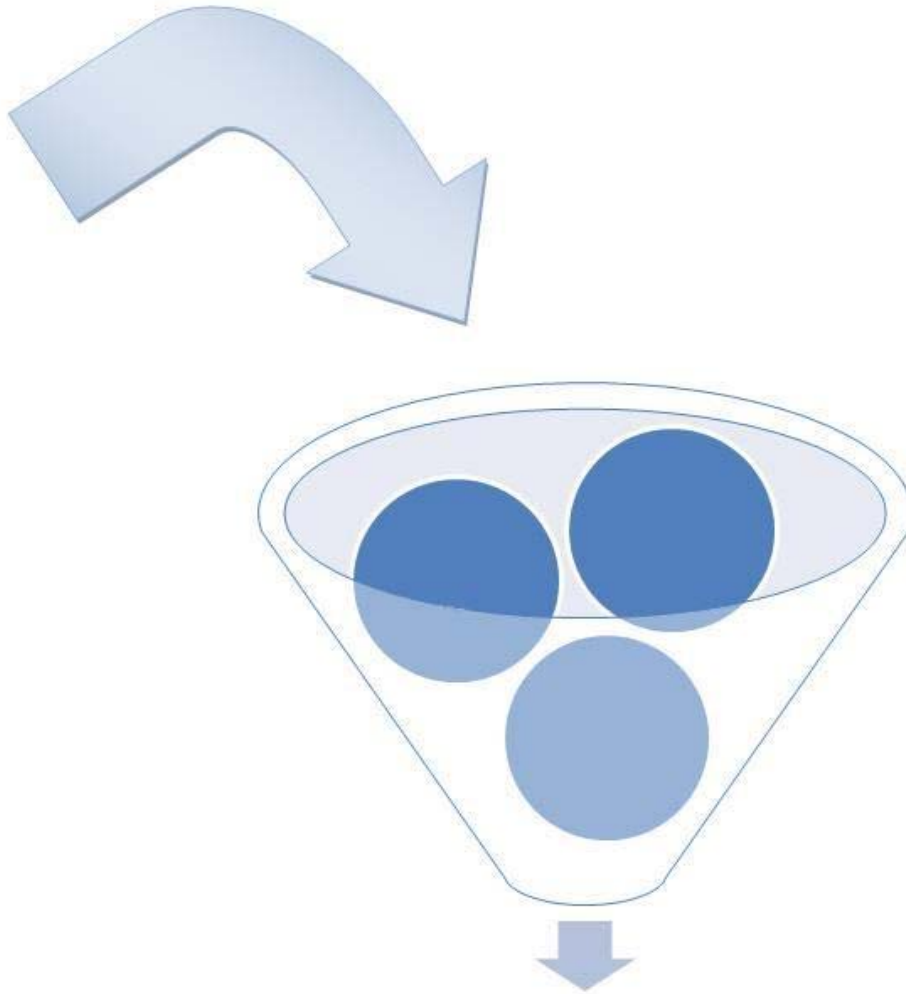




المواضيع الرئيسية في هذا الفصل:

- 1- مدخل إلى برمجة الـ Socket بالدوت نت
- 2- معمارية البرمجة المتزامنة في الـ TCP Unicasting
- 3- معمارية البرمجة المتزامنة في الـ UDP Unicasting, Multicasting and Broadcasting

3.1: مدخل إلى برمجة الـ Socket بالدوت نت :

تعرف الـ Socket على أنها الأداة المنطقية التي يتم من خلالها ربط كل من الـ Transport Layer والـ Network Layer في بقية طبقات الـ TCP/IP ومن خلالها يمكننا تعريف البروتوكول المستخدم في عملية النقل والـ Port الخاص به وكذلك الـ IP Address لطرف المقابل. وقد تم دعم الـ .NET Framework بمجموعة من الـ Classes والتي تمكننا من التعامل مع الـ Socket ضمن الـ System.Net.Socket Namespace ويبين الجدول التالي أهم هذه الـ Classes:

Class	Description
Socket	وهو الـ Class الأساسي للتعامل مع الـ Socket بالدوت نت ويحتوي على جميع التعريفات والإعدادات المحتمل التعامل معها إنشاء عمل أي نوع من الاتصال باستخدام أي نوع من بروتوكولات النقل في الـ Transport Layer و الـ Network Layer حيث يتم تعريفها وتحديدها بشكل يدوي.
TcpClient TcpListener UdpClient	تعرف هذه الـ Classes بالـ Helper Socket Classes وهي بالأساس Simple Socket Classes معرفة وجاهزة لاستخدامها بشكل مباشر بالبرنامج بحيث تعتمد الإعدادات الافتراضية والهدف من وجودها تسهيل التعامل مع الـ Socket بشكل عام وهي بديل عن استخدام الـ Socket Class لمن يريد استخدام الـ Socket لإرسال بيانات مباشرة بأحد البروتوكولين الـ TCP أو الـ UDP.

