برمجة بروتوكول النقل في الزمن الحقيقي RTP في أنظمة الـ VOIP Multicasting Conferencing Systems

المقدمة:

سأقدم في هذه المقال تحليل لأهم التحديات التي تواجه مبرمجي أنظمة المؤتمرات وذلك من خلال دراسة العوامل المؤثرة على جودة نقل الصوت والصورة عبر شبكة الإنترنت وكيفية التغلب على هذه المشكلات باستخدام بروتوكول النقل في الزمن الحقيقي RTP Real Time Protocol وشرح كيفية التعامل معه برمجيا.

يعتبر استخدام برمجيات الدردشة Chatting وأيضا الـConferencing من أكثر الأمور التي تستخدم بشكل يومي ومع دخول هذه المجالات في أنظمة التعلم الإلكتروني تتزايد الحاجة لنقل كميات اكبر من البيانات بزمن الحقيقي Real Time Transportation لعدد اكبر من المتصلين و يعتبر عامل الزمن Timestamp من أهم التحديات والتي تواجه مبرمجي هذه الأنظمة ويظهر هذا التحدي جليا عند التعامل مع شبكة الإنترنت لنقل صوت المحاضر إلى عدد كبير من المتصلين فتأخر وصول جزء من البيانات يعني تقطع في الصوت ووصول جزء قبل الأخر يعني خروج صوت غير مفهوم ومشاكل أخرى قد تحدث في المزامنة بين الصوت والصورة وهذا الأمر غير مقبول في أنظمة الـConferencing التي تعتمد على الزمن الحقيقي والجودة العالية المفترضة منها وخاصة في شبكات الـMulicasting.

ويمكننا أن نلخص هذه التحديات بثلاثة عوامل رئيسية:

أولا حجم البيانات المرسلة وعلاقته بحجم الـBuffer عند المرسل بناء على سرعة الشبكة:

يعرف الـBuffer من طرف المرسل بأنه المساحة التخزينية التي يتم حجزها بذاكرة وبمقدار محدد لتخزين البيانات بشكل مؤقت قبل إرسالها و من الطبيعي في برمجيات الـConferencing أنه كلما زادت جودة وسرعة الشبكة قلت حاجتنا للـBuffer من طرف المرسل إذ يزيد حجم الـBuffer نسبة التأخر في عملية نقل الصوت ولكنه يضمن وصول أكثر أمانا للبيانات فبزيادة حجم الـBuffer يكون هنالك أمل أكبر لإعادة إرسال البيانات التي يتم فقدانها إثناء النقل. والتحكم في حجم الـBuffer يعتمد

أيضا على طبيعة البرنامج وكمثال فإن برمجيات البث الإذاعي أقل حاجة لزمن الحقيقي من البرمجيات التي يتم فيها التفاعل بين الأطراف وبذلك يمكن أن نزيد فيها حجم الـBuffer لتخزين مدة محددة من العرض قبل عملية الإرسال ومن الأمثلة على ذلك برنامج Microsoft Media Encoder وهو برنامج يأتي مع المجموعة Microsoft Expression Studio والذي من ضمن خواصه التعامل مع والـWindows Media Server لعملية بث الصوت والصورة سواء بشكل مباشر Live أو من ملف MM مسجل.

ثانيا (Voice Jitter (Loss & Delay:

يعتبر هذا التحدي من واحد من أكبر التحديات التي تواجه مبرمجي أنظمة المؤتمرات وذلك لسبب هام ومعروف وهو أنه كلما زاد حجم البيانات المحملة على الـ RTP Packet زاد الاحتياج إلى قناة نقل ذات جودة أعلى والسبب اختلاف حجم الـFrame الأقصى Max Size of frame والتي تستطيع قناة النقل أن تحمله وكمثال فإن الحجم الأقصى للـFrame في شبكة الـEthernet هو 1500 Bytes وهذا يعني أنه في حالة نقل البيانات على شبكات أبطأ عندها يجب تقليل حجم الـFrame وكلما قل حجم الـFrame زاد عدد تلك الـFrames وبالتالي فرصة اكبر لضياع Frames إثناء النقل كما سيحتاج المستقبل فترة أطول و Buffer أكبر لإعادة ترتيب تلك الـFrames إذ يتراوح حجم الـHeader الخاص بالـIPv4 من ٢٠ إلى ٦٠ Bytes بناءا على الـOptions المستخدم فإذا لم يتم استخدام الـOptions عندها سيكون حجم الـHeader ثابت وهو ٢٠ Bytes ويحتاج الـUDP Header إلى 8Bytes و الـRTP Header إلى 12 Bytes وفي هذه الحالة فإن صافي حجم الـ RTP Frame الواحد بدون البيانات هو 40 Bytes.

Empty RTP Frame Size = 20 Bytes for IPv4 Header+ 8 Bytes for UDP Header + 12 Bytes For RTP Header = 40Bytes.

وكمثال إذا كنت تريد نقل 1024 K Bytes من البيانات على شبكة الـEthernet فإنك ستحتاج إلى تقسيم البيانات على الحجم ألأعظمي لقناة النقل هذه وهو 1500 Bytes ولحساب ذلك:

1500 Bytes – 40 Bytes for each frame = 1460 Bytes

1024 KB to Bytes = 1024^2

(1024^2) / 1460 = 718 Frames

وبذلك نحتاج إلى تقسيم تلك البيانات إلى 718 Frames مما يعني زيادة حجم البيانات بمقدار (718 X 40) ٢٨٧٢٠ Bytes وتعتبر هذه الزيادة حجم زائد يتحملها في النهاية المستقبل وبالتالي زيادة في Delay لإجراء عمليات الترتيب والمعالجة لتلك البيانات قبل عرضها.

