Entendiendo la programación de Sockets UDP con GTK# y .NET

por Martín A. O Márquez <xomalli@gmail.com>

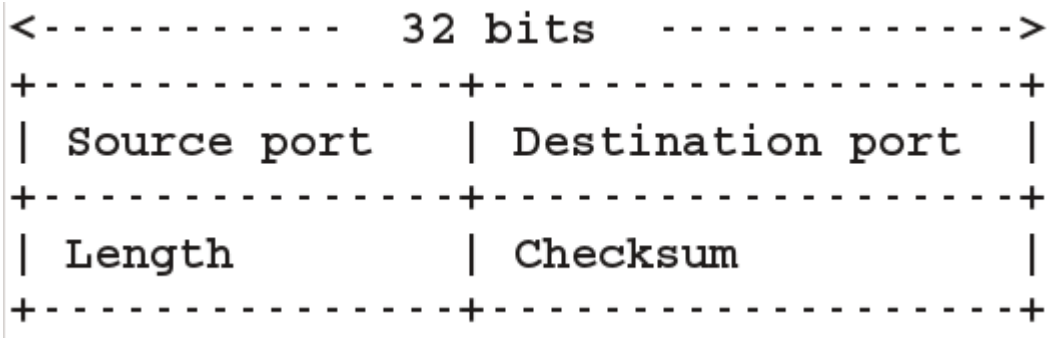
UDP (User Datagram Protocol) es un protocolo de datagramas, no orientado a la conexión mucho más sencillo, poco fiable y de alto rendimiento, que a diferencia de TCP no incluye todas características para garantizar la entrega de paquetes, tales como recuperación de fallas, chequeo de errores o control de flujo. Debido a estas características UDP debe utilizarse en casos específicos, donde se asuma que el rendimiento es mejor que la fiabilidad:

1. Cuando se transmite una pequeña cantidad de datos en periodos muy cortos de tiempo.
2. Aplicaciones de “consulta y respuesta”, que trasmiten una consulta y esperan una respuesta inmediata.
3. En sistemas de compresión para la transmisión de audio y video que pueden aceptar cierta cantidad de corrupción o perdida de paquetes.
4. Algunas aplicaciones tienen su propio mecanismo de entrega de transporte, así que no necesitan de una capa de transporte. Estas aplicaciones usaran UDP mas eficientemente que TCP.

Por su rendimiento, UDP es recomendable en aplicaciones en donde es aceptable que algunos paquetes se pierdan durante la comunicación, como video streaming de video o algún protocolo multicasting como DNS.

La cabecera UDP tiene solo 4 campos: puerto origen, puerto destino, longitud y checksum. Los campos origen y destino tiene la misma funcionalidad que se tienen en la cabecera TCP. La longitud del campo especifica la longitud de la cabecera UDP y el campo checksum permite la revisar integridad del paquete.

Fig 1 Los campos del paquete UDP.



La clase UDPClient

La clase **UdpClient** es responsable de manejar mensajes punto a punto (point-to-point) y multicasting con el protocolo UDP; la clase UdpClient utiliza endpoints, esta clase puede ser construida inicialmente con el parámetro de un host o bien este parámetro puede especificarse durante el método *Send()*.

Tambien proporciona un método *Receive()* que bloquea el hilo de ejecución hasta que un datagrama es recibido.

La clase **UdpClient** es una pequeña interfaz comparado a la clase **TcpClient**. Esto refleja la simple naturaleza de este protocolo en comparación con TCP. Mientras que ambas clases TcpClient y UdpClient utilizan un Socket internamente, la clase UdpClient no contiene un método para regresar un flujo de red (stream) para escribir y leer, en lugar de eso tiene un método *Send()* que acepta un arreglo de bytes como parámetro, mientras el método *Receive()* recibe un arreglo de bytes.