ويبين الشكل التالي التركيب العام لـ Socket Class بالدوت نت:

Socket Class Declaration:

```
System.Net.Sockets.Socket MySocket = new Socket(
AddressFamily, // Network Layer Example: InterNetwork
, SocketType, //Socket Transport Type For Example: Stream
, ProtocolType, //Transport Protocol For Exmaple TCP );
```

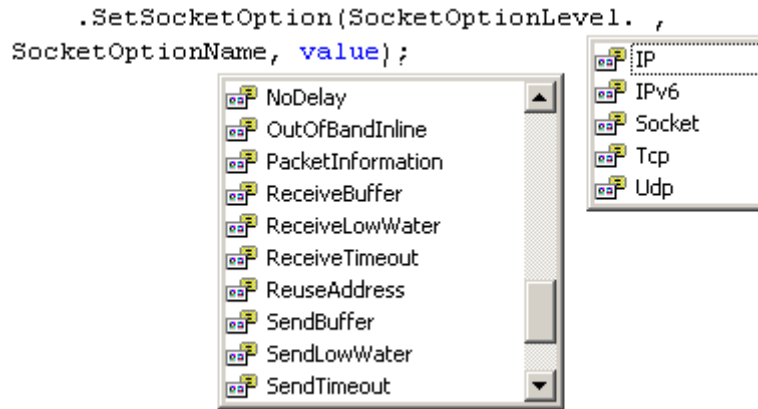
حيث يتم في الباروميتر الأول تحديد الـ Network Layer Protocol والذي سوف تتعامل معه في تحديد عنوان المرسل والمستقبل ويعطيك عدد من الخيارات ومنها IPX والمستخدم في شبكات الـ Novel أو ATM أو NetBIOS Address وغيرها، ومن أهم هذه الخيارات الـ InterNetwork ويوضع عند التعامل مع بروتوكول الـ IPv4 والـ InterNetworkV6 والذي يوضع عند التعامل مع الـ IPv6.

في الباروميتر الثاني يتم تحديد الـ Socket Transport Type وينقسم أسلوب النقل في العادة إلى ثلاثة أقسام رئيسية وهي Stream أو Dgram (Data gram) أو Raw Socket وبشكل دائم يتم استخدام الـ Stream في حالة اعتماد الـ TCP في عملية النقل والـ Dgram في حالة التعامل مع الـ UDP في عملية النقل والـ Raw Socket مع البروتوكولات التي لا تطلب التعامل مع الـ Transport Layer Protocols بحيث لا تحتاج في عملية التراسل إلى استخدام Port محدد ومن الأمثلة عليها الـ ICMP (لمزيد من المعلومات انظر الفصل السابع Raw Socket Programming)، وأخيرا نحدد نوع البروتوكول المستخدم للاتصال هل هو UDP أو TCP أو بروتوكولات أخرى مثل الـ ICMP Internet Control Message Protocol أو الـ IGMP Internet Group Management Protocol وغيرها، وهنا سوف نختار الـ TCP أو الـ UDP ومن المعروف أن بروتوكول الـ TCP هو بروتوكول موجه وهذا يعني إجراء عملية التحقق من الوصول والتوصيل إلى شخص ما

محدد أما بروتوكول الـ UDP فهو بروتوكول سريع نسبيا و لكنه لا يدعم عملية التحقق من الوصول السليم للبيانات المرسله وهو مفيد جدا لإجراء عملية البث Broadcast وإنشاء مجموعات البث Multicast Group وهو ما سيتم شرحه لاحقا في هذا الفصل.

3.1.1: التعديل على خواص الـ Socket باستخدام الدالة SetSocketOption:

يمكننا من خلال الخاصية SetSocketOption التعديل في أي أمر له علاقة بعملية الاتصال سواء على مستوى الـ Network Layer أو الـ Transport Layer أو حتى على مستوى الـ Socket نفسه كما وسنحتاج هذه الدالة في تفعيل الـ Multicasting و الـ Broadcasting على الـ Socket أما الصيغة العامة لها فهي كالتالي:



من خلال الـ Socket Object نختار الدالة SetSocketOption حيث تحتوي على ثلاثة باروميترات الأول يحدد مستوى التعديل المطلوب Socket Option Level وتقسم إلى ثلاثة مستويات:

الأول: على مستوى طبقة الـ Network Layer ويتبين بالـ IP والـ IPv6 وسنحتاجه عند الحاجة لتعديل شيء ما على مستوى الـ IP مثل الـ TTL أو إدارة الـ Multicasting.

