طريقك لعمل Voice Chat

By Using Microsoft Visual C#

فصل من كتاب احترف برمجة الشبكات الإصدار الثاني
رابط الإصدار الثاني
http://www.enashir.com/blogs/fadi

تأليف فادي عبد القادر

Chapter 11
Voice Over IP Programming

A. The Concept & Requirements of Voice Communication Systems

B. How to Create a Voice Chat Throw Dot Net Using Unmanaged API's Functions

C. Testing UDP Multicasting, TCP and Thinking in SCTP to Transfer Voice Throw Networks

D. How to Create a Voice Conference System Using Microsoft Direct Play 9

بسم الله الرحمن الرحيم

المقدمة:

تتلخص الفكرة الأساسية من نقل الصوت عبر بروتوكول الإنترنت IP بتحويل الصوت إلى مجموعة من الـ Bits تجمع في Byte Array ثم كبسلته ليتم نقله ك Datagram Packets عبر الشبكة ، وللاستقبال الصوت في الطرف الأخر يتم تجميع ال Packets مرة أخرى في مصفوفة Byte Array ، وتتم عملية القراءة وفق مبدأ الـ FIFO – First In First Out أي القادم أولا يعرض أولا ...

تكمن المشكلة الأساسية بنقل الصوت في مدى توفر الشروط اللازمة حتى يتم إيصال وعرض الصوت بالشكل السليم و وفق الترتب الذي أرسل عليه ، وتعتبر محدودبات ومشاكل بروتوكولات ال Transport Layer من أهم ما دعا Microsoft من العزوف عن دعم الـ Dot Net لعملية نقل الصوت وخاصة في بيئة النظام الحالي ، ومن المعروف أن نظام التشغيل Windows XP يدعم الاتصال باستخدام بروتوكول TCP أو UDP فقط وهذا يعني انك إذا كنت تعمل تحت منصة نظام التشغيل Windows XP فإن أي عملية اتصال لن تكون إلا باستخدام واحد من هذه البروتوكولات ...

أولا : The Requirements of Voice Communication Systems

سوف نناقش في هذا الجزء متطلبات نقل الصوت عبر الشبكة ومشاكل نقل الصوت باستخدام بروتوكول الـ TCP و الـ UDP ...

– متطلبات نقل الصوت المثلى :

1 - أسلوب النقل Stream

2 – البروتوكول المستخدم لنقل الصوت يجب أن يدعم Delivered on Sequence

3 – تعتمد سرعة النقل على مدى حجم الضغط المستخدم Voice Compression والجودة المطلوبة ويفضل في هذه الحالة أن لا تقل سرعة النقل عن 31 KB\S بمعدل لا يقل عن 8.000 KHz كحد أدنى لجودة الصوت.

- السؤال الذي يطرح نفسه الآن ، هل وفر بروتوكول الـ TCP والـ UDP هذه الأمور ؟

أولا بروتوكول ال TCP : يدعم بروتوكول ال TCP كل هذه الأمور وبكفاءة عالية لاكن المشكلة الوحيدة في هذا البروتوكول هو عدم إمكانية استخدامه لعمل Multicast Conference System إذ انه من المعروف أن ال Multicasting وال Broadcasting من الأمور الخاصة ببروتوكول ال UDP ولا يدعم ال TCP أي من هذه الأمور وهو ما بينته في الفصل السابق، إذ يعتر الـ TCP بروتوكول موجه Oriented Protocol لذلك لا يمكن الاعتماد عليه في حالة حاجتنا لعمل Multicast Conference System أو في حالة البث الإذاعي Broadcasting .. إذا الحل الأخر والوحيد هو بروتوكول الـ UDP في حالة حاجتنا لهذه الأمور.

ثانيا بروتوكول ال UDP : لا يعتبر هذا البروتوكول حل جيد لعملية نقل الصوت بالكفاءة العالية إذ أنه لا يدعم عمليةDelivered on Sequence وهو ما سبب من استحالة عمل Fragmented للـ Packets المرسل ومن المعروف أن حجم Ethernet Encapsulation لا يزيد عن 1500 KB للـ Packet الواحد وهو الحجم الأقصى للـ Datagram Encapsulation الخاص بالـ Ethernet لذلك في حالة قمنا بعمل Fragmented لصوت فإننا لن نضمن وصول الصوت وفق الترتيب المرسل وهو ما يسبب مشكلة كبيرة في عملية إعادة ترتيب الـ Fragments المرسل ومن هذه النقطة قدمت الكثير من الشركات والمنظمات العالمية حلول خاصة لعملية نقل الصوت عبر ال UDP منها منظمة الـ ITU- International Telecommunications Union بتقديمها أسلوب النقل H.323 ومنظمة IETF Internet Engineering Task Force بتقديمها أسلوب النقل RTP – Real Time Transport Protocol ، حيث أضاف هذا المعيار تحسينات على بروتوكول ال UDP لعملية نقل الصوت في الزمن الحقيقي إذ يستخدم أسلوب ال Stream المستخدم في TCP لاكن تحت منصة ال UDP وقد حل هذا المعيار بعض هذه المشاكل لاكن ليس

جميعها ، إذ أن الحاجة أصبحت ملحة لوجود بروتوكول يدعم عملية النقل وفق الترتيب الصحيح Delivered on Sequence بالإضافة إلى دعم الاتصال ك Stream ودعم لل IP Multicasting وال Broadcasting ، وكان الحل بإنشاء بروتوكول أخر وهو الـ – SCTP Stream Control Transmission Protocol لاكن المشكلة أن منصة Windows لا تدعم هذا البروتوكول وقد تم دعمه بشكل كامل في نظام التشغيل Linux ، كما وعدت Microsoft بدعم هذا البروتوكول في الإصدار التالي من نظام التشغيل Windows والذي سآتي على شرحه في الجزء التالي من هذا الفصل.

ثانيا :The Concept Of Voice Communication :

تمر عملية التقاط الصوت بمجموعة من المراحل تبدأ بالتقاط الصوت من المايكروفون وتمثيل الذبذبات الصوتية ثم تحويلها إلى مجموعة من الـ Bits وذلك بعمل Sampling لذبذبات الصوتية الملتقطة وبعد هذه العملية يمكننا نقل الصوت عبر الشبكة وتمر عملية نقل الصوت عبر الشبكة بمجموعة من المراحل وهي :

1- في الـ Application Layer ، طريقة التقاط الصوت وتحويله إلى Bits وهو ما ذكرته سابقا ، و استخدام تقنيات لضغط الصوت Audio Compression Teachings وحتى يمكن إرساله عبر الإمكانيات المحدودة لشبكة الاتصال.

