valore va bene)
byte[] data = cli.Receive(ref ipe);
```

Nota bene: anche se non si intende utilizzare l'*endpoint*, è necessario passarlo al metodo.

## 2.4 Ricezione in un thread separato

L'esecuzione di `Receive()` sospende l'esecuzione del thread fino all'arrivo di un pacchetto; ciò è un problema se l'oggetto `UdpClient` funziona da server e dunque si suppone che debba restare in attesa dei messaggi inviati dai client. In questo caso è necessario eseguire `Receive()` in un thread secondario, in modo da non bloccare il thread principale e dunque l'intero programma.

Il codice che segue implementa un server che produce un "eco" dei datagrammi ricevuti.

```
static void Main(string[] args)
{
    UdpClient server = new UdpClient(10000); // ascolta sulla porta 10000
    Task.Run(() => ServerEcho());
    ...
    Console.ReadKey();
}

private void ServerEcho()
{
    while (true)
    {
        IPEndPoint ipe = null;
        byte[] data = server.Receive(ref ipe); // ipe contiene l'endpoint del mittente
        server.Send(data, data.Length, ipe); // rispedisce il pacchetto al mittente.
    }
}
```

Nota bene: il metodo non termina mai, restando costantemente in attesa di datagrammi, che rispedisce al mittente. L'*end point* del mittente è memorizzato nella variabile `ipe`, valorizzata dal metodo `Receive()`.

Quello mostrato è un esempio di comunicazione *unicast*. Di seguito mostro come inviare e ricevere pacchetto attraverso un indirizzo *multicast*.

## 3 Comunicazione *multicast*

Il protocollo IP prevede un range di indirizzi dedicato alla comunicazione *multicast*, nella quale un datagramma può essere ricevuto contemporaneamente da più soggetti. In questo range esiste un sotto insieme di indirizzi locali (visibili soltanto all'interno della rete locale):

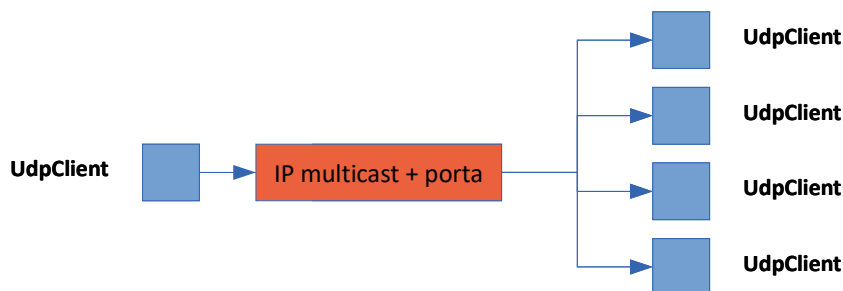
**224.0.0.0 - 224.0.0.255**

Esiste inoltre un sotto insieme di indirizzi globali (pubblici):

**224.0.1.0 - 238.255.255.255**

### 3.1 Registrazione su un indirizzo *multicast*

Nella comunicazione *multicast* i datagrammi non vengono inviati a un host ma a un indirizzo *multicast*: tutti processi "registrati" su quell'indirizzo riceveranno i datagrammi.



La registrazione avviene mediante il metodo `JoinMulticastGroup()`.

Nel codice seguente, un oggetto si registra su un indirizzo *multicast* locale e quindi resta in attesa di datagrammi, che si limita a visualizzare.

```
static void Main(string[] args)
{
    IPAddress ipMulticast = IPAddress.Parse("224.0.0.1");
    UdpClient receiver = new UdpClient(10000);
    receiver.JoinMulticastGroup(ipMulticast);

    Task.Run(() => ReceiverLoop());
    ...
    Console.ReadKey();
}

private void ReceiverLoop()
{
    while (true)
    {
        IPEndPoint ipe = null;
        byte[] data = receiver.Receive(ref ipe);
        string msg = Encoding.UTF8.GetString(data);
        Console.WriteLine(msg);
    }
}
```

Nota bene: la registrazione (metodo `JoinMulticastGroup()`) all'indirizzo *multicast* non è sufficiente; è necessario che l'oggetto `UdpClient` sia collegato a una porta ben definita, che sarà la stessa utilizzata dagli altri processi per l'invio dei datagrammi.

### 3.2 Invio di datagrammi a un indirizzo *multicast*

L'invio di datagrammi funziona con lo stesso meccanismo visto finora e, di per sé, non richiede registrazione. Il mittente deve semplicemente specificare l'*end point* remoto, che in questo caso è rappresentato da "indirizzo multicast + porta".

### 3.3 Invio e ricezione di datagrammi in *multicast*

Se l'applicazione deve sia inviare che ricevere datagrammi, è necessario utilizzare due oggetti `UdpClient` distinti, uno per l'invio, l'altro per la ricezione. I due oggetti devono essere collegati a porte locali distinte, e soltanto quello usato per la ricezione dovrà registrarsi all'indirizzo *multicast* mediante il metodo `JoinMulticastGroup()`.