الثاني: على مستوى طبقة الـ Transport ويحدد بالـ TCP والـ UDP Transport Protocols.

الثالث: على مستوى الـ Socket ويحدد عند الحاجة لتعديل في كلا الطبقتين الـ Network والـ Transport.

كما يحدد في الباروميتر الثاني نوعية الـ Option التي نريد تفعيلها أو منعها أو تغيير في قيمها ويبين الجدول 3.1.1.1 ما يمكننا القيام به من خلال هذه الدالة.

سيتم التطرق إلى هذه الـ Options خلال موضوعات فصول الكتاب عند الحاجة إليها.

ويوضع في الباروميتر الأخير الـ Object Value فإذا كانت العملية هي السماح أو الرفض عندها تأخذ إما True أو False وإذا كانت تغيير أو تحديد قيم ما فتأخذ القيمة التي نريد تغييرها.

Member name	Description
AcceptConnection	Socket is listening.
AddMembership	Add an IP group membership.
AddSourceMembership	Join a source group.
BlockSource	Block data from a source.
Broadcast	Permit sending broadcast messages on the socket.
BsdUrgent	Use urgent data as defined in RFC-1222. This option can be set only once, and once set, cannot be turned off.
ChecksumCoverage	Set or get UDP checksum coverage.
DontFragment	Do not fragment IP datagrams.
DontLinger	Close socket gracefully without lingering.
DontRoute	Do not route; send directly to interface addresses.
DropMembership	Drop an IP group membership.
DropSourceMembership	Drop a source group.
ExclusiveAddressUse	Enables a socket to be bound for exclusive access.
HeaderIncluded	Indicates application is providing the IP header for

	outgoing datagrams.
IPOptions	Specifies IP options to be inserted into outgoing datagrams.
IpTimeToLive	Set the IP header time-to-live field.
KeepAlive	Send keep-alives.
MaxConnections	Maximum queue length that can be specified by Listen.
MulticastInterface	Set the interface for outgoing multicast packets.
MulticastLoopback	IP multicast loopback.
MulticastTimeToLive	IP multicast time to live.
NoChecksum	Send UDP datagrams with checksum set to zero.
NoDelay	Disables the Nagle algorithm for send coalescing.
OutOfBandInline	Receives out-of-band data in the normal data stream.
PacketInformation	Return information about received packets.
ReceiveBuffer	Send low water mark.

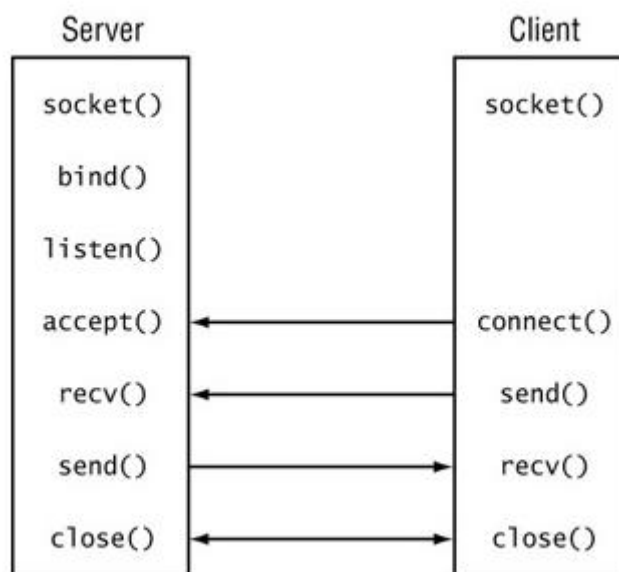
ReceiveLowWater	Receive low water mark.
ReceiveTimeout	Receive time out.
ReuseAddress	Allows the socket to be bound to an address that is already in use.
SendBuffer	Specifies the total per-socket buffer space reserved for sends. This is unrelated to the maximum message size or the size of a TCP window.
SendLowWater	Specifies the total per-socket buffer space reserved for receives. This is unrelated to the maximum message size or the size of a TCP window.
SendTimeout	Send timeout.
TypeOfService	Change the IP header type of service field.
UnblockSource	Unblock a previously blocked source.
UseLoopback	Enable Or Disable a Loop Back Connection

الجدول 3.1.1.1 ويبين إعدادات الـ Socket التي يمكننا القيام بها من خلال الدالة SetSocketOption (المصدر Microsoft MSDN)

3.2: معمارية البرمجة المتزامنة في الـ TCP Transport Protocol:

لإجراء أي عملية اتصال من خلال الـ TCP لابد في البداية من وجود طرف المستقبل وفي العادة يكون ذلك الطرف إما الـ Server في حالة الـ Client/Server Architecture أو الـ Peer Client في حالة الـ Peer-to-Peer Architecture لمزيد من المعلومات أنظر الفصل الحادي عشر.

