Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

LABORATORIO DE CONTENIDOS DIGITALES

Práctica 4: SERVICIO DE VIDEOCONFERENCIA USANDO COMUNICACIONES MULTICAST UDP. EVALUACIÓN DE PRESTACIONES

Profesor:

Antonio Javier García Sánchez.

En esta práctica se va a implementar una **Video Conferencia usando comunicaciones Multicast UDP**. La funcionalidad de nuestra aplicación se resume en los siguientes puntos:

- Un Servidor de Video captura imágenes y las difunde a través de un grupo Multicast.
- Los mensajes que forman las imágenes se envían vía UDP a cada uno de los clientes.
- Cada cliente del grupo Multicast recibe los mensajes, forma la imagen y la presenta al usuario.

Esta aplicación necesita utilizar la librería *DirectShow* para el tratamiento y envío de la imagen en tiempo real. En la versión de Visual Studio 2017, esta librería se encuentra incluida.

La figura representa el esquema a implementar por el alumno :

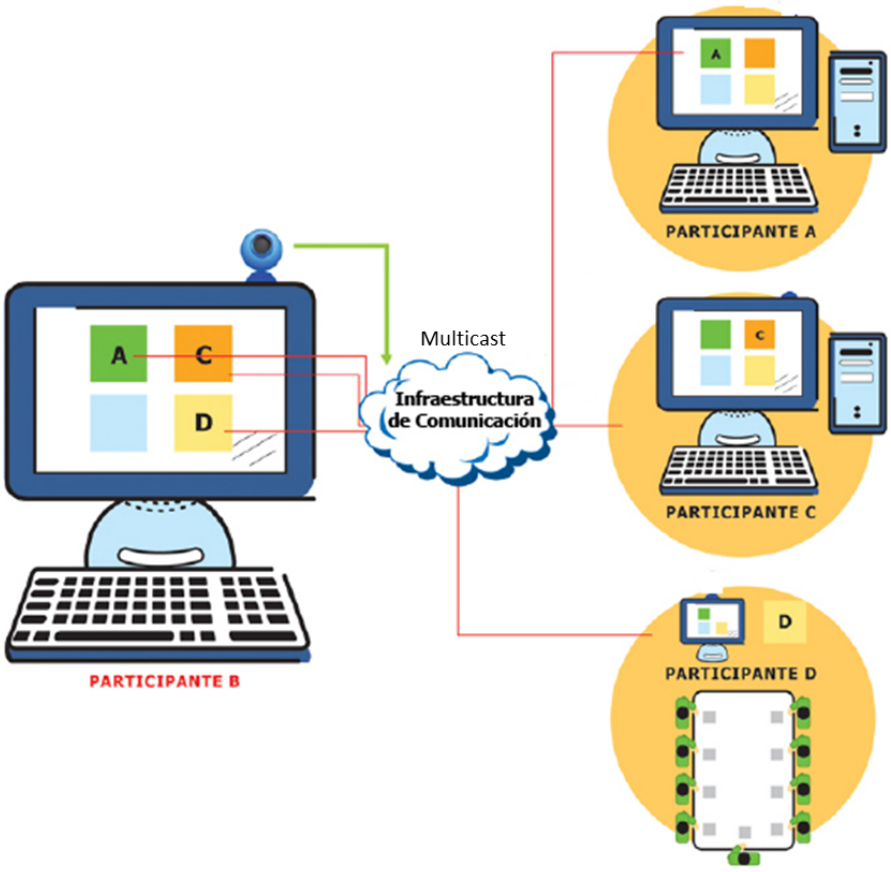


Figura. Video Conferencia Multicast UDP

1.- Video Streaming Server

Para la realización de la estación servidora necesitamos crear una aplicación que obtenga la imagen en tiempo real de la cámara web, se conecte al grupo multicast y envíe mediante protocolo UDP las imágenes que se vayan obteniendo de la cámara.

A) Captura de Video a través de una WebCam

La *using a* añadir es la siguiente: using System.Drawing.Imaging.