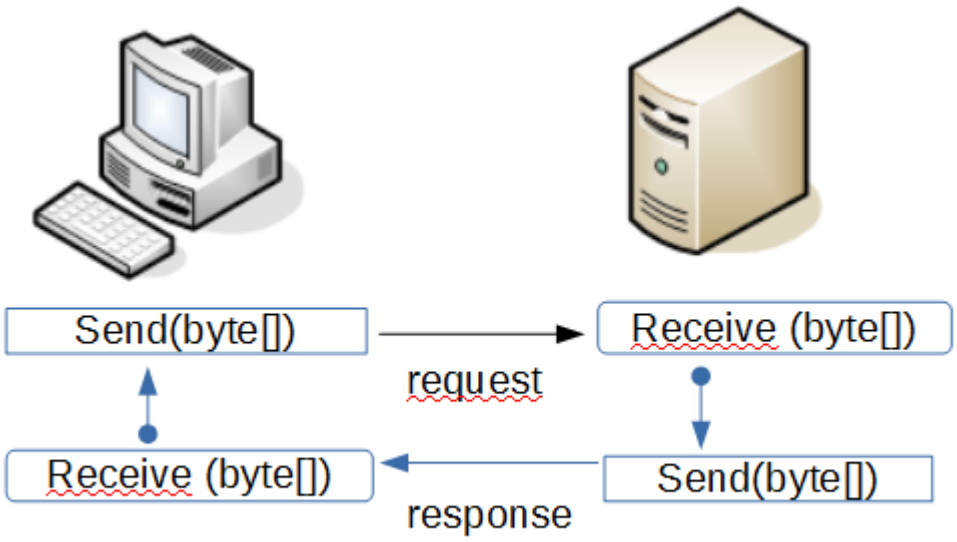
Fig 2 Miembros esenciales de la clase UDP Client.

Metodo o propiedad	Descripción
Receive	Recibe un datagrama UDP que fue enviado por una máquina remota.
Send	Envia un datagrama UDP a una máquina remota.
Client	Obtiene el socket subyacente que utiliza la clase.

Ejemplo de una comunicación entre un clientes y un servidor UDP

Es importante entender que en un modelo de comunicación sin conexión como UDP la comunicación se limita estrictamente a un modelo de petición/respuesta, que se lleva de la siguiente manera:

Fig 3 Modelo de comunicación petición/respuesta



Cada petición es un arreglo de bytes se salida procedente del cliente. Este arreglo de bytes se envía al servidor mediante TCP/IP, donde se convierte en la entrada del servidor. La respuesta entonces es el arreglo de bytes que salen del servidor para actuar como la entrada del cliente. Debido a la simpleza de este modelo, presenta las siguientes limitaciones:

- Falta de conexión continua
- Falta de fiabilidad
- Seguridad insuficiente

Para mostrar como trabaja la comunicación entre un cliente y un servidor UDP utilizando la clase UdpClient escribí dos programas: un servidor y un cliente UDP. Ambos programas utilizan una interfaz de usuario (GUI) en GTK# para comunicarse entre ellos.

Fig 4 Ejemplo de un time server UDP con una GUI GTK#

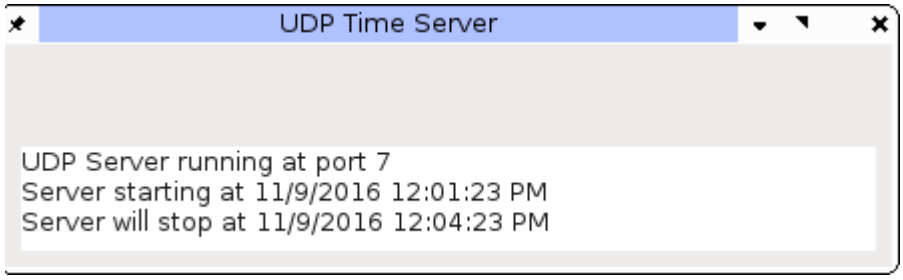
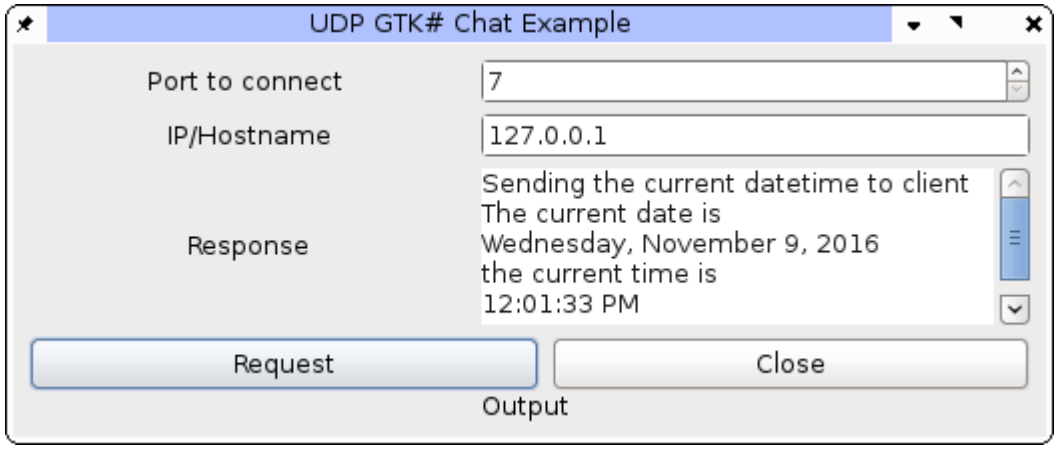