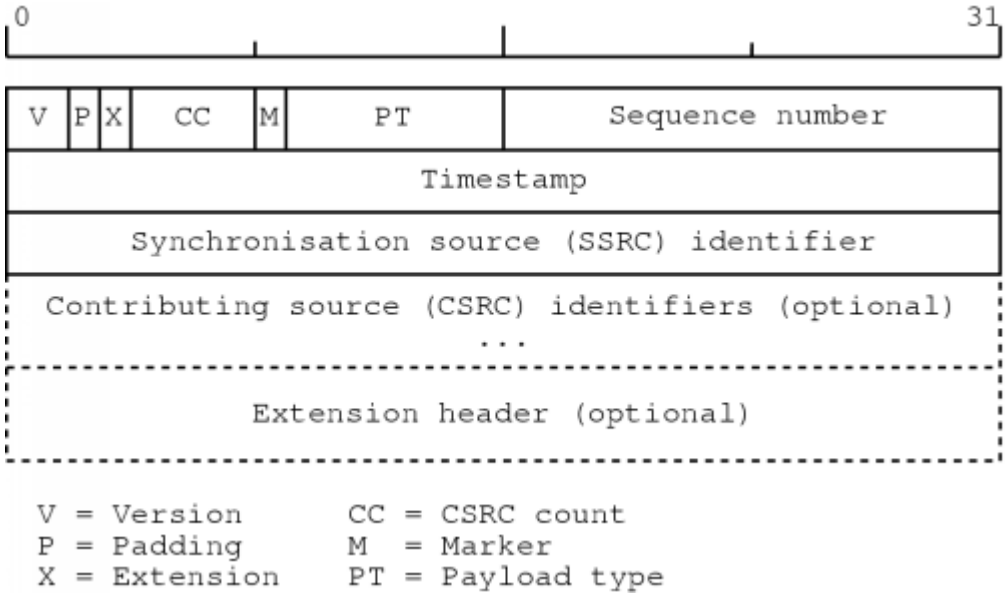
ثالثا وصول البيانات بشكل غير مرتب Out-Of- Sequence وكيفية التغلب على هذه المشكلة:

تحدثت في كتاب (احترف برمجة الشبكات والنظم الموزعة) عن عملية نقل البيانات من خلال بروتوكولات النقل المعروفة وهي الـ TCP والـ UDP وقد بينا عيوب كل منها في عملية النقل فمن المعروف أن الـTCP بروتوكول جيد لنقل البيانات الكبيرة الحجم والتي لا يعتبر فيها عامل الزمن الحقيقي لنقل أمر مهم ومن عيوبه أنه لا يدعم عمليات الـMulticasting والـBroadcasting لكن من أهم خواصه التأكد من نقل البيانات بالشكل الصحيح وبالترتيب السليم وبهذا فإن بروتوكول الـUDP سريع في عملية النقل للبيانات التي لا يزيد حجمها عن الحجم الأقصى للـFrame والتي تستطيع قناة النقل تحمله وبالتالي فإن نقل البيانات التي يزيد حجمها عن الحجم الأقصى للـFrame لا يمكن نقلها باستخدام الـUDP إلا إذا تم تقسيمها على مجموعة من الـFrames ومن المعروف أن الـUDP لا يدعم عملية ترتيب الـFrames وبالتالي فإن وصول Frame قبل الآخر قد يسبب فشل عملية النقل بأكملها ومن هنا دعم بروتوكول الـUDP ببروتوكول الـRTP لكي يتمكن المستقبل من إعادة ترتيب الـFrames بعد نقلها كما ويدعم الـRTP عملية محاولة لتصليح الـFrames التي قد

تصل مشوهة وذلك من خلال FEC – Frame Error Correction وبالتالي محاولة الإصلاح بدلا من إعادة طلب الإرسال.

مكونات الـRTP Real-Time Transport Protocol Header:

يتكون الـRTP Header من 12 Bytes أساسية Fixed مقسمة كما هو واضح في الشكل التالي:



١- **Version**: ويتكون من 2 bits ويوضع فيه الإصدار الـRTP المستخدم وحاليا نحن نعمل على الإصدار الثاني.

٢- **Padding**: ويتكون من 1 bit وهو Flag يبين إذا كنا سنضيف معلومات إضافية على الـRTP Header وكمثال يمكن أن يستخدم هذا الـPadding في حالة كنا نريد إرسال الـRTP Payload بشكل مشفر وبهذه الحالة يضاف الـPadding والذي سيحتوي على معلومات حول موقع الـPayload المشفر في أسفل الـRTP Header.

٣- **Extension**: ويتكون من 1 bit وهو Flag يبين إذا كان الـ RTP Header يحتوي على Extensions في نهاية الـRTP Header أم لا والتي قد تحتوي على معلومات خاصة بالـApplication الذي يستخدم الـRTP لنقل البيانات.

٤- **CSRC Count** : ويتكون من 4 bits وفيه يبين عدد الـ Content Source Identifiers الملحقة مع الـRTP Header ويستخدم هذا الـField في حالة كنا نريد دمج أكثر من RTP Stream في RTP Stream واحد فمن المعروف أنه لا يمكن تحميل أكثر من Payload Type واحد على الـRTP Stream وبالتالي في حالة كنا نريد إرسال الصوت والصورة فلا بد من استخدام اثنين من الـRTP Stream وبالتالي قد تظهر مشاكل لاحقة في عملية المزامنة بين الصوت والصورة ولحل هذه المشكلة يمكن دمج الـ RTP Stream الخاص بالصوت والـRTP Stream الخاص بالصورة بـ RTP Stream واحد وتميزها بإعطاء Identifier خاص لكل منها ويتكون الـ IdentifierللـCSRC من 32 bits وقد يصل عدد الـCSRC التي يمكن تحميلها على الـRTP Header إلى 16 بحجم أعظمي 32X16 = 480 bits.