En primer lugar, el alumno deberá detectar el número y características de las cámaras web que el PC posee:

```
foreach(Camera cam in CameraService.AvailableCameras)
```

Se añadirán las siguientes variables globales:

```
private CameraFrameSource _frameSource;
private static Bitmap _latestFrame;
```

Una vez seleccionada la cámara (por ejemplo en un *comboBox* y haciendo uso de la instrucción *foreach* indicada arriba), se debe iniciar la captura de imágenes por la cámara web. Para ello se establece diversos parámetros como el tamaño de la imagen capturada o el número de imágenes por segundo.

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

La imagen es visualizada a través del elemento *pictureBoxDisplay*

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

Con esta última línea nos garantizamos que se capture una imagen de la cámara web. El código del método es el siguiente:

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

Con el manejador de eventos *PaintEventHandler* creamos un método *drawLatestImage,* el cual es el encargado de generar una imagen *Bitmap*. Esto es esencial para poder: (i) visualizar la imagen y (ii) poder enviarla en modo Multicast.

```
private static Bitmap _latestFrame;
private void drawLatestImage(object sender, PaintEventArgs e)
        {
            if (_latestFrame != null)
            { // Aquí se deberá redimensionar la imagen _latestFrame a 320x240
e.Graphics.DrawImage(_latestFrame, 0, 0, _latestFrame.Width,
_latestFrame.Height);
// Aquí se insertaría el código para enviar imágenes
        }
```

Para redimensionar la imagen se podrá generar otro Bitmap con un nuevo tamaño despecificado:

```
Bitmap(bitmap_old, new Size(new_width, new_height));
```

B) Comunicaciones Multicast UDP

Los requisititos que debe implementar son:

- Añadirse al grupo multicast.
- Enviar imágenes en formato JPEG al cliente.

B.1 Crear un grupo multicast.

Los *using* que habrá que añadir son:

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

Para crear un grupo multicast se va a utilizar el objeto *UdpClient*. *UdpClient* proporciona servicios de red mediante el protocolo de datagramas de usuarios (UDP).

Dentro de sus métodos se selecciona *JoinMulticastGroup*, que internamente proporciona la creación y adhesión al grupo multicast:

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

IPEndPoint representa un punto final de red como una dirección IP y un número de puerto. En el caso servidor se inicia dicho punto final con la dirección Multicast y puerto del cliente de video.

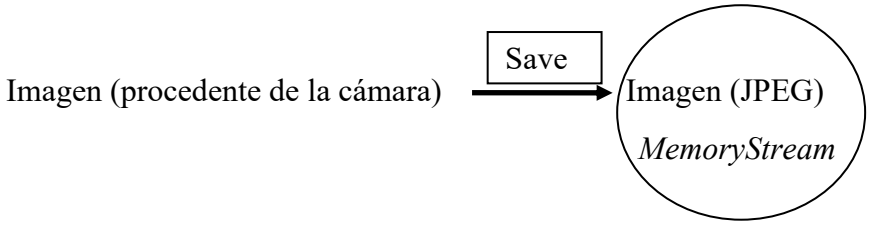
```
IPEndPoint remote=new IPEndPoint(multicastaddress, UDP_PORT);
```

B.2 Envío de imágenes.

La aplicación Video Server envía los mensajes al Cliente usando el método *Send* de *UdpClient.* Se envía un datagrama al extremo remoto con la siguiente sintaxis:

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

Otro aspecto a tener en cuenta es como manejar cada imagen de video que captura la cámara y convertirlo en fotogramas JPEG. En primer lugar el alumno se familiarizará con la clase *MemoryStream*, la cual hace de manejador de datos streaming creando un objeto. Este objeto es el que utiliza para convertir el conjunto de datos que forman la imagen en el formato JPEG. Para ello el alumno utilizará el método *Save* ya visto en la práctica del "Convertidor de Imágenes" y lo ejecutará sobre la imagen capturada de la cámara pasándole un objeto de tipo *MemoryStream* e indicándole una conversión a JPEG. El siguiente esquema resume el proceso indicado:



En el argumento *buffer* contiene la imagen en formato JPEG. Este argumento solo maneja **array de bytes** por tanto habrá que convertir la imagen en formato JPEG en este tipo. Para ello se puede usar el método *Toarray()* del objeto de tipo *MemoryStream.*

2.- Cliente de vídeo.

La aplicación cliente al iniciarse se debe conectar al grupo multicast, obtener la imagen enviada por el servidor y mostrarla por pantalla. Todo ello de forma recurrente hasta que el servidor deje de enviar imágenes.