Fig 5 Ejemplo de un cliente UDP con una GUI GTK#



Pasos para la construcción de un servidor UDP GTK#

El servidor UDP envía la fecha y la hora actual de la máquina a los clientes que se conecten en su número de puerto, durante los minutos que el servidor se encuentre activo, ambos el número de puerto y los minutos son especificados por el usuario en la pantalla.

El proyecto del servidor se compone de dos clases:

1. La clase **MainWindowServer.cs** es la clase que construye la GUI del programa, obtiene los parámetros: número de puerto y minutos de ejecución del subproceso servidor, informa acerca de su estado o las posibles excepciones.
2. La clase **Program.cs** es la clase principal que donde se ejecuta el servidor.

Para construir un servidor UDP se requieren de los siguientes pasos:

1) Crear una objeto *UdpClient* que al construirse sin argumentos se le asigna un número de puerto de la IP local.

```
    UdpClient udpClient = new UdpClient();
```

2) Construir un objeto IPEndpoint asociando una IP para este ejemplo utilice la dirección local y el número de puerto donde se conectara para enviar la respuesta (response) del servidor.

```
    IPEndPoint IPremoteEP = new
IPEndPoint(IPAddress.Parse("127.0.0.1"),serverPort);
```

3) Utilizar la clase *Encoding* para convertir el texto de la respuesta en un arreglo de bytes UTF8, y enviarla por la red con el método *Send()*.

```
    byte[] data = Encoding.UTF8.GetBytes(response.ToString());
    udpClient.Send(data,data.Length,IPremoteEP);
```

Aquí código fuente completo de la clase MainWindowServer (Server)

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using Gtk;
using System.Threading;


namespace GtkSamples.Networking.UDPTimeServer
{
 public class MainWindow : Gtk.Window
 {
 VBox mainLayout = new VBox();
 Button btnStart = new Button("Start");
 HBox controlBox = new HBox();
 SpinButton sbtnPort = new SpinButton(7D, 10000D, 1D);
 SpinButton sbtnMinsToRun = new SpinButton(1D,11D,1D);
 TextView txtLog = new TextView();
 int serverPort = 1;

 public MainWindow() : base(WindowType.Toplevel)
 {
  this.Title = "UDP Time Server";
  this.SetDefaultSize(366,111);
  this.DeleteEvent += new DeleteEventHandler(OnWindowDelete);
  this.btnStart.Clicked += new EventHandler(Start);
  mainLayout.BorderWidth = 8;
  controlBox.BorderWidth = 8;
  controlBox.Spacing = 6;
```

```
          controlBox.PackStart(new Label("Port 1-10000"),false,true,0);
sbtnMinsToRun.ClimbRate = 1D;
sbtnMinsToRun.Numeric = true;
          controlBox.PackStart(sbtnPort,true,true,0);
controlBox.PackStart(new Label("Minutes to run (max 11)"),false,true,0);
sbtnMinsToRun.ClimbRate = 1D;
sbtnMinsToRun.Numeric = true;
controlBox.PackStart(sbtnMinsToRun, true, true, 0);
          controlBox.PackStart(btnStart,false,false,0);
mainLayout.PackStart(controlBox,false,true,0);
          mainLayout.PackStart(txtLog,true,true,0);
this.Add(mainLayout);
          this.ShowAll();
}

protected void OnWindowDelete(object o,DeleteEventArgs args)
{
System.Environment.Exit(Environment.ExitCode);
}