٥- **Marker**: ويتكون من 1 bit وهو Flag يبين بداية ونهاية الإرسال لكل مجموعة من البيانات وكمثال في حالة نقل صورة Image Frame على أكثر من RTP Packet فإن ذلك الـMarker سيحتوي على قيمة ١ في أول Frame يتم إرساله لمعرفة بداية ونهاية الإرسال لتلك الصورة.

٦- **Payload Type**: ويتكون من 7 bits توضح فيها نوع البيانات التي سيتم تحميلها على الـRTP Packet وكما أوضحنا سابقا لا يمكن أن يتم تحميل أكثر من نوع من البيانات على نفس الـRTP Stream ويبين الجدول التالي أنواع الـRTP Payload التي يمكن تحميلها على الـRTP Stream:

PT	encoding name	audio/video (A/V)	clock rate (Hz)	channels (audio)	
0	PCMU	A	8000	1	[RFC3551]
1	Reserved				
2	Reserved				
3	GSM	A	8000	1	[RFC3551]
4	G723	A	8000	1	[Kumar]
5	DVI4	A	8000	1	[RFC3551]
6	DVI4	A	16000	1	[RFC3551]
7	LPC	A	8000	1	[RFC3551]
8	PCMA	A	8000	1	[RFC3551]
9	G722	A	8000	1	[RFC3551]
10	L16	A	44100	2	[RFC3551]
11	L16	A	44100	1	[RFC3551]
12	QCELP	A	8000	1	
13	CN	A	8000	1	[RFC3389]
14	MPA	A	90000		[RFC3551]
15	G728	A	8000	1	[RFC3551]
16	DVI4	A	11025	1	[DiPol]
17	DVI4	A	22050	1	[DiPol]
18	G729	A	8000	1	
19	reserved	A			
20	unassigned	A			
21	unassigned	A			
22	unassigned	A			
23	unassigned	A			
24	unassigned	V			
25	CelB	V	90000		[RFC2029]
26	JPEG	V	90000		[RFC2435]
27	unassigned	V			
28	nv	V	90000		[RFC3551]
29	unassigned	V			
30	unassigned	V			
31	H261	V	90000		[RFC2032]
32	MPV	V	90000		[RFC2250]
33	MP2T	AV	90000		[RFC2250]
34	H263	V	90000		[Zhu]
35--71	unassigned	?			
72--76	reserved for RTCP conflict avoidance				[RFC3550]
77--95	unassigned	?			
96--127	dynamic	?			[RFC3551]

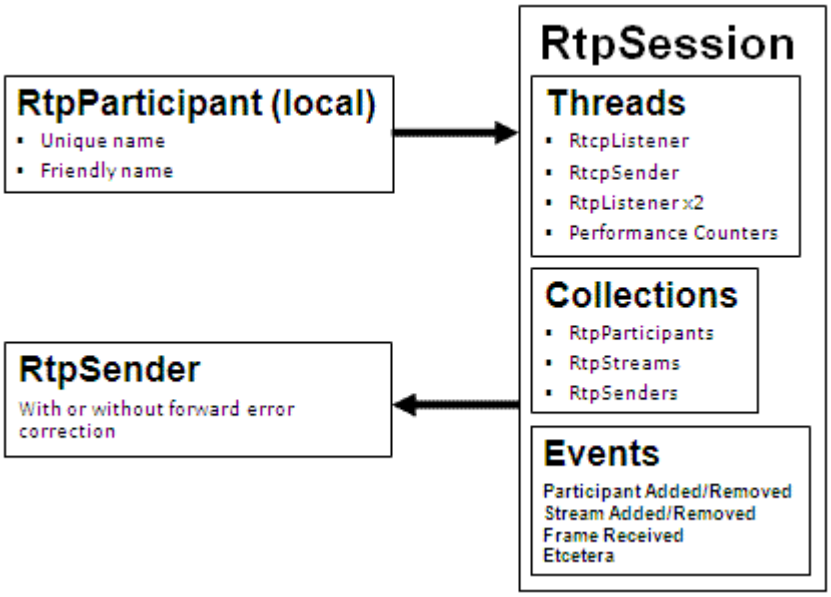
٧- **Sequence Number**: ويتكون من 16 bits ويحتوي على الرقم المتسلسل والذي يولد في البداية بشكل عشوائي ويزيد بمقدار واحد لكل RTP Packet يتم إرساله ويستفاد من هذا الرقم لمعرفة موقع الـRTP Packet عند استلامه لإجراء عملية الترتيب كذلك يستفاد منه في اكتشاف فقدان إي من الـRTP Packets المرسلة.

٨- **Timestamp**: ويتكون من 32 bits ويحتوي على الوقت الزمني الذي تم فيه تحميل البيانات على الـRTP Packet ويستفاد من هذه البيانات بشكل كبير لتغلب على مشكلات الـJitter والمزامنة بين عملية الإرسال والاستقبال والزمن المستغرق بينهما.

٩- **Synchronization Source Identifier SSRC** : ويتكون من 32 bits ويحتوي على رقم متسلسل يولد عشوائيا لتميز كل RTP Stream عن غيره بحيث لا يتشابه RTP Stream مع الآخر على نفس الـSession مما قد يؤدي إلى ما يسمى بالـRTP Stream Conflict.