2- في الـ Transport Layer ، وهو من أهم الأمور التي يجب أخذها بعين الاعتبار إذ أن المفاضلة بين اختيار بروتوكول الـ UDP أو الـTCP تعتمد على مدى الحاجة التي نريدها ودقة الصوت من جهة أخرى إذ أن أفضل طريقة لنقل الصوت هي استخدام تقنيات الـStream لاكن من المعروف أن بروتوكول الـUDP لا يدعم عملية النقل ك Stream كونه لا يدعم التوصيل وفق الترتيب Delivered on Sequence إذ أننا في هذه الحالة لن نتمكن من عمل الـ Fragmentation للـ Buffer حيث لن نضمن وصول الـ Fragments وفق الترتيب الذي أرسل عليه وسوف نضطر إلى التقيد بمحدوديات الـ Ethernet للـ Packet وهي 1500 KB للـ Packet الواحد ولحل هذه المشكلة سوف نلجأ إلى تبني بعض التقنيات الجديدة والتي تعتمد على بروتوكول الـ UDP وحتى يتم نقل الصوت ك Stream ومنها أسلوب النقل H.323 والذي ذكرته سابقا ، لاكن لم تدعم الدوت نيت أي من هذه التقنيات ، لذلك عندما نريد نقل صوت من جهاز إلى آخر ك Stream لابد لنا من استخدام بروتوكول الـ TCP لاكن كما ذكرنا سابقا فإنه لا يدعم الـ IP Multicasting و الـ Broadcasting

3- في الـ Network Layer يتم عنونة الـ Packets وإذا ما قررنا اعتماد الـ UDP فإننا سوف نتمكن من عمل البث الإذاعي Broadcasting ومجموعات البث Multicasting

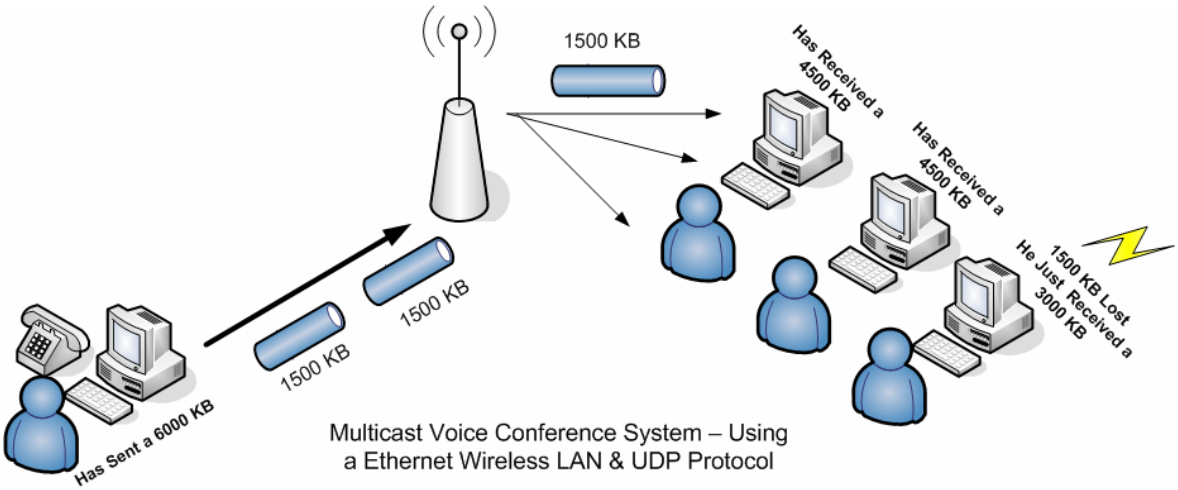
4- في الـ Data Link Layer سيتم تحديد طبيعة ولإرسال سواء باستخدام الـ Ethernet أو غيره وفي هذه الحالة سيتم الاعتماد على الـ Ethernet لاكن مشكلته كما ذكرتها سابقا بمحدودية حجم الـ Frame إذ لا تتجاوز الـ 1500 KB

5- في الـ Physical Layer طبعا المشاكل التي قد تحدث إثناء عملية النقل كثيرة جدا وقد يحدث تأخير Delay لسبب أو لآخر أو قد تضيع بعض الـ Bits إثناء الإرسال لذلك لابد من وجود بروتوكولات تدعم التصحيح لكل هذه المشاكل والتي قد تحدث إثناء عملية الإرسال.

يستقبل الطرف المقابل الـ Bits من طبقة الـ Physical Layer وتمر عبر الـ Data Link Layer ومن ثم الـ Network Layer وفي إثناء هذه المرحلة فإن مستقبل الـ Packets قد يكون هو الشخص المعني في حالة كان أسلوب البث Unicast أو قد يكون جزء من مجموعة الاستقبال Multicast أو قد يكون من ضمن الشبكة التي تم الإرسال لها ك Broadcast لذلك في حالة كونه جزء من مجموعة فإن جهة الإرسال غير معنية بالجهة التي سوف تستقبل الـ Packets وفي هذه الحالة فإنه غير معني سواء استقبلت جزء

من الـ Packets أو كلها حيث لن يتم إرسال أي Acknowledgment إلى المرسل لذلك قد تحدث الكثير من المشاكل إثناء هذه المرحلة منها ضياع جزء من الـ Packets المرسل والذي سوف يسبب وصول الصوت بشكل متقطع ، وطبعا سوف يكون الاعتماد في هذه الحالة على بروتوكولات الطبقة الأعلى وهي هنا Transport Layer فإذا كان المرسل والمستقبل يستخدم الـ TCP فإن كل هذه المشاكل سوف تحل لاكن المشكلة تكمن في كونه يستخدم الـ UDP حيث لا يوجد حل إلا بإتباع معيار مساند يضمن وصول كافة الـ Packets بترتيب الذي أرسل عليه وبدون ضياع أجزاء من الـ Packets المرسل.

الشكل التالي يوضح عملية ضياع بعض الـ Packets إثناء الإرسال باستخدام شبكة Ethernet Wireless LAN و UDP IP Multicasting مما سوف يسبب تقطيع في الصوت:



ثالثا: How to Create a Voice Chat Throw Dot Net Using Unmanaged API's Functions :

كما قلنا سابقا فإن الدوت نيت لم تدعم أي من عمليات التقاط وعرض الصوت ، لاكن لإجراء هذه العمليات لابد من استخدام مجموعة ملفات الـ DLL والتي تأتي مع نظام التشغيل ومنها ملف winmm.dll الشهير ، والخاص بالتعامل مع وسائل الـ Multimedia في نظام التشغيل ، حيث يدعم هذا الملف مجموعة من الـ Methods لالتقاط الصوت عبر المايكروفون وتخزينه في Byte Array Buffer ومن ثم عرضه مرة اخرى وهذه الـ Method هي :

waveInGetNumDevs والتي تستخدم لتحديد عدد أجهزة الإدخال والمربوطة مع الـ Sound Card ولا تأخذ أي باروميترات.

waveInAddBuffer وتستخدم لتخزين الـBits الواردة من جهاز الإدخال في Byte Array Buffer وتأخذ هذه الـMethod ثلاثة باروميترات وهي:

waveInAddBuffer(IntPtr hwi, ref WaveHdr pwh, int cbwh)
حيث يمرر للأول جهاز الإدخال والذي تم اختياره و يحدد في الثاني Reference لموقع تخزين الـ Buffer وفي الثالث يحدد حجم الـ Buffer المستلم

الـميثود waveInClose و waveInOpen لفتح وإغلاق الاتصال مع جهاز الإدخال.
الميثود waveInPrepareHeader لتجهيز وحجز الـ Buffer وتأخذ نفس الباروميترات الموجودة في waveInAddBuffer.