protected void Start(object o,EventArgs args)
{
try
{
Thread worker = new Thread(Run);
worker.IsBackground = true;
worker.Start();
}
catch (System.Exception ex)
{
txtLog.Buffer.Text += ex.Message;
}
}

void Run()
{
controlBox.Hide();
serverPort = Convert.ToInt32(sbtnPort.Text);
uint mins = Convert.ToUInt32(sbtnMinsToRun.Text);
DateTime timeToStart = DateTime.Now;
DateTime timeToStop = DateTime.Now.AddMinutes(mins);
//Step 1 create the UdpClient
UdpClient udpClient = new UdpClient();
StringBuilder buf = new StringBuilder();
buf.AppendFormat("UDP Server running at port {0}\n", serverPort);
buf.AppendFormat("Server starting at {0}\n", timeToStart.ToString());
buf.AppendFormat("Server will stop at {0}\n", timeToStop.ToString());
txtLog.Buffer.Text = buf.ToString();
do
{
StringBuilder response = new StringBuilder();
response.AppendLine("Sending the current datetime to client");
response.AppendLine("The current date is ");
response.AppendLine(DateTime.Now.ToLongDateString());
response.AppendLine("the current time is ");
response.AppendLine(DateTime.Now.ToLongTimeString());
//step 2 create the remote endpoint to send the data.
IPEndPoint IPremoteEP = new IPEndPoint(IPAddress.Loopback,serverPort);
//step 3 convert the string data to a byte array and sends it
```

```
  byte[] data = Encoding.UTF8.GetBytes(response.ToString());
  udpClient.Send(data,data.Length,IPremoteEP);
  } while (DateTime.Now < timeToStop);
  udpClient.Close();
  txtLog.Buffer.Text += "Server stopping at " + DateTime.Now.ToString();
  }
 }
}
```

El código fuente de la clase Program (Server)

```
using System;
using Gtk;

namespace GtkSamples.Networking.UDPTimeServer
{
 class MainClass
 {
 public static void Main(string[] args)
 {
  Application.Init();
  MainWindow win = new MainWindow();
  win.Show();
  Application.Run();
  }
 }
}
```

Pasos para la construcción de un cliente UDP GTK#

El cliente UDP solicita un petición al número de puerto del servidor, recibiendo la respuesta como una matriz de bytes, la convierte a una cadena y la muestra en la pantalla.

El proyecto del cliente UDP GTK# se compone de 2 clientes:

1. 1. La clase **MainWindow.cs** es la clase que construye la GUI del cliente, solicita los parámetros: numero de puerto y dirección IP para configurar la comunicación, maneja los eventos para recibir la respuesta del servidor y mostrar el resultado en pantalla o las posibles excepciones.
2. 2. La clase **Program.cs** es la clase principal donde se ejecuta el cliente

Para construir un cliente UDP se requieren de los siguientes pasos:

1) Se crea un objeto UdpClient y se le especifica como parámetro el número de puerto al cual se enlazará.

```
UdpClient client = new UdpClient(portToReceive);
```

2) Se crea el IPEndPoint para recibir la matriz de bytes con los parámetros IPAddress.Any (Indica que el servidor debe escuchar por clientes en todas las interfaces de red) y el número de puerto cero (para solicitar números de puertos dinámicos).

```
IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
```

3) Se obtiene el arreglo de bytes del IPEndPoint

```
    byte[] data = client.Receive(ref anyIP);
```

4) Se convierte el arreglo de bytes a una cadena

```
    text = Encoding.UTF8.GetString(data);
```

5) Una vez obtenido el mensaje se debe cerrar el Socket

```
    client.Close();
```

El código fuente completo de la clase MainWindow (cliente)