A) Comunicaciones Multicast UDP

Se utiliza como en el caso servidor la clase *UdpClient*, teniendo ahora en cuenta que hay que añadirse y enviar los mensajes a dicho grupo. Para ello además de los pasos realizados en el servidor se utiliza la clase *IPEndPoint* de la siguiente manera:

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

Es decir, el cliente toma su IP y recibirá los datos en el puerto designado.

Además para que el cliente pueda aceptar varias conexiones, el alumno deberá implementar los métodos *Client.SetSocketOption* y *Client.Bind* de la clase *UdpClient*.

El cliente se queda bloqueado hasta recibir datos de imágenes de la aplicación Video Server. Se usa para ello el método *Receive* dentro de la clase *UdpClient* que recibe un array de bytes.

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

Estos datos son manejados por la clase *MemoryStream* con el objeto de poder convertirlos en una imagen. Para ello, una vez creado el objeto *MemoryStream* el cual debe contener

los datos de la imagen, se usa el método *FromStream* de la clase *Image.* Automáticamente .NET detecta que la imagen está en formato es JPEG.

B) Visualización de imágenes

Simplemente con el método *Image* de *pictureBoxDisplay,* se podrá visualizar las imágenes en formato JPEG.

NOTA: Como se ha comentado la aplicación servidora queda bloqueada recibiendo los datos de los clientes, provocando que el entorno gráfico no se ejecute. Para solucionar este problema se va implementar tasks gestionado por el CLR, el cual decide si la aplicación debe utilizar hilos de ejecución, cuantos y cuándo serán lanzados. Un ejemplo lo tenemos en el siguiente bloque de código:

```
Task t1 = new Task(visualizar_imagen);
t1.Start();
private void visualizar_imagen()
        {
            while (true)
            {
                try
                {
                ……….
               }
```

3.- Evaluación de Prestaciones de Servicio de Video.

En esta segunda parte del proyecto, el alumno tendrá que enviar paquetes de video insertado en un protocolo de comunicaciones. Para ello, diseñará una cabecera que como mínimo tenga los campos *número de imagen*, *número de secuencia del paquete recibido dentro de esa imagen y timestamp*. Cada paquete estará formado por estos campos más el *payload* de información de video obtenido del método *ImageToByteArray(_latestFrame)* del objeto de tipo *MemoryStream,* en particular *Byte[] buffer* . El alumno deberá:

1) En el emisor, obtener el tamaño óptimo de payload, que haga que la secuencia de imágenes se visualice correctamente.
2) En el receptor, por una parte, se leerán las cabeceras con objeto de calcular las métricas de latencia y jitter, además del número de paquetes perdidos. En ese caso, se valorará que esta presentación sea de forma gráfica. Por otra, el alumno deberá componer la imagen JPEG con el payload de los paquetes.
3) Se valorará que el alumno implemente el protocolo RTP como protocolo de comunicaciones.

Se valorará la inclusión de los ANEXOS 1 Y 2 con objeto de observar el cambio de prestaciones.

ANEXO 1. CHAT MULTICAST UDP

1.- Aplicación Servidora.

Los requisitos son los siguientes:

- Crear un grupo multicast.
- Recibir los datos de todos los clientes que se añaden al grupo multicast.
- Presentar en una aplicación los mensajes enviados por un cliente.

1.1 Crear un grupo multicast.

Los *using* que habrá que añadir son:

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

Para crear un grupo multicast se va a utilizar el objeto *UdpClient*. *UdpClient* proporciona servicios de red mediante el protocolo de datagramas de usuarios (UDP).

Dentro de sus métodos se selecciona *JoinMulticastGroup*, que internamente proporciona la creación y adhesión al grupo multicast:

```
UdpClient udpclient = new UdpClient(8080);
IPAddress multicastaddress=IPAddress.Parse("224.0.0.1");
udpclient.JoinMulticastGroup(multicastaddress);
```

IPEndPoint representa un punto final de red como una dirección IP y un número de puerto. En el caso servidor no se tiene que unir a ningún otro servidor, la referencia es nula.

```
IPEndPoint remote=null;
```

1.2. Recepción de los datos.

El servidor queda bloqueado hasta recibir datos de los clientes. Se usa para ello el método *Receive* dentro de la clase *UdpClient* que recibe un array de bytes.

