Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

LABORATORIO DE TECNOLOGÍAS DIGITALES

Practice 5: Video Conference using UDP Multicast communications.

Professor:

Antonio Javier García Sánchez.

In this practice, a **Video Conference using UDP Multicast communications.** The functionality of our application is summarized in the following points:

- A Video Server captures images and broadcasts them through a multicast group.
- The messages that make up the images are sent via UDP to each of the clients.
- Each client of the multicast group receives the messages, forms the image and presents it to the user.

This application needs to use the library *DirectShow* for the treatment and delivery of the image in real time. In the version of Visual Studio 2013, this library is included.

Figure 1 represents the scheme to be implemented by the student:

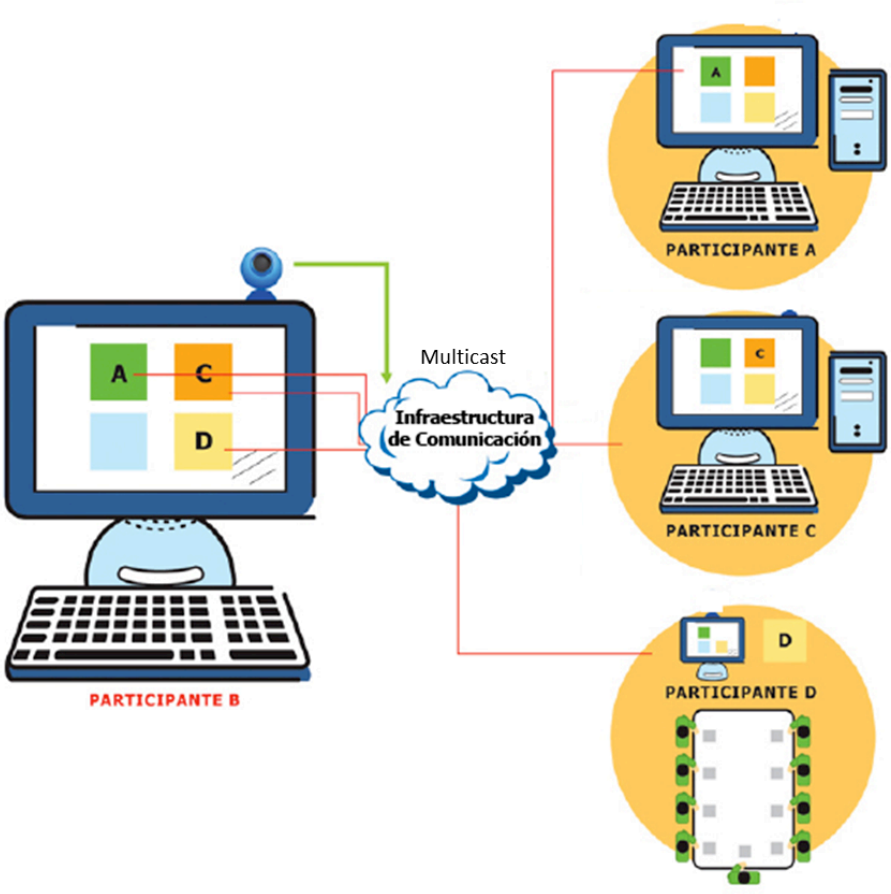

Figure 1. UDP Multicast Video Conference

1.- Video Streaming Server

For the realization of the server station we need to create an application that obtains the image in real time from the webcam, connects to the multicast group and sends the images that are obtained from the camera through the UDP protocol.

A) Video Capture through a WebCam

The *using a* add is as follows: using System.Drawing.Imaging.

First, the student must detect the number and characteristics of the web cameras that the PC has:

```
Camera cam in CameraService .AvailableCameras
```

Once the camera has been selected (for example, in a *comboBox),* capture of images by webcam should be started. For this, various parameters are established such as the size of the captured image or the number of images per second.

```
private CameraFrameSource _frameSource;
Camera c = ( Camera ) comboBoxCameras.SelectedItem; setFrameSource ( new CameraFrameSource
(c));
_frameSource.Camera.CaptureWidth = 320;
_frameSource.Camera.CaptureHeight = 240;
_frameSource.Camera.Fps = 20;
_frameSource.NewFrame + = OnImageCaptured;
```