الميثود waveInUnprepareHeader ويتم استدعائها بعد تعبئة الـ Buffer حتى يتم إرسال الـ Buffer ومن ثم تفريغه للاستعداد لتعبئته مرة أخرى.
الميثود waveInReset لإرجاع مؤشر الـPointer الخاص بالـ Buffer إلى صفر
الميثود waveInStart و الميثود waveInStop بدأ وإغلاق عملية الإدخال من المايكروفون.

ولاستخدام هذه الميثود في الدوت نيت نقوم بتعريفها أولا باستخدام DllImport وكما يلي:

```
[DllImport(winmm.dll)]
public static extern int waveInGetNumDevs();
[DllImport(winmm.dll)]
public static extern int waveInAddBuffer(IntPtr hwi, ref WaveHdr pwh, int
cbwh);
[DllImport(winmm.dll)]
public static extern int waveInClose(IntPtr hwi);
[DllImport(winmm.dll)]
public static extern int waveInOpen(out IntPtr phwi, int uDeviceID,
WaveFormat lpFormat, WaveDelegate dwCallback, int dwInstance, int
dwFlags);
[DllImport(winmm.dll)]
public static extern int waveInPrepareHeader(IntPtr hWaveIn, ref WaveHdr
lpWaveInHdr, int uSize);
[DllImport(winmm.dll)]
public static extern int waveInUnprepareHeader(IntPtr hWaveIn, ref WaveHdr
lpWaveInHdr, int uSize);
[DllImport(winmm.dll)]
public static extern int waveInReset(IntPtr hwi);
[DllImport(winmm.dll)]
public static extern int waveInStart(IntPtr hwi);
[DllImport(winmm.dll)]
public static extern int waveInStop(IntPtr hwi);
```

وكما سوف نستخدم مجموعة الـMethods التالية لتحويل الـ Byte Array Buffer إلى صوت مرة أخرى وعرضه على جهاز الإخراج :

```
[DllImport(winmm.dll)]
public static extern int waveOutGetNumDevs();
[DllImport(winmm.dll)]
public static extern int waveOutPrepareHeader(IntPtr hWaveOut, ref WaveHdr
lpWaveOutHdr, int uSize);
[DllImport(winmm.dll)]
public static extern int waveOutUnprepareHeader(IntPtr hWaveOut, ref
WaveHdr lpWaveOutHdr, int uSize);
[DllImport(winmm.dll)]
public static extern int waveOutWrite(IntPtr hWaveOut, ref WaveHdr
lpWaveOutHdr, int uSize);
[DllImport(winmm.dll)]
public static extern int waveOutOpen(out IntPtr hWaveOut, int uDeviceID,
WaveFormat lpFormat, WaveDelegate dwCallback, int dwInstance, int
dwFlags);
```

```
[DllImport(winmm.dll)]
public static extern int waveOutReset(IntPtr hWaveOut);
[DllImport(mmdll)]
public static extern int waveOutClose(IntPtr hWaveOut);
[DllImport(mmdll)]
public static extern int waveOutPause(IntPtr hWaveOut);
[DllImport(mmdll)]
public static extern int waveOutRestart(IntPtr hWaveOut);
[DllImport(mmdll)]
public static extern int waveOutGetPosition(IntPtr hWaveOut, out int lpInfo,
int uSize);
[DllImport(mmdll)]
public static extern int waveOutSetVolume(IntPtr hWaveOut, int dwVolume);
[DllImport(mmdll)]
public static extern int waveOutGetVolume(IntPtr hWaveOut, out int
dwVolume);
```

وحتى نتمكن من عرض محتويات الـ Buffer نستخدم التعريف التالي:

```
using System;
using System.Runtime.InteropServices;
using System.Resources;
using System.IO;

  public class Winmm
        {
                public const UInt32 SND_ASYNC = 1;
                public const UInt32 SND_MEMORY = 4;

                [DllImport("Winmm.dll")]
                public static extern bool PlaySound(byte[] data, IntPtr hMod,
UInt32 dwFlags);
                public Winmm()
                {}
public static void PlayWavResource(byte[] buffer)
{
        PlaySound(buffer, IntPtr.Zero,  SND_ASYNC |  SND_MEMORY);
}
}
```

حيث نمرر للـ PlaySound Method الـ Byte Buffer والمستلم من Method الاستقبال الخاصة بالـ Socket وكما يلي:

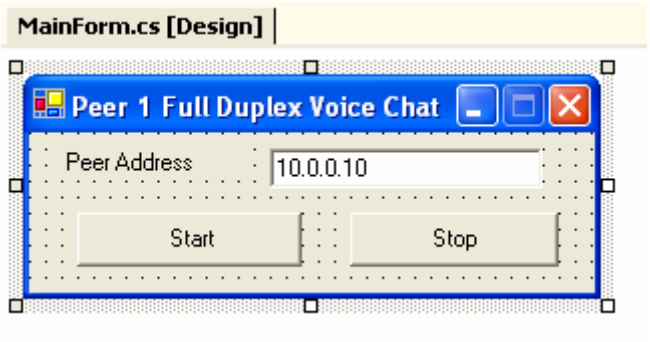
```
void Voice_Receiver()
        {
UdpClient sock = new UdpClient(5020);
sock.JoinMulticastGroup(IPAddress.Parse(multicast_IP.Text));
IPEndPoint iep = new IPEndPoint(IPAddress.Any, 0);
byte[] voice_Come = sock.Receive(ref iep);
Winmm.PlayWavResource(voice_Come);
sock.Close();
        }
```

البدء بإنشاء برنامج المحادثة الصوتية Voice Chat System :