استخدام الـ RTP In .NET:

لا يدعم الـ.NET Framework 3.5 أية Classes للـRTP Protocol لكن قدمت Microsoft مجموعة من الـThird Party Kits ضمن مشروعها المفتوح المصدر Microsoft Conference XP والذي يحتوي على نواة الـRTP Protocol بشكل كامل وقد تم تقسيمه إلى مجموعة من الـClasses بناء على كيفية عمل الـRTP وكما في الشكل التالي:



أولا الـRTPSession والـRtpParticipant : يستخدم الـRTP Session بشكل أساسي في عملية إدارة جلسة الاتصال والتي يمكن أن يتم فيها إرسال مجموعة من الـRTP Stream كما ويمكن أن يتصل بالـSession الواحدة مجموعة من المشتركين Participants كذلك يمكن للمشترك الواحد أن يتصل بأكثر من RTP Session وتميز كل RTP Session بالعنوان ورقم الـPort الذي يتم الإرسال له كذلك يتم إضافة CNAME لتميز محتويات كل RTP Session عن الأخرى وكمثال في الدوت نيت:

```
RtpSession rtpSession = new RtpSession(endpoint, new
RtpParticipant("My Audio Session", ParticipatorName),
true, true);
```

إذ يمرر الـEndpoint Object والذي يحتوي على عنوان Multicast IP بالإضافة إلى Port الاستقبال وينضم المرسل أو المستقبل إلى الـRTP Session بتمرير الـ RtpParticipant Object إلى الـRTP Session ولتعريفه يجب أن يتم تمرير الـCNAME واسم المتصل إليه بعد ذلك نكمل تعريف الـRTP Session بتمرير قيم True أو False ويحدد في الأول أنك تريد الانضمام إلى الـ RTP Session فقط ويحدد الثاني إذا كنت تنوي الإرسال باستخدام تلك Session.

ثانيا الـRtpEvents: وهو مجموعة الأحداث التي تحدث داخل الـRTP Session والتي يمكن أن يوضع Action معين عند حدوث أي منها وتقسم هذه الأحداث إلى ثلاثة أقسام وهي:

أولا أحداث ترتبط باكتشاف أخطاء أو مشاكل إثناء الإرسال أو الاستقبال ويمكن أن نسميها Exceptions Events وهي:

1- DuplicateCNameDetected
2- FrameOutOfSequence
3- HiddenSocketException
4- InvalidPacket
5- InvalidPacketInFrame
6- NetworkTimeout
7- PacketOutOfSequence
8- RtpParticipantTimeout
9- RtpStreamTimeout

ثانيا حدث وحيد لجلب معلومات Report عن حالة الـRTP Received Data وهو ReceiverReport.

ثالثا أحداث تتعلق بالانضمام إلى Session وإضافة وحذف RTP Session Participant & Stream Add/Remove Events وهي:

1- RtpParticipantAdded
2- RtpParticipantDataChanged
3- RtpParticipantRemoved
4- RtpStreamRemoved
5- RtpStreamAdded

وأكثر ما يهمنا في هذه الأحداث هو ضبط متى يتم الانضمام إلى RTP Session لبدأ عملية الاستقبال وكذلك متى يتم الخروج من الـRTP Session لإيقاف الاستقبال ولتعامل مع هذه الأحداث يجب أولا أن نقوم بعمل Hook لها وربط الحدث بالدالة التي سيتم تنفيذها عند حدوث ذلك الحدث وكما يلي كمثال:

```
private void HookRtpParticipantEvents()
{
// Add Remove Participant Events
RtpEvents.RtpParticipantAdded += new
RtpEvents.RtpParticipantAddedEventHandler(RtpParticipantA
dded);
RtpEvents.RtpParticipantRemoved += new
RtpEvents.RtpParticipantRemovedEventHandler(RtpParticipan
tRemoved);
}

private void HookRtpStreamEvents()
{
// Add Remove Stream Events
RtpEvents.RtpStreamAdded += new
RtpEvents.RtpStreamAddedEventHandler(RtpStreamAdded);
RtpEvents.RtpStreamRemoved += new
RtpEvents.RtpStreamRemovedEventHandler(RtpStreamRemoved);
}

private void RtpParticipantAdded(object sender,
RtpEvents.RtpParticipantEventArgs ea)
{
ShowMessage(ea.RtpParticipant.Name + " has joined");
}

private void RtpParticipantRemoved(object sender,
RtpEvents.RtpParticipantEventArgs ea)
{
ShowMessage(ea.RtpParticipant.Name + " has left");
}
```

```
private void RtpStreamAdded(object sender,
RtpEvents.RtpStreamEventArgs ea)
{
ea.RtpStream.FrameReceived += new
RtpStream.FrameReceivedEventHandler(FrameReceived);
}

private void RtpStreamRemoved(object sender,
RtpEvents.RtpStreamEventArgs ea)
{
ea.RtpStream.FrameReceived -= new
RtpStream.FrameReceivedEventHandler(FrameReceived);
}
```

وينطبق ذلك الشكل في التعامل مع كافة أحداث الـRTP Events السابقة الذكر.