With this last line we guarantee that an image is captured from the webcam. The method code is as follows:

```
public void OnImageCaptured (Touchless.Vision.Contracts. IFrameSource
frameSource, Touchless.Vision.Contracts. Frame frame, double fps)
                {
                        _latestFrame = frame.Image;
                        pictureBoxDisplay.Invalidate ();
                }
```

With the event handler *PaintEventHandler* we create a method *drawLatestImage,*
which is responsible for generating an image *Bitmap.* This is essential to be able to: (i) view the image and (ii) be able to send it in multicast mode.

```
private static Bitmap _latestFrame;
private void drawLatestImage ( object sender, PaintEventArgs e) {

                        if (_latestFrame! = null ) {

e.Graphics.DrawImage (_latestFrame, 0, 0, _latestFrame.Width, _latestFrame.Height);

. . . .
                }
```

The image is displayed through the element *pictureBoxDisplay*

```
pictureBoxDisplay.Paint + = new PaintEventHandler (drawLatestImage); _frameSource.StartFrameCapture ();
```

B) UDP Multicast Communications

The requirements that you must implement are:

- Add yourself to the multicast group.
- Send images in JPEG format to the client.

B.1 Create a multicast group.

The *using* to be added are:

```
using System.Net;
using System.Net.Sockets;
using System.IO;
```

To create a multicast group the object *UdpClient. UdpClient* provides network services using the User Datagram Protocol (UDP).

Within its methods is selected *JoinMulticastGroup,* that internally provides the creation and joining the multicast group:

```
UdpClient udpserver = new UdpClient ();
IPAddress multicastaddress = IPAddress.Parse ("224.0.0.1"); udpserver.JoinMulticastGroup
(multicastaddress);
```

IPEndPoint represents a network endpoint as an IP address and a port number. In the server case, said endpoint is started with the multicast address and port of the video client.

```
IPEndPoint remote = new IPEndPoint (multicastaddress, UDP_PORT);
```

B.2 Sending images.

The Video Server application sends the messages to the Client using the method *Send* from *UdpClient.* A datagram is sent to the remote end with the following syntax:

```
public int Send (byte [], int, IPEndPoint) ;
udpServer.Send (buffer, buffer.Length, remote);
```

In the argument *buffer* contains the image in JPEG format. This argument only handles **byte array** therefore, the image in JPEG format will have to be converted into this type. For this you can use the method *Toarray ().*

Another aspect to consider is how to handle each video image that the camera captures and convert it into JPEG frames. First, the student will become familiar with the class *MemoryStream,* which acts as a streaming data manager by creating an object. This object is what you use to convert the data set that makes up the image in the JPEG format. For this the student will use the method *Save* already seen in the practice of the "Image Converter" and will run it on the image captured from the camera. The following diagram summarizes the indicated process:

2.- Video client.

When the client application starts, it must connect to the multicast group, obtain the image sent by the server and display it on the screen. All this on a recurring basis until the server stops sending images.

A) UDP Multicast Communications

The class is used as in the server case *UdpClient,* Now taking into account that you have to add yourself and send the messages to that group. For this, in addition to the steps performed on the server, the class *IPEndPoint* as follows:

```
IPEndPoint remoteep = new IPEndPoint ( IPAddress .Any, UDP_PORT);
```

That is, the client takes your IP and will receive the data on the designated port.

In addition, so that the client can accept several connections, the student must implement the methods *Client.SetSocketOption* Y *Client.Bind* of the class *UdpClient.*

The client is blocked until receiving image data from the Video Server application. The method is used for this *Receive* inside the class *UdpClient* which receives an array of bytes.

```
Byte [] buffer = UdpCliente.Receive ( ref localEp);
```

This data is handled by the class *MemoryStream* in order to convert them into an image. To do this, once the object is created *MemoryStream* which must contain the image data, the method is used *FromStream* of the class *Image.*
.NET automatically detects that the image is in JPEG format.

B) Viewing images

Simply with the method *Image* from *pictureBoxDisplay,* images can be viewed in JPEG format.

NOTE: As mentioned, the server application is blocked receiving the data from the clients, causing the graphical environment not to be executed. To solve this problem, tasks managed by the CLR will be implemented, which decides if the application should use execution threads, how many and when they will be launched. We have an example in the following code block:

```
Task t1 = new Task (display_image);
```

```
t1.Start ();
private void display_image ()
                {
                        while ( true )
                        {
                                try
                                {
                                .....
                            }
```