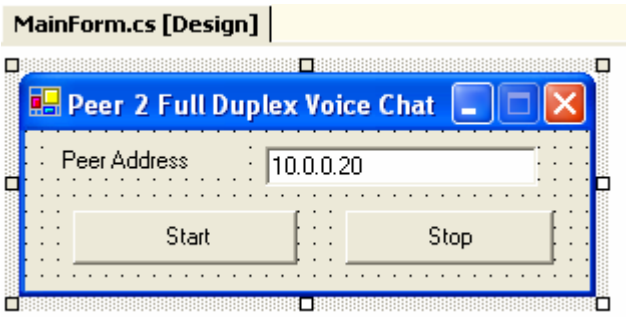
سوف نجزئ عملية التقاط الصوت وتخزينه في الـ Buffer ثم عرضه مرة أخرى في مجموعة من الـClasses وهو تقسيم تم استخدامه في الكثير من البرمجيات الخاصة ب Microsoft ومنها برنامج Windows Sound Recorder وسوف نجمع هذه الـ Classes في ملف واحد نسميه Voice Library وسوف أرفق محتويات هذه الـ Classes في ملحقات هذا الفصل ، وهذه الـ Classes هي:

WaveIn Class وسوف نستخدمه لوضع كافة الـ Methods الخاصة بالتقاط الصوت وتخزينه في Byte Array
WaveOut Clasas وسوف نستخدمه لعرض الصوت الأتي من الـ Buffer ثم عرضه
WaveStream Class والذي سوف نستخدمه لتحويل الصوت إلى Stream حيث يسهل إرساله عبر الشبكة ويشبه عمله عمل MemoryStream المستخدمة في الدوت نيت
الميثود FifoStream لتنظيم الـ Stream بحيث يتم عرض الداخل أولا خارج أولا
الميثود WaveNative ويتم فيها وضع كافة التعريفات للـ Methods الخاصة بالملف winmm.dl والتي شرحناها سابقاً.

سوف نستخدم في هذا المثال بروتوكول الـ UDP لعملية النقل ومعتمدا على أسلوب البث Full Diplex Unicast Voice Chat System وللبدء سوف يكون الشكل العام لبرنامج الاتصال كما يلي :



وهنا صورة برنامج الطرف المقابل :



وسوف نقوم بكبسلة الـ Classes السابقة في ملف Voice.dll وسوف نضعه في الـ References الخاصة بالبرنامج وحتى نستطيع استخدام هذا الملف في جميع البرامج التي سوف تستخدم عملية الاتصال الصوتي بعد هذه العملية سنقوم الملف باستخدام الـ Using وكما يلي:

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
using Voice;
```

ثم نقوم بتعريف الـSocket والـThread والذي سوف نستخدمه في البرنامج ويفضل وضع هذه التعريفات في بداية البرنامج أي بعد تعريف الـ Class الرئيسي والهدف من هذه العملية هي القدرة على إغلاق الـ Socket والـ Thread عند إطفاء البرنامج وحتى لا تبقى في الذاكرة عند إغلاق برنامج الاتصال ، ويتم ذلك كما يلي:

```
public class Form1 : System.Windows.Forms.Form
{
private Socket socket;
private Thread thread;
```

وسوف نعرف Object من الـClasses السابقة ونعرف الـBuffer الذي سيتم تسجيل الصوت المراد إرساله والـBuffer الذي سيتم عرض الصوت المستلم من الـSocket

```
private WaveOutPlayer m_Player;
private WaveInRecorder m_Recorder;
private FifoStream m_Fifo = new FifoStream();
private byte[] m_PlayBuffer;
private byte[] m_RecBuffer;
```

في الـ Constructure الخاص بالبرنامج أو في الـ Form Load Event قم بكتابة التعريف الخاص بالـ Socket والـ Thread

```
public Form1()
{
InitializeComponent();
socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
ProtocolType.Udp);
thread = new Thread(new ThreadStart(Voice_In));
}
```

سوف نضع في الـVoice_In Method الكود الخاص بعملية استقبال الصوت من الـSocket وكما يلي:

```
private void Voice_In()
{
        byte[] br;
        socket.Bind(new IPEndPoint(IPAddress.Any, 5020));

while (true)
        {
        br = new byte[16384];
        socket.Receive(br);
        m_Fifo.Write(br, 0, br.Length);
        }
}
```

حيث يتم استقبال الصوت من الشبكة باستخدام الـReceive Method ثم نمرر الصوت المستقبل إلى الـ m_Fifo.Write Method وحتى يتم تنفيذه وتحويله إلى صوت مرة أخرى.

أما الـMethod التي تقوم بتسجيل الصوت وإرساله إلى الجهاز الأخر فهي:

```
private void Voice_Out(IntPtr data, int size)
{
```

```
//for Recorder
if (m_RecBuffer == null || m_RecBuffer.Length < size)
m_RecBuffer = new byte[size];
System.Runtime.InteropServices.Marshal.Copy(data, m_RecBuffer, 0, size);
//Microphone ==> data ==> m_RecBuffer ==> m_Fifo
socket.SendTo(m_RecBuffer, new
IPEndPoint(IPAddress.Parse(Peer_IP.Text),5030));
}
```

لاحظ أنه في حالة إذا ما أردنا عمل برنامج Full Duplex بحيث يرسل ويستقبل في نفس الوقت فإننا بحاجة إلى تعريف Tow Ports واحد للإرسال وأخرى للاستقبال وفي الطرف الأخر تكون Port الإرسال لديك هي Port الاستقبال لديه والعكس صحيح ...

في زر البدء يتم تنفيذ الميثود التالية:

```
private void Start()
{
        Stop();
try
{
WaveFormat fmt = new WaveFormat(44100, 16, 2);
m_Player = new WaveOutPlayer(-1, fmt, 16384, 3, new
BufferFillEventHandler(Filler));
m_Recorder = new WaveInRecorder(-1, fmt, 16384, 3, new
BufferDoneEventHandler(Voice_Out));
}
catch
{
Stop();
throw;
}
}
```

أما في زر الإيقاف فيتم تنفيذ الميثود التالية:

```
private void Stop()
{
if (m_Player != null)
        try
                {
                m_Player.Dispose();
                }
        finally
        {
        m_Player = null;
        }
if (m_Recorder != null)
        try
        {
        m_Recorder.Dispose();
        }
        finally
        {
        m_Recorder = null;
```