ثالثا الـ RtpSender والـ RtpListener : يمكن الإرسال مباشرة بعد تعريف الـRTP Session ولتعريف الـRtpSender يجب أن يمرر له الـRTP Session الذي قمنا بتعريفه سابقا ويضاف إليها أيضا الـPayload Type الذي سيتم إرساله على الـRTP Session ولتمييز الـRTP Sender في الـSession عن غيره يمكن أن يتم تعريف Hash Table لذلك اختياريا أو قيمة null وتمريره لدالة CreateRtpSender وكما يلي كمثال:

```
RtpSender rtpsender = rtpSession.CreateRtpSender("My VOIP
Sender", PayloadType.dynamicAudio, null);
```

أو يمكن تعريفه بالشكل التالي بحيث تدعم عملية FEC – Frame Error Correction للـRTP Packet وذلك لدعم إمكانية تصحيح الـPacket عند وصوله مع وجود أخطاء بدلا من طلب إعادة الإرسال له ويصبح الشكل العام لدالة كما يلي كمثال:

```
RtpSender rtpsender = rtpSession.CreateRtpSenderFec("My
VOIP Sender With FEC", PayloadType.dynamicAudio, null,
CDataPX, CFecPX);
rtpSender.Send(buffer);
```

وأما بخصوص الاستقبال بالـ RtpListener فتعمل هذه الدالة ضمن الـ RTP Session وذلك بربط تلك الدالة بحدث الانضمام إلى الجلسة وفي هذه الحالة بمجرد انضمامك إلى الجلسة يتم تنفيذ الحدث وبدأ الاستقبال ولتنفيذ ذلك يجب أولا عمل Hook للـRTP Events التي قمنا بتعريفها سابقا وعند بدأ إرسال بيانات ما من قبل أي طرف من الأطراف يتم تنفيذ الحدث وبدأ الاستقبال ويمكنك الوصول إلى

محتويات البيانات المستقبلة من خلال الخاصية Frame.Buffer والمستلمة من الـ FrameReceivedEventArgs :

```
private void FrameReceived(object sender,
RtpStream.FrameReceivedEventArgs ea)
{
Byte[] Buufer = ea.Frame.Buffer;
}
```

رابعا الـRTCP وهو مختصر لـReal-Time Transport Control Protocol:
ومن أهم استخداماته أنه يعمل مع الـRTP لإدارة التحكم في العمليات التي تتم في أنظمة المؤتمرات وكذلك تقديم تقارير عن حالة تلك العمليات وتقسم إلى:

١- تقارير الإرسال SR وتقارير الاستقبال RR
٢- تفاصيل مرسل البيانات Source description SDES
٣- إدارة الانضمام والخروج لـ\ومن مجموعة Add Remove Membership.
٤- تعريف application-defined APP جديد على الـRTP Session.

ويمكن أن نستفيد من كل ذلك من خلال الـRTCP Namespace وكمثال يمكن استخدام الـ RtcpListener Class لتعامل مع البيانات التي يتم استقبالها من خلال الـRTP Session و إدارة عمليات الـMembership والتي ذكرناها سابقا وأيضا تقديم معلومات عن حالة شبكة الاتصال والتي تربط المرسل بالمستقبل، وسأقدم بدروس لاحقة معلومات أكبر عن استخدامات الـ RTCP Protocol في إدارة أنظمة الـConferencing.

التعامل مع الـDirect Sound والـRTP لعلية نقل الصوت كذلك حل بعض المشاكل المتعلقة بالـ Jitter في أنظمة الـ Multicast VOIP Conferencing:

تحدت في الدرس السابق عن مكونات الـRTP وكيفية استخدامه في بيئة الدوت نيت وفي هذا الدرس سنقوم بتطبيق برمجة نظام بسيط لبث الصوت من نقطة إلى مجموعة One-To-Many وسنقوم أيضا بالتحكم بخواص الـBuffer بهدف المزامنة بين عملية التسجيل والإرسال والعرض والتحكم به لتقليل الـDelay والـLoss في عملية الإرسال بناءا على سرعة الشبكة.

أولا: استخدام الـDirectSound في التعامل مع الصوت :

لتعامل مع الـDirectSound لابد أولا من إضافة الـMicrosoft.DirectX والـ Microsoft.DirectX.DirectSound Namespaces إلى الـReferences في المشروع ويحتوي الـDirectSound على عدد من الـClasses والتي تستخدم في برمجة كل ما يتعلق بالصوت تقريبا مثل التسجيل والـEncoding والعرض والكثير من الأمور وما يهمنا في هذا المشروع هو استخدام الـClasses التالية:

WaveFormat: ويستخدم لتحديد تفاصيل الـWave Format مثل عدد الـChannels مثلا ١ أو ٢ و ونوع الـModulation المستخدم مثل الـPCM-Pulse Code Modulation وعدد الـSamplesPerSecond وعدد الـBitsPerSample لأستخدم هذه المعلومات في عملية تحويل الذبذبات الصوتية إلى Bits لتمكين نقلها عبر الشبكة.

CaptureBufferDescription: ويستخدم لتحديد حجم الـ Bytes Buffer والذي سيتم حجزه في الذاكرة لاستقبال WAVE Bits الملتقطة.

CaptureDevicesCollection: وهو Array Of Devices ويستخدم لإرجاع كافة الـHardware Devices Info المتعلقة بوحدات الإدخال\الإخراج الصوتية المتاحة لديك لتحديد واحد منها في عملية التقاط الصوت من المايكروفون وعرض الصوت على السماعات.

DeviceInformation: ويستخدم لتحديد واحد من الـDevices التي سيتم إرجاعها من CaptureDevicesCollection حيث سيرجع الـ GUID Globally Unique Identifier الخاص بالـSound Driver لديك.

Capture: ويعرف به الـDriver GUID الذي يتم جلبه من DeviceInformation لاستخدامه في الـCaptureBuffer لالتقاط الصوت وتخزينه في الـBuffer.

CaptureBuffer: وهو الـClass المسئول عن عملية التقاط الصوت ووضعه في الـBuffer إذ يمرر له الـ CaptureBufferDescription Object والـCapture Object لبدأ عملية التقاط الصوت ضمن المعلومات الممررة له وتخزينه في الـ Buffer المحدد.

BufferPositionNotify: وهو Handler Notification Class يستخدم لتنفيذ Event محدد عند وصول الـBuffer إلى منطقة معينة.