```
using System;
using Gtk;
using System.Net;
using System.IO;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace GtkSamples.Networking.UDPClient
{
    public class MainWindow : Gtk.Window
    {
    VBox mainLayout = new VBox();
 HBox buttonBox = new HBox();
    Table tableLayout = new Table(4,2,false);
 SpinButton spbtnPort = new SpinButton(7D, 10000D, 1D);
 Entry txtHostName = new Entry();
 TextView txtLog = new TextView();
 Button btnRequest = new Button("Request");
 Button btnClose = new Button("Close");
 Label lblMsg = new Label("Output ");
 int portToReceive = 0;
  ScrolledWindow scrollLog = new ScrolledWindow ();

  public MainWindow():base(WindowType.Toplevel)
        {
            this.Title = "UDP GTK# Chat Example";
   this.WidthRequest = 516;
            this.DeleteEvent += new DeleteEventHandler(OnWindowDelete);
   this.btnClose.Clicked += new EventHandler(Close);
   this.btnRequest.Clicked += new EventHandler(Request);
            mainLayout.BorderWidth = 8;
   tableLayout.RowSpacing = 6;
   tableLayout.ColumnSpacing = 6;
   tableLayout.Attach(new Label("Port to connect "), 0, 1, 0, 1);
   spbtnPort.Numeric = true;
   tableLayout.Attach(spbtnPort, 1, 2, 0, 1);
   tableLayout.Attach(new Label("IP/Hostname"), 0, 1, 1, 2);
   tableLayout.Attach(txtHostName, 1,2, 1, 2);
   tableLayout.Attach(new Label("Response"), 0, 1, 2, 3);
   txtLog.Editable = false;
   scrollLog.Add (txtLog);
   tableLayout.Attach(scrollLog, 1, 2, 2, 3);
   buttonBox.Spacing = 6;
```

```
    buttonBox.PackStart(btnRequest, true, true, 0);
    buttonBox.PackStart(btnClose, true, true, 0);

              mainLayout.PackStart(tableLayout,false,true,0);
    mainLayout.PackStart(buttonBox, false, true, 0);
    mainLayout.Add(lblMsg);
              this.Add(mainLayout);
              this.ShowAll();
         }

   protected void OnWindowDelete(object o, DeleteEventArgs args)
   {
    System.Environment.Exit(Environment.ExitCode);
   }

   protected void Close(object o, EventArgs args)
   {
    System.Environment.Exit(Environment.ExitCode);
   }

   protected void Request(object o, EventArgs args)
   {
     if (string.IsNullOrEmpty(txtHostName.Text))
    {
     lblMsg.Text += "Hostname null";
    }
    else
    {
     portToReceive = Convert.ToInt32(spbtnPort.Text);
     //step 1 Create a UdpClient and specifies a port number to bind
     UdpClient client = new UdpClient(portToReceive);
     string text = null;
     try
     {
      //step 2 Create the IPEndPoint to receive the byte array
      IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
      //step 3 Get the byte array
      byte[] data = client.Receive(ref anyIP);
      //step 4 Convert the byte array to string
      text = Encoding.UTF8.GetString(data);
      txtLog.Buffer.Text = text;
     }
     catch (System.Exception ex)
     {
      lblMsg.Text += ex.Message;
     }
     finally
     {
      //step 5 Close the Socket
      client.Close();
     }
    }
   }

    }
}
```

El código fuente de la clase Program (cliente)

```
using System;
using Gtk;