```
        }
        m_Fifo.Flush(); // clear all pending data
}
```

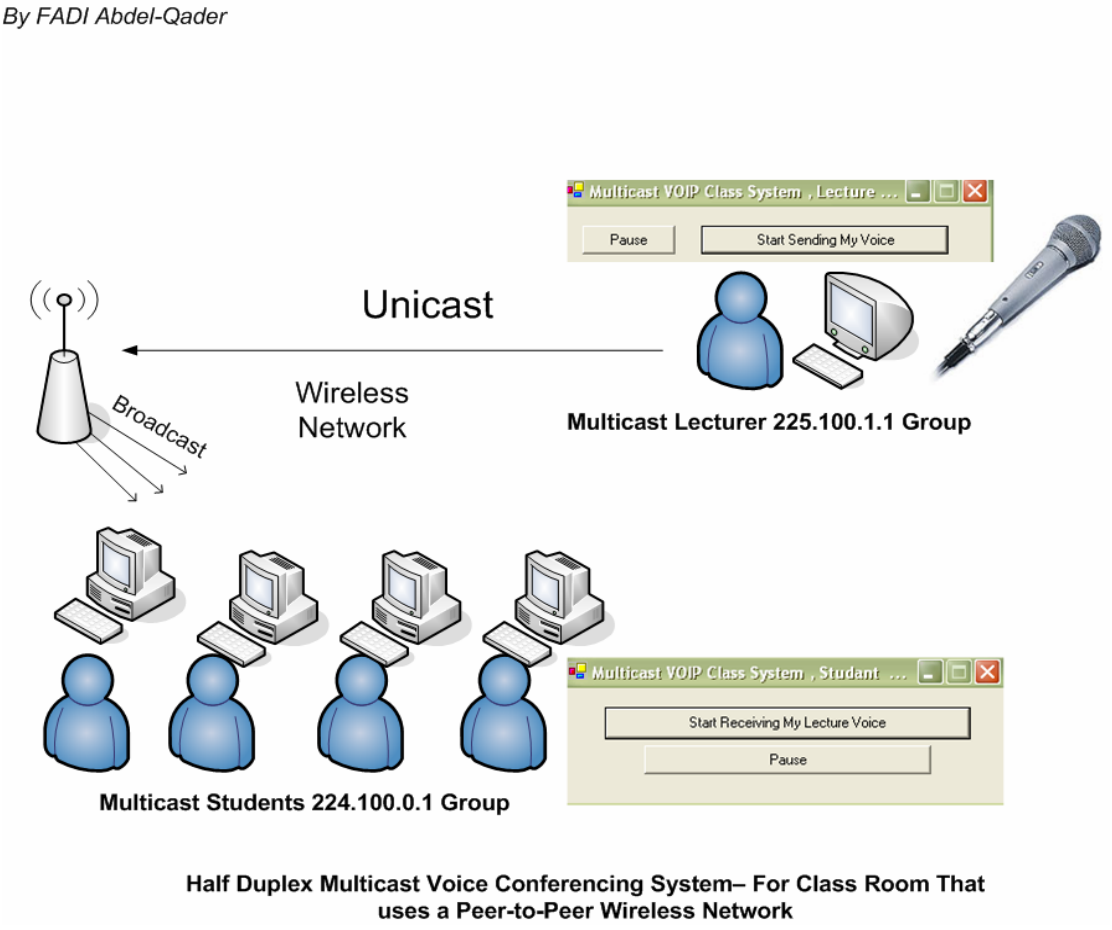
الميثود التي تقوم بعرض الـ Voice Buffer والمستلم من Socket على السماعة:

```
private void Filler(IntPtr data, int size)
{
if (m_PlayBuffer == null || m_PlayBuffer.Length < size)
m_PlayBuffer = new byte[size];
if (m_Fifo.Length >= size)
m_Fifo.Read(m_PlayBuffer, 0, size);
else
for (int i = 0; i < m_PlayBuffer.Length; i++)
m_PlayBuffer[i] = 0;
System.Runtime.InteropServices.Marshal.Copy(m_PlayBuffer, 0, data, size);
// m_Fifo ==> m_PlayBuffer==> data ==> Speakers
}
```

رابعا: Testing TCP,UDP and Thinking in SCTP to Transfer Voice Throw Networks :

لإنشاء برنامج Multicast Half Duplex Voice Chat System نقوم بإضافة التعريفات التالية والخاصة بالـ Multicasting والتي شرحناها في الفصل السابق ، وسوف نعتمد على وجود برنامجين واحد للمحاضر يتم من خلاله تسجيل الصوت وآخر لطالب حيث يستمع فيه لمحاضرة المعلم وكما في الشكل التالي:

By FADI Abdel-Qader



Half Duplex Multicast Voice Conferencing System– For Class Room That uses a Peer-to-Peer Wireless Network

في برنامج الإرسال (برنامج المعلم) لا يختلف الكود بشيء فقط عند الإرسال يتم ذلك باستخدام الـ IP Multicasting وكما يلي:

```
 private void Voice_Out(IntPtr data, int size)
{
//for Recorder
if (m_RecBuffer == null || m_RecBuffer.Length < size)
m_RecBuffer = new byte[size];
System.Runtime.InteropServices.Marshal.Copy(data, m_RecBuffer, 0, size);
//Microphone ==> data ==> m_RecBuffer ==> m_Fifo
socket.SendTo(m_RecBuffer, new IPEndPoint(IPAddress.Parse("224.0.1.7"),
5020));
}
```

في الطرف المستقبل نقوم بالانضمام إلى الـIP Multicast Group ومن ثم الاستقبال من خلاله وكما يلي:

```
private void Voice_In()
{
UdpClient sock = new UdpClient(5000);
sock.JoinMulticastGroup(IPAddress.Parse("224.0.1.7"));
IPEndPoint iep = new IPEndPoint(IPAddress.Any,0);


while (true)
        {
        m_Fifo.Write(sock.Receive(ref iep), 0,sock.Receive(ref iep).Length);
        }
}
```

الآن نفذ البرنامج ...
لاحظ أن الصوت قد يتقطع أحيانا وبتأكيد السبب واضح وهو أنه في حالة استخدام بروتوكول الـ UDP والـ Multicasting فإن عملية الإرسال ستكون عشوائية وهذا قد يسبب ضياع واحد أو أكثر من الـPackets المرسلة عبر الشبكة كل فترة وبما أن بروتوكول الـ UDP لا يدعم أي من عمليات التحقق من الوصول وعمليات التوصيل على الترتيب فإن حدوث واحد أو أكثر من هذه المشاكل أمر محتمل ...

دعنا الآن نجرب عملية الإرسال باستخدام بروتوكول TCP ، لاحظ أن هذا البروتوكول هو بروتوكول موجه Oriented Protocol كما يدعم جميع عمليات التحقق من الوصول بالإضافة إلى كونه يدعم الاتصال بين الطرفين باستخدام أسلوب الـ Stream وهو ما يميز هذا البروتوكول عن غيره إذ أننا في حالة استخدامه لن نضطر إلى الأخذ بحجم البيانات المرسلة حيث يتم عمل Fragmentation لها بكل سهولة بالإضافة إلى سهولة تجميعها مرة أخرى وبدون الخوف من مشاكل الـ Delivered Out on Sequence ...
كل ما علينا تغييره هو تعريف الـ Socket السابق وجعله كما يلي :

```
socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream ,
ProtocolType.Tcp);
```

أما في عملية الإرسال فيمكننا استخدام الـNetwork Stream Class وكما يلي:

```
 private void Voice_Out(IntPtr data, int size)
{
//for Recorder
```

```
if (m_RecBuffer == null || m_RecBuffer.Length < size)
m_RecBuffer = new byte[size];
System.Runtime.InteropServices.Marshal.Copy(data, m_RecBuffer, 0, size);
//Microphone ==> data ==> m_RecBuffer ==> m_Fifo
sock.Connect(new IPEndPoint(IPAddress.Parse("10.0.0.10"),5020));
NetworkStream ns = new NetworkStream (socket);
ns.Write (m_RecBuffer,0,m_RecBuffer.Length);
}
```

في الطرف المستقبل نقوم باستخدام الـ Network Stream مرة أخرى لاكن للاستقبال:

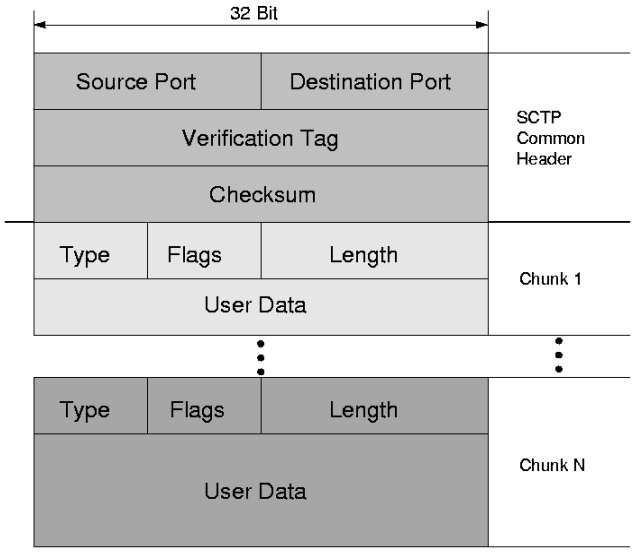
```
private void Voice_In()
{
        Socket sock = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream , ProtocolType.Tcp);
        sock.Bind(new IPEndPoint(IPAddress.Parse("10.0.0.10"),5020));
        NetworkStream ns = new NetworkStream (sock);
        while (true)
        {
        byte[] buffer = new byte [16384];
        ns.Read (buffer,0,16384);
        m_Fifo.Write(buffer, 0, buffer.Length); }
}
```

لاحظ أن دقة الصوت أصبحت ممتازة كما أنه لا يوجد أي تقطيع في الصوت لاكن المشكلة تكمن في أننا لن نستطيع الاستفادة من هذه الإمكانيات الرائعة في الـMulticasting أو الـBroadcasting ...

والحل الوحيد هو إما استخدام المعايير السابقة مع بروتوكول الـUDP أو استخدام بروتوكول الـTCP أو الانتظار لحين الانتهاء من مشروع Long Horn حتى تدعم Microsoft بروتوكول الـSCTP ...

السؤال الذي يطرح نفسه الآن ما هو الجديد بهذا البروتوكول هل حل المشكلة؟؟

الجواب بكل بساطة نعم قد حل المشكلة حيث أن معمارية هذا البروتوكول الذي استفاد من ميزات الـTCP والدعم المقدم من قبل الـUDP لعمليات الـMulticasting إذ أصبح لدينا الآن منصة قوية يعتمد عليها في عمل الـFragmentation وإعادة ترتيبها بكل سهولة بالإضافة إلى دعمه عملية Delivered on Sequence وهذا واضح من بنية الـHeader الخاصة بهذا البروتوكول انظر إلى الشكل التالي:



خامسا: How to Create a Voice Conference System Using Microsoft Direct Play 9 :

دعمت Microsoft تقنيات رائعة جدا للنقل الصوت في الإصدار الخاص بDirect Play9 ويسمى أيضا DirectPlay Transport Protocol وهو جزء من مجموعة الـDirectX وكان الهدف من إطلاقها وجود معيار موحد لمبرمجين الألعاب فيما يخص الـNetwork Games، وتعتمد هذه المكتبة على مجموعة من المعايير الخاصة بتشبيك حيث كان الهدف منها هو جعل عملية الاتصال ممكنة تحت جميع البيئات المختلفة و سواء كان البروتوكول المستخدم هو الـTCP/IP أو IPX الخاص بNovel فإن عملية الاتصال ممكنة وبدون أي اختلافات من النواحي البرمجية، ومن أهم ميزات الـ DirectPlay Transport Protocol:
- Reliable delivery of messages حيث يدعم عملية التحقق التوصيل للجهة المعنية
- Sequential a delivery of messages حيث يدعم التوصيل وفق الترتيب الصحيح
- Send prioritization حيث يدعم عملية وضع أولويات للإرسال بناء على الأهمية
- Streaming Session حيث تدعم عملية النقل كStream Data ،

قدمت Microsoft هذه الحلول كبدائل لدعم بروتوكول الـSCTP المنتظر ، وقد حلت جميع المشكلات التي كانت تواجه المبرمجين لنقل الصوت عبر بروتوكول الـTCP أو الـUDP وحل محله أسلوب آخر لربط ضمن مستوى طبقة الـ Transport Layer ، وتحتوي الـ Direct Play على مجموعة ضخمة من الـClasses ومن أهمها :

أولا : الـ Connect Classes والخاصة بعملية الربط:
الـ Address , Guid , Peer Classes حيث تستخدم عند إنشاء الاتصال مع الطرف الأخر ، وتستخدم DirectPlay طريقة لتمييز البرنامج عن الأخر بتوليد Hash Code خاص بكل برنامج ويتم ذلك باستخدام الـ Guid Class ويتم تمرير الكود المولد وعنوان الجهاز المقابل إلى الـ Peer Class وهذه العملية شبيهة بشكل كبير لعملية الربط باستخدام الـ Socket في بروتوكول TCP/IP ، ويتم استخدامها كما يلي كمثال:

```
using Microsoft.DirectX.DirectPlay;
.
.
.
Address hostAddress = new Address();
hostAddress.ServiceProvider = Address.ServiceProviderTcpIp;
                        // Select TCP/IP service provider

ApplicationDescription dpApp = new ApplicationDescription();
appGuid = Guid.NewGuid();          // Create a GUID for the application
dpApp.GuidApplication = appGuid;   // Set the application GUID
dpApp.SessionName = "My Session";  // Optional Session Name

myPeer.Host(dpApp, hostAddress);   // Begin hosting
```

طبعا يجب الاختيار طبيعة البروتوكول المستخدم سواء كان TCP/IP أو IPX وحتى نستطيع وضع العنوان المقابل أو الـ IP Address ويتم ذلك كما يلي:

```
using Microsoft.DirectX.DirectPlay;
ServiceProviderInfo[] mySPInfo;
Peer myPeer;
System.Windows.Forms.ListBox listBox1;
ApplicationDescription myAppDesc;
.
```

```
.
myPeer = new Peer();
hostAddress = new Address();
peerAddress = new Address();

// Set the service provider to TCP/IP
peerAddress.ServiceProvider = Address.ServiceProviderTcpIp;
hostAddress.ServiceProvider = Address.ServiceProviderTcpIp;