Notify: ويحدد فيه عدد الـBytes التي إن وصل لها معبئ الـBuffer فسيتم عمل Trigger للـEvent الذي سيقوم بعملية معينة كتفريغ الـBuffer من جديد.

Device: وسيستخدم في طرف المستمع وفيه يتم تحديد الـSound Card الذي سيتم عرض الصوت عليه ويربط فيه جزأين أولا الـApplication الذي سيتم التعامل معه في عملية معالجة وعرض الصوت والثاني الـ CooperativeLevel Priority.

BufferDescription: وسيستخدم في طرف المستمع إذ يحتوي على مجموعة من الـProperties والـMethods لتحديد وإرجاع معلومات عن الـBuffer الذي سيستخدم في تجميع الـBytes الواردة من مصدر ما.

SecondaryBuffer: وسيستخدم في طرف المستمع ويمرر له الـ BufferDescriptionوالـDevice Object لبدأ عملية تجميع الـBuffer ومن ثم إمكانية عرضه على الـSound Device المحدد.

ثانيا: مراحل بناء نظام لنقل الصوت إلى مجموعة من المتصلين باستخدام الـDirectSound والـRTP:

بعدما قمنا بشرح موجز عن استخدام كل Class من الـClasses التي سيتم التعامل معها في الـDirect Sound سنتحدث في هذه الجزء عن كيفية استخدمها برمجيا بشكل مجموعة من المراحل المترابطة (سأقوم بإضافة تعليق باللغة الإنجليزية على كل سطر من اسطر المثال لتسهيل تتبع الكود):

١- التعامل مع الـSound Card لعملية تسجيل الصوت من المايكروفون والتعامل مع الـ WAVE Format :

المرحلة الأولى تعريف الـClasses الخاصة بالـDirect Sound وكما تم شرحها في الأعلى:

```
private CaptureBufferDescription
captureBufferDescription;
private AutoResetEvent autoResetEvent;
private Notify notify;
private WaveFormat waveFormat;
private Capture capture;
private CaptureBuffer captureBuffer;
private Device device;
private SecondaryBuffer playbackBuffer;
private BufferDescription playbackBufferDescription;
```

المرحلة الثانية سنقوم بإنشاء دالة لتعريف الـSound Card الذي سيستخدم في عملية تسجيل الصوت وكذلك هي نفسها عند المستقبل:

```
public void SetVoiceDevices(System.Windows.Forms.Control
AppForm_TypeThis)
{
// Use The Recommended settings For Sound Devices
```

```
SetVoiceDevices(
0, // Device Number (First Device)
1, // Channels      (2 if Stereo)
AppForm_TypeThis, // Application Form Pointer
16, // BitsPerSample
22050); // SamplesPerSecond
}

public void SetVoiceDevices(int deviceID, short channels,
System.Windows.Forms.Control AppForm_TypeThis, short
bitsPerSample, int samplesPerSecond)
{
// Installization Voice Devices
device = new Device(); // Sound Input Device
device.SetCooperativeLevel(AppForm_TypeThis,
CooperativeLevel.Normal); // Set The Application Form and
Priority
CaptureDevicesCollection captureDeviceCollection = new
CaptureDevicesCollection(); // To Get Available Devices
(Input Sound Card)
DeviceInformation deviceInfo =
captureDeviceCollection[deviceID]; // Set Device Number
capture = new Capture(deviceInfo.DriverGuid); // Get The
Selected Device Driver Information

//Set up the wave format to be captured.
waveFormat = new WaveFormat(); // Wave Format declaration
waveFormat.Channels = channels; // Channels  (2 if
Stereo)
waveFormat.FormatTag = WaveFormatTag.Pcm; // PCM - Pulse
Code Modulation
waveFormat.SamplesPerSecond = samplesPerSecond; // The
Number of Samples Peer One Second
waveFormat.BitsPerSample = bitsPerSample; // The Number
of bits for each sample
waveFormat.BlockAlign = (short)(channels * (bitsPerSample
/ (short)8)); // Minimum atomic unit of data in one byte,
Ex: 1 * (16/8) = 2 bits
waveFormat.AverageBytesPerSecond = waveFormat.BlockAlign
* samplesPerSecond; // required Bytes-Peer-Second Ex.
22050*2= 44100
captureBufferDescription = new
CaptureBufferDescription();
captureBufferDescription.BufferBytes =
waveFormat.AverageBytesPerSecond / 5; //Ex. 200
milliseconds of PCM data = 8820 Bytes (In Record)
captureBufferDescription.Format = waveFormat; // Using
Wave Format
```

```
// Playback
playbackBufferDescription = new BufferDescription();
playbackBufferDescription.BufferBytes =
waveFormat.AverageBytesPerSecond / 5;  //Ex. 200
milliseconds of PCM data = 8820 Bytes (In Playback)
playbackBufferDescription.Format = waveFormat;
playbackBuffer = new
SecondaryBuffer(playbackBufferDescription, device);
bufferSize = captureBufferDescription.BufferBytes;
}
```