1.3. Representación de datos.

Los textos provenientes de los distintos clientes deben ser impresos por pantalla. La representación se realiza con un *ListBox* y con un *Textbox*:

- *ListBox*, que contiene un historial de todos los mensajes que han llegado
- *Textbox,* que contiene el último mensaje recibido.

El servidor recibe un array de bytes en el método *Receive*. Los componentes *ListBox* y *Textbox* soportan *strings.* El paso a *strings* sin utilizar punteros ni array de char, está solucionado en *.Net* con el uso de la clase *Encoding*. *Encoding* representa una codificación de carácter. Dentro de esta clase, se utiliza el tipo *Unicode* que codifica cada carácter como dos bytes consecutivos.

NOTA: Como se ha comentado la aplicación servidora queda bloqueada recibiendo los datos de los clientes, provocando que el entorno gráfico no se ejecute. Para solucionar este problema se va implementar un hilo de ejecución con el código de recepción de datos de los clientes.

La implementación de *threads* se realiza con la clase *Thread*, añadiendo el siguiente *using*:

```
using System.Threading;
```

Un ejemplo sería el siguiente:

```
Thread t = new Thread(new ThreadStart(MyThreadMethod));
t.Start();

private void MyThreadMethod()
            {
                  while(true)
                  {
                     .
                     //Código de recepción de datos
                     .
                  }
```

2.- Aplicación Cliente.

Los requisititos que debe implementar son:

- Añadirse al grupo multicast.
- Enviar los mensajes al cliente.

2.1. **Añadir a un grupo multicast.**

Se utiliza como en el caso servidor la clase *UdpClient*, teniendo ahora en cuenta que hay que añadirse y enviar los mensajes a dicho grupo. Para ello además de los pasos realizados en el servidor se utiliza la clase *IPEndPoint* de la siguiente manera:

IPEndPoint `remoteep = new IPEndPoint( multicastaddress, 8080 );`

2.2. Interfaz gráfica.

Usaremos el componente *RichTexBox* que permite al usuario escribir textos en pantalla. Estos textos son *strings* a enviar.
Crearemos un **botón** para enviar los mensajes al servidor.

2.3. Envío de mensajes.

El cliente envía los mensajes al servidor usando el método *Send* de *UdpClient.* Se envía un datagrama al extremo remoto con la siguiente sintaxis:

public int Send(byte[], int, IPEndPoint);

ANEXO 2. TRANSMISIÓN DE AUDIO MEDIANTE CODIFICACIÓN A-LAW

El objetivo de esta práctica es la implementación de una transmisión de audio mediante codificación *A-law*. La funcionalidad de nuestra aplicación es la siguiente:

- Transmisión de Audio mediante un micrófono a un host remoto usando UDP como protocolo de comunicaciones y *A-law* como codificador de audio.
- Recepción de audio y reproducción de éste mediante unos altavoces.

La figura 2 representa las aplicaciones a crear:

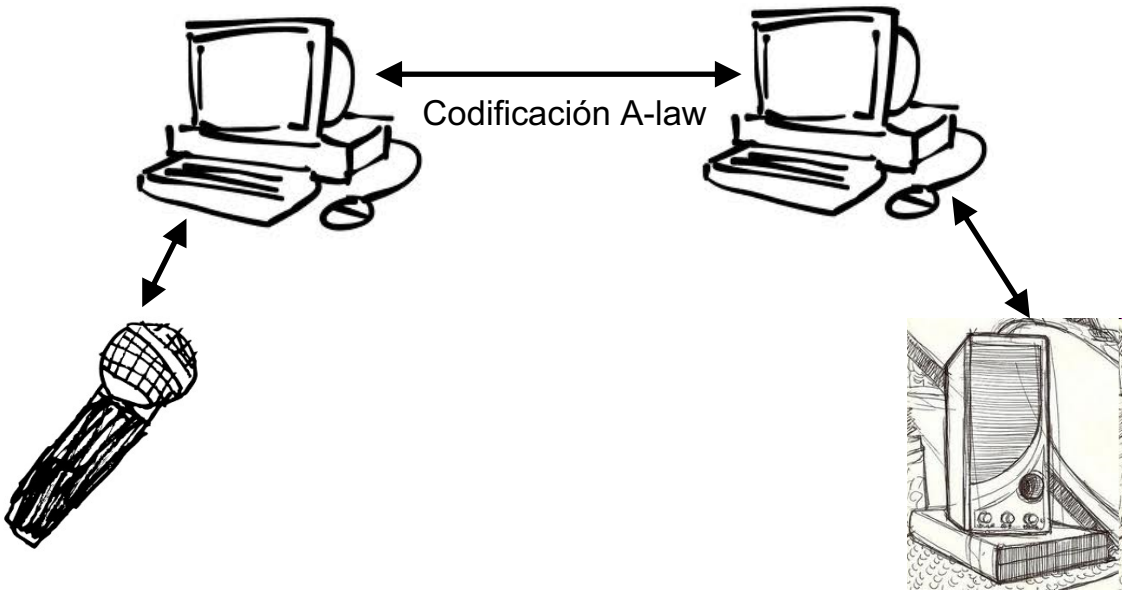


Figura 2. Transmisión Audio

1. Transmisión/Recepción de Audio.

Como comentario general a este apartado se cargarán las siguientes *using*:

```
using ALaw;
using NAudio.Wave;
using NAudio.CoreAudioApi;
```

NOTA: Es necesario tener instalado el paquete Naudio

1.1 Establecimiento de datos

En primer lugar se va a establecer el formato y las propiedades de la onda de audio mediante la clase *waveformat*. El alumno deberá conocer los siguientes miembros de esta clase: *Channels, FormatTag, SamplesPerSecond, BitsPerSample, BlockAlign, AverageBytesPerSecond*.

En el diseño de los miembros anteriores, se aconseja utilizar los siguientes parámetros:

- `short channels = 1; //Stereo.`
- `short bitsPerSample = 16; //16Bits`
- `int samplesPerSecond = 22050; //22KHz`
- `Codificación PCM`

Para ello se puede implentar la siguiente función:

```
private void listBox1_SelectedIndexChanged(object sender,
EventArgs e)
        {
            WaveIn waveIn = new WaveIn();
            int inputDeviceNumber = listBox1.SelectedIndex;
            waveIn.BufferMilliseconds = 50;
            waveIn.DeviceNumber = inputDeviceNumber;
            waveIn.WaveFormat = new WaveFormat(8000, 16, 1);
            waveIn.DataAvailable += OnAudioCaptured;
            waveIn.StartRecording();
        }
```

1.2. Captura de audio desde un micrófono

```
void OnAudioCaptured(object sender, WaveInEventArgs e)
    {
      byte[] encoded = ALawEncoder.ALawEncode(e.Buffer);
      udpSender.Send(msg, msg.Length, endPoint);
    }
```

Será tan sencillo como implementar la codificaión ALaw y su envío remoto.

NOTA 1: El código del codificador *A-law* será proporcionado por el profesor.
NOTA 2: El codificador *A-law* reduce el tamaño del buffer a la mitad.

1.3. Reproducción de audio desde un altavoz

En el reproductor de audio habrá que crear los objetos WaveOut y Waveformat con los valores indicados en el lado de la captura de audio.

```
waveOut = new WaveOut();
            waveProvider = new BufferedWaveProvider(new
WaveFormat(8000, 16, 1));
            waveProvider.DiscardOnBufferOverflow = true;
```

Y para reproducer en el altavoz se puede implementar una función tal que así:

```
private void ListenerThread(object state)
        {
            var endPoint = (IPEndPoint)state;
            try
            {
                waveOut.Init(waveProvider);
                waveOut.Play();
                while (listening)
                {
                    byte[]   b   =   clientAudio.Receive(ref
endPoint);
                    byte[] payload = deserializeHeader(b,
false); // Implementar por el alumno
                    short[]            decoded            =
ALawDecoder.ALawDecode(payload);
                    byte[] result = new byte[decoded.Length
* 2];
                    Buffer.BlockCopy(decoded, 0, result, 0,
result.Length);
                    waveProvider.AddSamples(result,      0,
result.Length);
                }
            }
            catch (SocketException)
            {
            }
        }
```

NOTA: El código de la decodificación de audio mediante *A-law* será proporcionado por el profesor.