ويمكننا استخدام الـ TCP فقط في حالة التراسل كـ Unicast وبالتالي هنالك طرف واحد فقط يستطيع استقبال البيانات بحيث يخصص Connection لكل متصل، ويمر الـ TCP بمجموعة من المراحل ويوضح الشكل 3.1 هذه المراحل لكلا الطرفين المتصل والمشار إليه بالـ Client والمستضيف والمشار إليه بالـ Server.



الشكل 3.1 ويبين مراحل الاتصال من خلال الـ TCP بين طرفين المتصل والمستضيف

ولإنشاء برنامج اتصال بسيط باستخدام الـ TCP يجب تعريف الـ Socket Class عند كلا الطرفين المتصل والمستضيف بالشكل التالي:

TCP Socket Class Declaration:

```

System.Net.Sockets.Socket MySocket = new Socket(
    AddressFamily.InterNetwork //InterNetwork = IPv4
    , SocketType.Stream // Streaming Session
    , ProtocolType.Tcp);

```

في طرف المتصل يتم من خلال الدالة Connect بدأ محاولة الاتصال بالطرف المقابل ويمرر إليها الـ IP Address والـ Port Number لطرف المستضيف مباشرة أو من خلال الـ IPEndPoint Class.

TCP Socket - Using The Connect Method:

```
// Pass The IP Address and The Port Number Directly
MySocket.Connect("127.0.0.1", 5000);

// Or Pass it By Using The IPEndPoint Class
System.Net.IPEndPoint ServerAddress = new
System.Net.IPEndPoint(System.Net.IPAddress.
    Parse("127.0.0.1"), 5000);
MySocket.Connect(ServerAddress);
```

بعد النجاح في عملية الاتصال يتم عمل Stream Session منطقي بين المتصل والمستضيف وتستطيع من خلاله البدء في عملية الإرسال والاستقبال ويتم من خلاله أيضا التأكد من أن عملية الاتصال مستمرة بين الطرفين. عند الإرسال لابد في البداية من عملية تحويل البيانات المراد إرسالها إلى Array of Bytes وذلك من خلال عمل Encoding لها ويعتمد ذلك على نوع البيانات المراد إرسالها فمثلا في حالة إرسال نص يتم تحويل ذلك النص من string إلى Bytes بحسب طريقة الترميز الخاصة به إذا كانت Unicode أو ASCII Code كمثال.

TCP Socket - Start Sending & Receiving in The Client Socket:

```
// For Sending:
byte[] TheStringAsByteArray =
UnicodeEncoding.Unicode.GetBytes("Hello World!");
MySocket.Send(TheStringAsByteArray);

// For Receiving:
//The initial declaration of the array indexes
byte[] ReceivedData = new byte[1024];
MySocket.Receive(ReceivedData);
string ReceivedMessage = UnicodeEncoding.Unicode
    .GetString(ReceivedData);
// To Close The Connction:
MySocket.Close();
```

أما في طرف المستضيف فيجب في البداية تعريف الـ Socket ثم عمل الـ Bind وهو ربط الـ IP Network الخاص بالمستضيف بالـ Transport Port التي سيتم من خلالها استقبال طلب الاتصال بعد ذلك ومن خلال الدالة Listen تبدأ عملية التصنت على الـ TCP Port الذي تم تحديده مسبقا ثم عند استقبال طلب اتصال يتم تحويله مباشرة إلى الدالة Accept والتي تقوم بدورها بإنشاء الـ Session بين المتصل والمستقبل ويتم وضعه على Socket Class Object خاص لكي يتم التعامل معه بشكل منفصل في عملية التراسل.