المرحلة الثالثة إنشاء الدالة الخاصة بالتقاط الصوت لتجهيز الضغط والإرسال:

```
private void StartRecordAndSend() // Send Recorded Voice
{
captureBuffer = new
CaptureBuffer(captureBufferDescription, capture); // Set
Buffer Size,Voice Recording Format & Input Voice Device
SetBufferEvents(); // Set the events Positions to Send
While Recording
int halfBuffer = bufferSize / 2; // Take the half buffer
size
captureBuffer.Start(true); // start capturing
bool readFirstBufferPart = true; // to know which part
has been filled (the buufer has been divided into tow
parts)
int offset = 0; // at point 0
MemoryStream memStream = new MemoryStream(halfBuffer); //
set the half buffer size to the memory stream

while (True)
{
//WaitOne() Blocks the current thread until the current
WaitHandle receives a signal
//WaitHandle("Encapsulates operating system-specific
objects that wait for exclusive access to shared
resources")
autoResetEvent.WaitOne();
memStream.Seek(0, SeekOrigin.Begin); //Sets the position
within the current stream to 0
captureBuffer.Read(offset, memStream, halfBuffer,
LockFlag.None); // capturing and set to MemoryStream
readFirstBufferPart = !readFirstBufferPart; // reflecting
the boolean value to set the new comming buffer to the
other part
```

```
offset = readFirstBufferPart ? 0 : halfBuffer; // if
readFirstBufferPart set to true then set the offset to 0
else set the offset to the half buffer
byte[] dataToWrite = memStream.GetBuffer;
// Here you can Compress the voice buffer
.
.
// Sending the compressed voice across Network
.
.
}
}
```

٢- إدارة الـBuffering وذلك بتجزيء الـBuffer إلى جزأين الأول يستخدم في عملية تخزين الصوت المسجل والثاني لتجهيزه لعملية المعالجة كإرساله أو ضغطه:

```
protected void SetBufferEvents()
{
// Goal: To Send While Recording
// To Set The Buffer Size to get 200 milliseconds and
divide it in half,
// so that when the first half is filled the data can be
used to send,
// while the second half of the buffer is being filled
with PCM Data

try
{
autoResetEvent = new AutoResetEvent(false); // To wait
for notifications
notify = new Notify(captureBuffer); // The number of
bytes that can trigger the notification event

// the first half
BufferPositionNotify bufferPositionNotify1 = new
BufferPositionNotify(); // to describe the notification
position
bufferPositionNotify1.Offset = bufferSize / 2 - 1; // (=
At the Half of The Buffer) to know where the notify event
will trigger
bufferPositionNotify1.EventNotifyHandle =
autoResetEvent.SafeWaitHandle.DangerousGetHandle(); //
Set The Event that will trigger after the offset reached

// the last half
```

```
BufferPositionNotify bufferPositionNotify2 = new
BufferPositionNotify();
bufferPositionNotify2.Offset = bufferSize - 1; // (= At
The Last Buffer)
bufferPositionNotify2.EventNotifyHandle =
autoResetEvent.SafeWaitHandle.DangerousGetHandle();

notify.SetNotificationPositions(new
BufferPositionNotify[] { bufferPositionNotify1,
bufferPositionNotify2 }); // The Tow Positions (First &
Last)
}
catch (Exception) { }
}
```

٣- ضغط الـBuffer المستلم باستخدام معايير الضغط المتاحة: في هذه المرحلة فالخيار متاح لك في اختيار طريقة الضغط المناسبة ويمكنك أيضا في هذه المرحلة القيام بعملية Coding معينة على الـBuffer مثلا تشفيره ثم ضغطه لكن وجب في البداية دراسة أنواع الضغط والتشفير المناسبة لصوت بهدف المحافظة على الـQoS Quality Of Server والتي سيتم التطرق لها لاحقا.

٤- تغليف الـBuffering في RTP Packet وإرساله عبر الشبكة إلى Multicast RTP Session:
قمت سابقا بشرح كيفية استخدام الـRTP Protocol لعملية الانضمام إلى مجموعة وكذلك كيفية تغليف الـByte Data بالـRTP Packet لمزيد من المعلومات أنظر الدرس الثاني.

٥- في الطرف المقابل (المستقبل) سيقوم بالانضمام إلى RTP Session ثم بدأ عملية الـListening على الـSession لاستقبال كما تم شرح هذه العملية في الدرس الثاني إذ أنه بعد الانضمام إلى RTP Session سيتم تنفيذ الـ RTP Session Event والذي بدوره سيمكنك من استقبال البيانات الواردة من المرسل وبعد ذلك يمكننا تجميع الـBuffer ثم عرضه.

```
private void RtpStreamAdded(object sender,
RtpEvents.RtpStreamEventArgs ea)
{
ea.RtpStream.FrameReceived += new
RtpStream.FrameReceivedEventHandler(FrameReceived);
```

```
}

private void FrameReceived(object sender,
RtpStream.FrameReceivedEventArgs ea)
{
PlayReceivedVoice(ea.Frame.Buffer);
}
```

٦- بعد عملية الاستقبال يتم تجميع الـBytes المستقبلة في Buffer لإجراء عملية فك الضغط ومن ثم عرضه على Sound Device لديك.

```
private void PlayReceivedVoice(byte[] VoiceBuffer)
{

//Decompress the received data
// byte[] byteDecodedData = Decompress(VoiceBuffer);

//Play it on the speaker device.
playbackBuffer = new
SecondaryBuffer(playbackBufferDescription, device);
playbackBuffer.Write(0, byteDecodedData, LockFlag.None);
// 0= is the Starting Point (the offset)
playbackBuffer.Play(0, BufferPlayFlags.Default); // 0 =
is The Priority of Sound for hardware that mixing the
voice resources

}
```

ثالثا: دراسة حجم الـBuffer وتأثره بسرعة الشبكة وعلاقة ذلك بالـJitter Loss And Delay:

سنقدم في هذا الدرس كيفية حساب حجم الـBuffer و التحكم به بناء على عوامل قدرة الشبكة وكذلك الجودة.