namespace GtkSamples.Networking.UDPClient
{
 class MainClass
 {
  public static void Main(string[] args)
  {
   Application.Init();
   MainWindow win = new MainWindow();
   win.Show();
   Application.Run();
  }
 }
}
```

A continuación unas imágenes del cliente y del servidor comunicándose entre si.

Fig 6 Configurando el servidor de fecha y hora

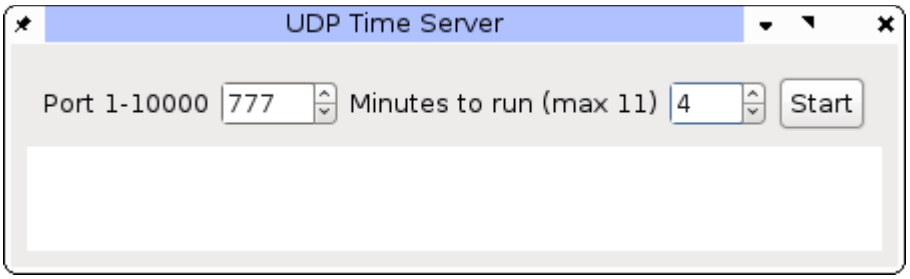


Fig 7 Ejecutando el servidor de fecha y hora

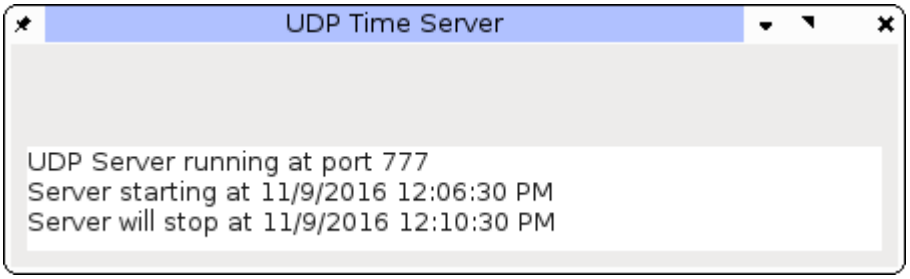


Fig 8 Ejecutando el cliente GTK# Udp

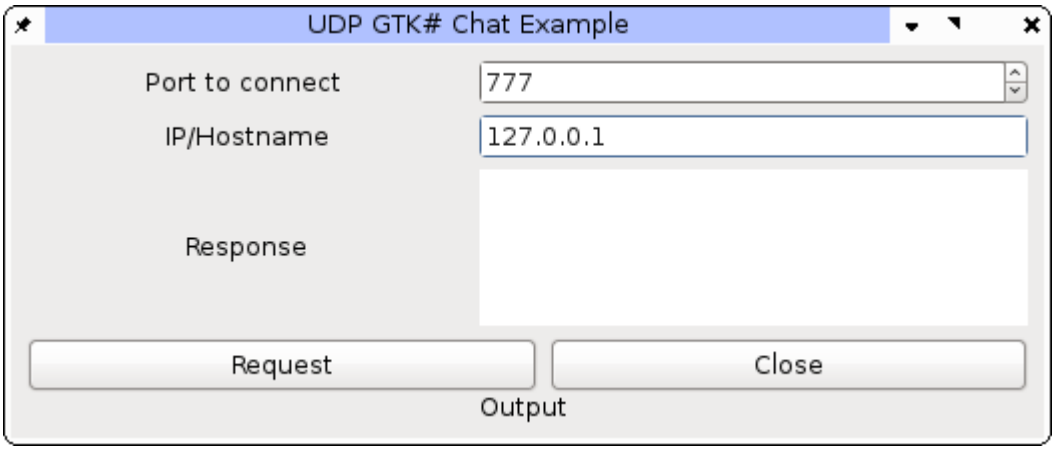
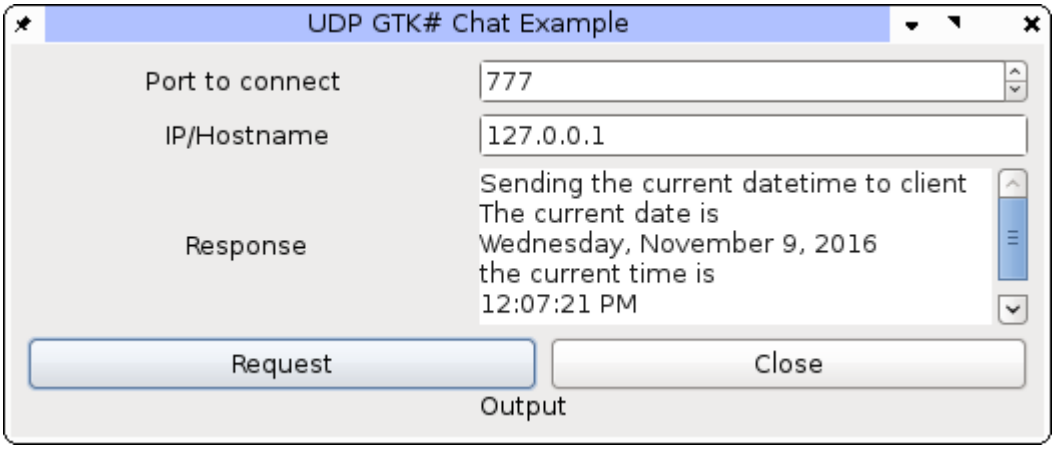


Fig 9 Obteniendo la respuesta de servidor UDP



Download el proyecto GTK# Udp Server para Xamarin o Visual Studio

Download el proyecto GTK# Udp Client para Xamarin o Visual Studio