TCP Socket - The Server Part:

```
// The Server Socket Declaration
Socket ServerSocket = new Socket(
AddressFamily.InterNetwork, SocketType.Stream ,
ProtocolType.Tcp);

// Any = Use All IP Addresses of the Server
IPEndPoint ipend = new IPEndPoint(IPAddress.Any, 5000);

// Bind Both The Network IP and The Transport Port
ServerSocket.Bind(ipend);

// -1= Allow for Unlimited number of Connections
ServerSocket.Listen(-1);

// Make a Special Socket For Each Client
Socket New_Client = ServerSocket.Accept();

// Sending:
byte[] SendingBuffer =
UnicodeEncoding.Unicode.
    GetBytes("Welcome to Our Server");
New_Client.Send(SendingBuffer);

// Receiving:
byte[] ReceivedBuffer = new byte[1024];
New_Client.Receive(ReceivedBuffer);
string msg =
UnicodeEncoding.Unicode.GetString(ReceivedBuffer);
```

ولتكرار عملية الإرسال والاستقبال لا بد من وضع جملة تكرار مثل الـ While بعد عملية الـ Listen وبالتالي يبقى المستضيف يعمل إلى أن ينفذ شرط ما كتمرير جملة معينة أو انتهاء عدد الدورات أو جعله يعمل إلى الأبد بوضع True مع جملة التكرار While وفي هذه الحالة فلا بد من فصل دالة الإرسال والاستقبال بوضعها على Thread خاص لكي لا يؤثر ذلك على عمل البرنامج بشكل عام وهو ما تم شرحه سابقا في الفصل الثاني وكمثال على ذلك:

TCP Socket - The Server Part With Using Threading:

```
void StartTheServerOnNewThread()
{
    Thread ServerThread = new Thread(new
        ThreadStart(ServerMethod));
    ServerThread.IsBackground = true;
    ServerThread.Start();
}

void ServerMethod()
{
    Socket ServerSocket = new
    Socket(AddressFamily.InterNetwork, SocketType.Stream,
    ProtocolType.Tcp);

    // Socket Binding
    IPEndPoint ipend = new IPEndPoint(IPAddress.Any, 5000);
    ServerSocket.Bind(ipend);

    // Lestening
    ServerSocket.Listen(-1);

    while (true)
    {
        Socket New_Client = ServerSocket.Accept();
        byte[] buffer = new byte[1024];
        New_Client.Receive(buffer);
        string ReceivedMessage =
        UnicodeEncoding.Unicode.GetString(buffer);
    }
}
```

3.2.1: الـ Cross Threading Exception والطريقة الأنسب لحلها:

دائماً ما نحتاج إلى عرض البيانات التي تم استقبالها من خلال دالة الـ Thread على Windows Control موجود على الـ Form كمثل الـ Listbox Windows Form Control وفي العادة ما نقوم به هو التالي:

Cross Thread Exception:

```
void ServerMethod()
{
    Socket ServerSocket = new
    Socket(AddressFamily.InterNetwork, SocketType.Stream,
    ProtocolType.Tcp);

    // Socket Binding
    IPEndPoint ipend = new IPEndPoint(IPAddress.Any, 5000);
    ServerSocket.Bind(ipend);

    // Lestening
    ServerSocket.Listen(-1);

    while (true)
    {
        Socket New_Client = ServerSocket.Accept();
        byte[] buffer = new byte[1024];
        New_Client.Receive(buffer);
        string ReceivedMessage =
        UnicodeEncoding.Unicode.GetString(buffer);
        listBox1.Items.Add(ReceivedMessage);
    }
}
```

عند تنفيذ البرنامج سيتم إرجاع Cross-Thread operation not allowed exception عند استقبال أي بيانات ومحاولة عرضها على Listbox والسبب في ذلك يعود إلى كون أن Listbox Control يعمل على Thread آخر غير المعرف عليه ذلك الـ Control ولتجاوز هذا الأمر ينصح بتمرير البيانات التي تم استقبالها من خلال وسيط وهو الـ Delegate حيث يتم عمل Invoke للبيانات المستقبلية من خلاله إلى الـ Windows Control ليصبح بشكل التالي:

Solve The Cross Thread Exception:

```
void ServerMethod()
{
    Socket ServerSocket = new
    Socket(AddressFamily.InterNetwork, SocketType.Stream,
    ProtocolType.Tcp);
```

```
// Socket Binding
IPEndPoint ipend = new IPEndPoint(IPAddress.Any, 5000);
ServerSocket.Bind(ipend);

// Lestening
ServerSocket.Listen(-1);

while (true)
{
    Socket New_Client = ServerSocket.Accept();
    byte[] buffer = new byte[1024];
    New_Client.Receive(buffer);
    string ReceivedMessage =
    UnicodeEncoding.Unicode.GetString(buffer);

    // Create a delegate to handle the cross-thread calls
    this.Invoke(new mydelegate(ViewText), ReceivedMessage);
}

// to pass the received text using this delegate
private delegate void mydelegate(string PassedData);
private void ViewText(string PassedData)
{
    listBox1.Item.Add(PassedData);
}
```