// Attach FindHostResponseEventHandler to receive
FindHostResponseMessages
myPeer.FindHostResponse += new
FindHostResponseEventHandler(myEnumeratedHosts);

// Call FindHosts to start the enumeration
myPeer.FindHosts(myAppDesc, hostAddress, peerAddress, null, 10, 0, 0,
        FindHostsFlags.OkToQueryForAddressing);
```

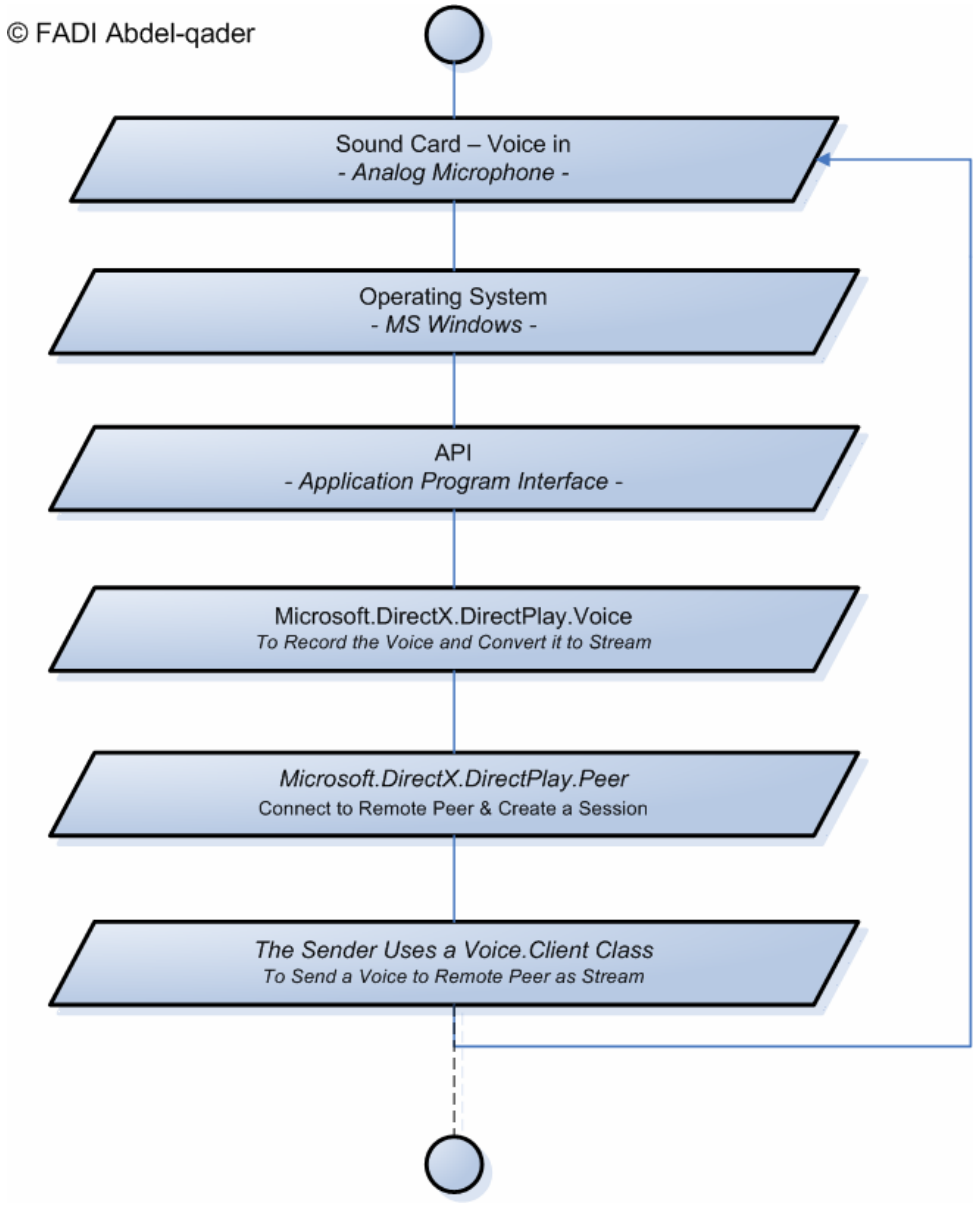
حيث تم تعريف نوع البروتوكول المستخدم وهو TCP/IP ويتم البحث عن الطرف الأخر في الشبكة باستخدام الـ FindHosts Method والموجودة ضمن الـPeer Class ، وتتم عملية الربط مباشرة باستخدام الـ Connect Method والموجودة ضمن الـ Peer Class وكما يلي:

```
using Microsoft.DirectX.DirectPlay;
// Structure for FindHostResponseMessages
public struct HostInfo
{
        public ApplicationDescription appdesc;
        public Address deviceAddress;
        public Address senderAddress;

}
.
.
.
Peer myPeer = new Peer();
HostInfo hostinfo = new HostInfo();

// The FindHostResponseEventHandler
public void myEnumeratedHosts(object o, FindHostResponseEventArgs args)
{
        hostinfo.appdesc = args.Message.ApplicationDescription;
        hostinfo.deviceAddress = args.Message.AddressDevice;
        hostinfo.senderAddress = args.Message.AddressSender;
}

// Attach the ConnectCompleteEventHandler to receive
ConnectCompleteMessages
myPeer.ConnectComplete += new
ConnectCompleteEventHandler(OnConnectComplete);
```

```
// Call connect passing the Host information returned in the
FindHostResponse event
myPeer.Connect(hostinfo.appdesc, hostinfo.deviceAddress,
hostinfo.senderAddress, null, ConnectFlags.OkToQueryForAddressing);
```

ثانيا: Microsoft.DirectX.DirectPlay.Voice والخاصة بكل عمليات تسجيل ونقل وعرض الصوت:

تمر عملية تسجيل ونقل وعرض الصوت بمجموعة من المراحل ونلخصها بالشكل التالي:

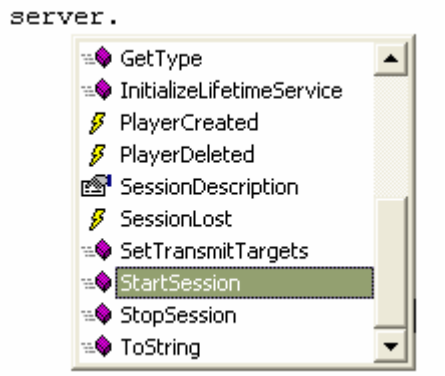


وسوف نقسمها إلى مجموعة من الـ Classes حسب الوظيفة لكل منها ،

1- الـ Classes الخاصة بطرف الـ Server لإنشاء وإدارة الـ Sessions :
Server Class ويستخدم كما يلي كمثال:

```
Server server = new Voice.Server(peerObject);
```

حيث نسند له ال Peer Object والذي تم اشتقاقه من الـ Peer Class ، ويحتوي الـ Server Object على الـ Methods الخاصة بعملية بدأ وإنهاء الجلسة بالإضافة إلى مجموعة من العمليات الأخرى:



ولبدأ الجلسة يجب أولا إسناد خصائص الجلسة إلى الـStartSession Method حيث يتم تعريفها من خلال الـ SessionDescription و كما يلي:

```
//set up session description for the voice server
Voice.SessionDescription sessionDesc = new Voice.SessionDescription();
sessionDesc.BufferAggressiveness = Voice.BufferAggressiveness.Default;
sessionDesc.BufferQuality = Voice.BufferQuality.Default;
sessionDesc.Flags = 0;
sessionDesc.SessionType = type;
sessionDesc.GuidCompressionType = compressionType;
```

ولإنشاء Voice Session نقوم بإنشاء Method نمرر لها ال Peer Object والـ Voice Session Type ونوع الضغط المستخدم Compression Type ويتم ذلك كما يلي:

```
protected void CreateVoiceSession(Peer dpp, Voice.SessionType type, Guid
compressionType)
   {
try
    {
//set up session description for the voice server
Voice.SessionDescription sessionDesc = new Voice.SessionDescription();
sessionDesc.BufferAggressiveness = Voice.BufferAggressiveness.Default;
sessionDesc.BufferQuality = Voice.BufferQuality.Default;
sessionDesc.Flags = 0;
sessionDesc.SessionType = type;
sessionDesc.GuidCompressionType = compressionType;

//start the session
        try
          {
             server.StartSession(sessionDesc);
             mIsHost = true;
             mInSession = true;
          }
          catch(DirectXException dxe)
          {
             throw dxe;
          }
```

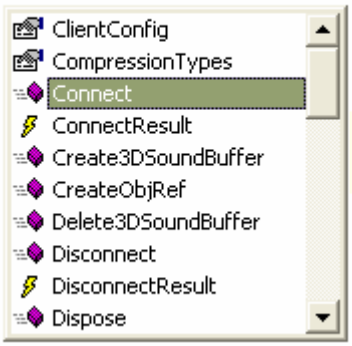
```
        }
        catch(Exception e)
        {
            throw e;
        }
    }
```

ويتم استدعائها كما يلي:

```
CreateVoiceSession(host, Voice.SessionType.Peer,
mConfigForm.CompressionGuid);
```

2- الـ Classes الخاصة بطرف الـ Client للاتصال مع الـ Sessions التي أنشئها الـServer:
الـ Client Class ومن أهم الـMethods الموجودة في الـ Client Class :

```
//Connect to voice session
client.Connect(soundConfig,
```



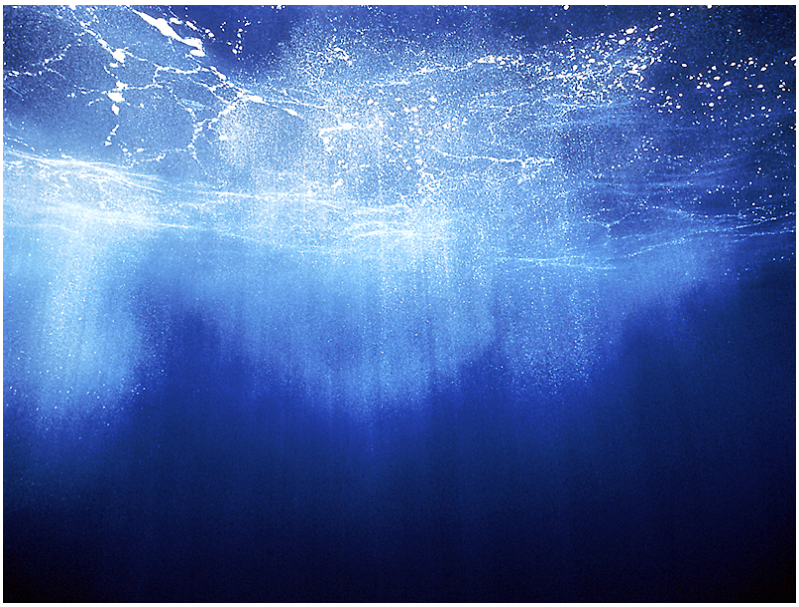
الميثود Connect وتستخدم لربط مع الـVoice Session حيث يرسل الـ Server رقم الـSession للـ Client وعندها يتمكن الـ Client من الدخول إلى الجلسة ، ويتم ذلك كما يلي:

```
    protected void ConnectToVoiceSession(Peer dpp, Form wnd)
    {
        try
        {
Voice.SoundDeviceConfig soundConfig = new Voice.SoundDeviceConfig();


//Set sound config to defaults
soundConfig.GuidPlaybackDevice  =
DSoundHelper.DefaultVoicePlaybackDevice;
soundConfig.GuidCaptureDevice = DSoundHelper.DefaultVoiceCaptureDevice;
soundConfig.Window = wnd;

//TODO: add error message for specific failures?

//Connect to voice session
client.Connect(soundConfig, mClientConfig, Voice.VoiceFlags.Sync);

//set state
mInSession = true;

//set transmit targets to all players
```

```
int[] xmitTargets = new int[1];
xmitTargets[0] = (int) PlayerID.AllPlayers;
client.TransmitTargets = xmitTargets;

//get sound device config to check for half-duplex
soundConfig = client.SoundDeviceConfig;
mHalfDuplex =  ((soundConfig.Flags &  Voice.SoundConfigFlags.HalfDuplex)
!= 0);

}
catch(Exception e)
{throw e;}
   }
}
```

لاحظ أن المشاكل في البروتوكولين الـ TCP والـUDP قد تم حلها في الـ DirectPlay لاكن وكما هو معروف فإن الهدف من إنشاء الـ Direct Play لم يكن سوى لدعم برمجة الألعاب ومع المرونة الكبيرة التي تقدمها الـ Direct Play في عملية الاتصال الصوتي إلا أنها تفتقر لميزات الـ Voice Over IP والتي يقدمها بروتوكول الـ SCTP الخاص بالـ Linux...

وهكذا بينا ملخص عن أهم الطرق لاتصال الصوتي عبر الشبكة وطرق برمجة الـ Voice Chat تحت منصة الدوت نيت واهم ميزات وعيوب بروتوكولات الـTransport Layer ومدى إمكانياتها لنقل الصوت عبر الشبكة وأخيرا شرح لأهم الـ Classes والمستخدمة في الاتصال الصوتي باستخدام الـ DirectPlay Transport Protocol.



الفصل الحادي عشر من كتاب أحترف برمجة الشبكات وبروتوكول الـ TCP/IP الإصدار الثاني ، يمكنك متابعة أخر أخبار الإصدار الثاني من الكتاب الإلكتروني المجاني على الرابط التالي:

http://www.enashir.com/blogs/fadi