بينا سابقا أن الزيادة في حجم الـBuffer يساعد المرسل على تقسيم البيانات المراد إرسالها وبالتالي إمكانية إرسال إحجام اكبر من البيانات على سرعات شبكة أقل لكن كما وذكرنا إذ أن ذلك يساعد على أمرين:

الأول: إمكانية ضياع Frames إثناء الإرسال بسبب زيادة عدد تلك الـFrames.

الثاني: حاجة المرسل إلى وقت أكثر لتقسيم تلك الـFrames وحاجة المستقبل لوقت و Buffer أكبر لإعادة ترتيب تلك الـFrames وبالمحصلة زيادة الـDelay.

في البداية سنقوم بتعريف أهم المصطلحات والتي سنستخدمها في عملية إدارة الـBuffer والتي قمنا بتخصيصها بدرس السابق في الدالة SetVoiceDevices:

Channels: وهو عدد القنوات التي يتم من خلالها تسجيل الصوت وعرضه وكمثال فإن نظام الـStereo يستخدم قناتين ويضاعف عدد القنوات المستخدمة حجم الـBuffer بحيث أنه يضرب بعدد القنوات التي يتم استخدامها.

Sample Peer Second: ويسمى أيضا Sampling rate وهي تعبير عن عدد الـSamples بالثانية الواحدة ومن المعروف أن الـSample هو قيمة لموقع معين من الموجة الصوتية في زمن محدد وأما الـSampling فهو عملية تحويل الـcontinuous signal إلى discrete signal بهدف تحويل تلك الـSignal إلى النظام الثنائي.

Bits Peer Sample : ويعبر عنه بالـbit depth وهو عدد الـBits التي ستحتاجها لتحويل Wave Sample واحدة من النظام التناظري إلى الرقمي وكلما زاد عدد الـbits كلما زادت جودة الصوت وزاد الاحتياج إلى حجم Buffer أكبر.

وبالتالي لحساب حجم الـBuffer الذي سنحتاجه في عمليات الإرسال والاستقبال ومن خلال المعادلة التالية يمكننا معرفة الحد الأدنى لحجم الـBuffer الذي سنحتاجه لإجراء عملية الإرسال:

Bit rate = (bit depth) x (sampling rate) x (number of channels).

The Minimum Size Of The Buffer in Bytes = Channels X (BitsPerSample/8).

Example: 2 X (16 bits /8) = 4 Bytes

The Minimum Size Of The Buffer in One Second = (The Minimum Size Of The Buffer in Bytes) X (Number of Samples Peer One Second)

*Example: 4Bytes * 22050 = 88200 Bytes For each Second.*

وهذا يعني أننا سنحتاج إلى Buffer لا يقل عن 88200 Bytes لتخزين ثانية واحدة من الصوت قبل عملية الضغط أو الإرسال وبتالي ولحساب السرعة المطلوبة من قناة الاتصال في الثانية الواحدة والذي إذا تحقق فسيكون هنالك مزامنة في عملية التحدث وبالتالي وصول الصوت بشكل مستمر وغير متقطع سنطبق الخطوات التالية:

أولا حساب عدد الـFrames المحتمل والذي سنحتاجها لإرسال 88200 Bytes في الثانية الواحدة:

(88200 the size of the buffer for each second) / (1500 Bytes the minimum size of the frame In Ethernet as example - 40 Bytes The Minimum Size of The Empty RTP Frame) = 61 Frames

ثانيا حساب حجم الـFrames الفارغة:

61 Frames X 40 Bytes = 2440 Bytes

ثالثا جمع كافة القيم وضربها بـ٨ لتحويلها إلى Bits وتقسيمها على ١٠٢٤ لمعرفة السرعة المطلوبة بالـ KB :

(88200 + 2240) (8) / 1024 = 706 K bit/Second

وهذا يعني أننا سنحتاج إلى شبكة بسرعة لا تقل عن 706Kb/S لنقل ثانية واحدة من الـWave Voice بمواصفات: قناتين اتصال و Sampling Rate 22050 و 16 Bits لجودة الصوت.

ولمزامنة عملية التسجيل و الإرسال قمنا بتنفيذ الدالة SetBufferEvents وذلك لتقسيم الـBuffer إلى جزأين الأول سيستخدم في تسجيل الصوت القادم من المايكروفون والثاني سيستخدم في عملية الإرسال، ولتنفيذ ذلك قمنا بإنشاء اثنين من الـNotifications الأول سينفذ عندما يصل تعبئة الـBuffer إلى النصف بحيث يعلن فيه عن امتلاء الجزء المخصص لتخزين الصوت وبالتالي إضافته إلى الجزء التالي من الـBuffer المخصص للإرسال وفي حالة امتلاء الجزء الثاني سيتم إرسال الـBuffer ومن ثم تنظيفه من جديد لتكرار العملية.

يمكن التحكم أيضا بزمن الإرسال ويعتمد ذلك على سرعة الشبكة من جهة وحجم الـBuffer من جهة أخرى وقد قمنا في المثال السابق بإرسال كامل الـBuffer عند كل 200 milliseconds وذلك بعد ضغطه باستخدام المعيار G.711 وهو ITU Standard ويعتبر من إحدى معايير الضغط المخصصة لصوت ويمكن باستخدام ذلك المعيار تقليل حجم الـWave Format إلى النصف تقريبا. لمزيد من المعلومات حول كيفية عمل هذا المعيار يمكن الرجوع إلى الرابط التالي:

http://en.wikipedia.org/wiki/G.711

لتحميل المثال الخاص بالمقال:

http://www.socketcoder.com/ArticleFile.aspx?index=2&ArticleID=64

مقالة ملخصة من الإصدار الثالث من كتاب المرجع الإلكتروني في برمجة الشبكات

فادي عبدالقادر ٢٠٠٩ – SocketCoder.Com