قمنا في المثال السابق بتعريف delegate يمرر باروميتر من نوع String ثم قمنا بإنشاء دالة ضمن الـ Main Thread الرئيسي حيث مررنا من خلالها النص الذي سوف يتم تمريره من خلال الـ Delegate باستخدام دالة الـ Invoke إلى الـ Listbox Control ويعتبر هذا الحل هو الحل الأمثل لتجاوز هذا الاستثناء كما يوجد طريقة لإيقاف ذلك الاستثناء وذلك بتمرير قيمة False إلى الخاصية CheckForIllegalCrossThreadCalls ضمن الـ Control Class ووضعها على حدث بداية تشغيل البرنامج كما يلي كمثال:

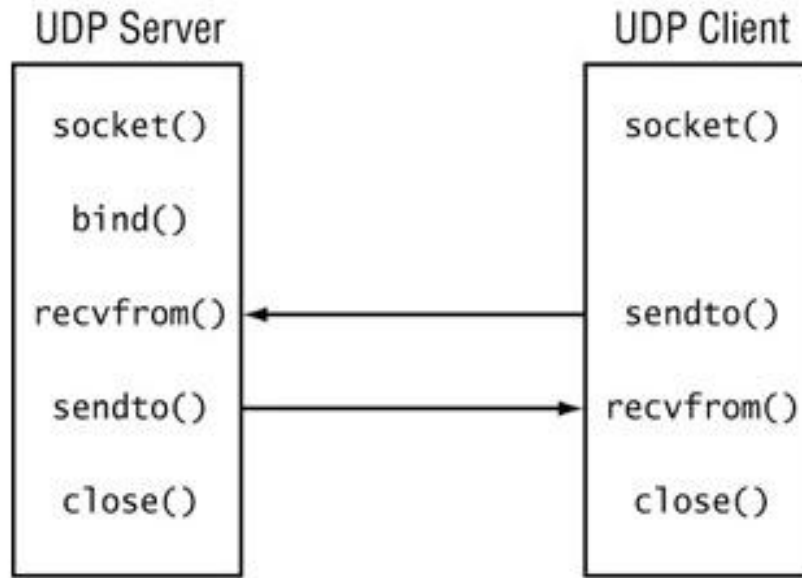
Disable The Cross Thread Exception Warning:

```
public Form1()
{
    InitializeComponent();
    Control.CheckForIllegalCrossThreadCalls = false;
}
```

لكن يبقى استخدام الطريقة الأولى أفضل لضمان استقرار النظام.

3.3: معمارية البرمجة المتزامنة في الـ UDP Transport Protocol:

إن معمارية الـ UDP بشكل عام هي أبسط من الـ TCP ففي الـ UDP سنحتاج فقط إلى تعريف الـ Socket في طرف المتصل ونبدأ الإرسال والاستقبال والسبب في ذلك أن الـ UDP لا يقوم بعملية إنشاء Session قبل عملية الإرسال كما هو الحال في الـ TCP ويكتفي الـ UDP بالإرسال إلى العنوان المحدد فقط والاستقبال كذلك وتعتبر هذه المعمارية أساسية في عملية الإرسال والاستقبال من خلال الـ UDP إذ لا يقوم الـ UDP بأي عملية تحقق من وصول الرسالة كذلك لا يقوم بعملية التحقق إذا كان الطرف المرسل إليه متصل بالشبكة أم لا ويبين الشكل 3.2 معمارية الاتصال من خلال الـ UDP.



الشكل 3.2 ويبين معمارية التراسل من خلال الـ UDP

في طرف المستضيف يتم تعريف الـ Socket ثم عمل Bind للـ Network IP والـ Transport Port Number الخاص بالمستضيف ثم تبدأ عملية الإرسال والاستقبال.

ما يميز الـ UDP Protocol غير السرعة هو إمكانية استخدامه لإرسال بيانات كـ Unicast و Multicast و Broadcast بعكس الـ TCP الذي يمكن استخدامه لتراسل كـ Unicast فقط والسبب في ذلك معمارية الـ UDP التي لا تحتوي على أي عمليات إنشاء اتصال مباشر بين المتصل والمستضيف. سنبيين في هذا الفصل كيفية الإرسال والاستقبال من خلال الـ UDP بثلاثة طرق الإرسال كـ Unicast و Multicast و Broadcast.

3.3.1: معمارية التراسل باستخدام الـ UDP كـ Unicast:

في عملية الإرسال أو الاستقبال نكتفي بتعريف الـ Socket وتحديد عنوان المستضيف وأخير إجراء عملية الإرسال باستخدام الدالة `SendTo` و الاستقبال باستخدام الدالة `ReceiveFrom` والتي يتم فيها إسناد عنوان المرسل إليه أو المستقبل منه.

UDP Socket – Unicast Client:

```
void ClientSender()
{
    // Create The UDP Socket
    Socket client_socket = new
    Socket(AddressFamily.InterNetwork, SocketType.Dgram,
    ProtocolType.Udp);
    // The Unicast IP End Point
    IPEndPoint ipend = new
    IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
    // Encoding
    byte[] buffer = UnicodeEncoding.Unicode.GetBytes("Hello
    Unicasting UDP World!");

    // Send the Text
    client_socket.SendTo(buffer, ipend);

    // Close The Socket
    client_socket.Close();
}

void ClientReceiver()
{
    // Create The UDP Socket
    Socket client_socket = new
    Socket(AddressFamily.InterNetwork, SocketType.Dgram,
    ProtocolType.Udp);
    // The Unicast IP End Point
    IPEndPoint ipend = new
    IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
    byte[] buffer = new byte[1024];
    EndPoint ip = (EndPoint)ipend;
    while (true)
    {
        client_socket.ReceiveFrom(buffer, ref ip);
        string ReceivedMessage =
        UnicodeEncoding.Unicode.GetString(buffer);
    }
}
```

في طرف المستضيف يتم تعريف الـ Socket ثم عمل Bind للـ IP Network والـ Transport Port Number الخاص بالمستضيف ثم تبدأ عملية الإرسال أو الاستقبال ويتم ذلك كما يلي:

UDP Socket – Unicast Server:

```
void UDP_Server()
{
    // Creation Socket
    Socket sock = new Socket(AddressFamily.InterNetwork,
    SocketType.Dgram, ProtocolType.Udp);

    // Socket Binding
    IPEndPoint ipend = new IPEndPoint(IPAddress.Any, 5000);
    sock.Bind(ipend);

    while (true)
    {
        // Receiving
        byte[] buffer = new byte[1024];
        EndPoint ip = (EndPoint)ipend;
        sock.ReceiveFrom(buffer, ref ip);
        // Encoding
        string ReceivedMessage =
        UnicodeEncoding.Unicode.GetString(buffer);
    }
}
```

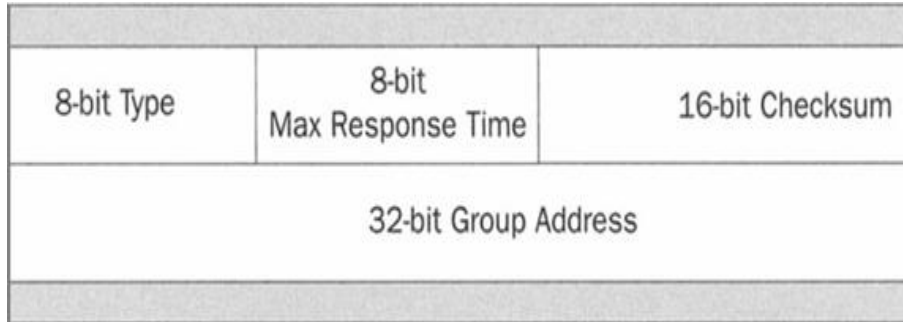
في العادة ما يتم استخدام الـ ReceiveFrom و الـ SendTo مع الـ UDP والسبب لأنه لا يوجد إعداد مسبق لعملية الاتصال كما هو الحال في الـ TCP حيث يتم فقط تعريف الـ Socket والإرسال أو الاستقبال مباشرة لذلك لا بد من إرسال عنوان المرسل إليه مع دالة الـ Send كما توفر لنا الدالة الـ ReceiveFrom إمكانية معرفة مرسل البيانات من خلال إسناد قيمة الـ EndPoint object كـ Reference Value إليها ويمكننا عرض عنوان صاحب الرسالة ورقم الـ Port الذي تم الإرسال من خلاله بتحويل قيمة الـ EndPoint إلى string كما يلي: `ip.ToString()`. كما يمكن أيضا استخدام ذلك العنوان للإرسال رد إلى المرسل وذلك بإسناد قيمة الـ EndPoint التي تم إسنادها من خلال الـ ReceiveFrom إلى الـ SendTo كمثال:

UDP Socket – Send a Replay By Using The EndPoint Object:

```
EndPoint ip = (EndPoint)ipend;
sock.ReceiveFrom(buffer, ref ip);
// To Send a Replay, For Example:
byte[] buffer =
UnicodeEncoding.Unicode.GetBytes("Received, Thanks.");
sock.SendTo(buffer, ip);
```

3.3.2: معمارية التراسل باستخدام الـ UDP كـ Multicast:

يعرف الـ Multicast على أنه الإرسال من شخص إلى مجموعة One-to-many باستخدام الـ Class D Subnet Mask في الـ IP Protocol وتتم عملية نقل البيانات من خلال الـ UDP Transport Protocol ويستخدم بروتوكول خاص لعملية إدارة المجموعات هو الـ IGMP – Internet Group Management Protocol والذي هو جزء من الـ Internet Protocol Model وكما يتضح من الشكل 3.3 فإن بروتوكول الـ IGMP يحتوي على عمليات التحقق من الوصول السليم للبيانات حيث يتم إرسال حجم البيانات الكلي لرسالة، و تحتوي أيضا على الـ Max response Time والذي يحدد فيه الفترة الزمنية للـ corresponding report، وكذلك نوع العملية (ضم إلى مجموعة Join to Group، الخروج من مجموعة Drop From Group، أو إرجاع معلومات عن المجموعة Membership Query) وأخيرا عنوان الـ IP للمجموعة.



الشكل 3.3 ويوضح تركيب بروتوكول الـ IGMP

تم تخصيص الـ Range في الـ IP Multicasting من 224.0.0.0 إلى 239.255.255.255 ونستطيع تحديده بثلاثة طرق إما بشكل يدوي Static أو Dynamic أو على أساس الـ Scope-Relative (لمزيد من المعلومات حول تخصيصات الـ Multicast Ranges انظر الرابط التالي:

<http://www.iana.org/assignments/multicast-addresses>

يتم نقل الـ Multicast Packets بين الـ Backbone Tunnels كـ Unicast حيث يتم إرسالها من داخل الشبكة إلى الـ Router و ترسل من Router إلى آخر عبر الـ Backbone Tunnel بأسلوب الـ Unicast وهو ما يوفر الكثير من الـ Bandwidth في الشبكة حيث ترسل نسخة واحدة إلى الـ Router ويقوم بتوزيعها على الأجهزة كـ Broadcast.

سنقوم في البداية بشرح كيفية التعامل مع العمليات الأساسية بالدوت نت والتي سنحتاجها للإرسال أو الاستقبال من الـ Multicast Group كالانضمام أو الخروج من مجموعة و الإرسال والاستقبال من مجموعة وشرح بعض الإعدادات الضرورية في عملية الإرسال كـ Multicasting.

3.3.2.1: الانضمام أو الخروج من مجموعة Join || Drop a multicast Group :

تتم عملية الانضمام إلى مجموعة Join a Group بالشكل التالي بعد تعريف الـ Socket:

UDP Socket – Join to Multicast Group:

```
// Socket Declaration
Socket MySocket = new Socket(AddressFamily.InterNetwork,
                             SocketType.Dgram, ProtocolType.Udp);

// IP Declaration
IPAddress Multicast_IP = IPAddress.Parse("224.0.0.1");

// Join to The Multicast Group Using The SocketOption
MySocket.SetSocketOption(SocketOptionLevel.IP,
                          SocketOptionName.AddMembership, new
                          MulticastOption(Multicast_IP));
```

حيث أن Multicasting هو جزء من الـ IP Infrastructure فإن التغيير في الـ Socket Option سيتم على مستوى الـ IP لذلك تم اختيار SocketOptionLevel.IP ، في الباروميتر الثاني حددنا نوع الإجراء الذي سيتم على مستوى الـ IP وقد قمنا بتعريف نوع الإجراء بـ AddMembership في الباروميتر الثالث يتم وضع القيمة الجديدة لتغيير المطلوب وقد قمنا بتمرير الـ IP Multicast وبنفس الطريقة يمكن حذف العضوية وذلك باختيار SocketOptionName.DropMembership في الـ SocketOption.

3.3.2.2: الإرسال و الاستقبال Sending And Receiving From a multicast Group :

قبل عملية الإرسال يفضل تحديد عدد الـ TTL – Time To Live والمقصود بها عدد الـ Routers التي يمكن أن تمر الـ Multicast Packets من خلالها حيث إذا تم وضع قيمة 1 فهذا يعني أن الـ Multicast Packets لن تخرج عن الـ Local Network والقيمة التي يفضل أن يتم وضعها هنا تعتمد على طبيعة الـ Infrastructure لشبكة التي سيعمل عليها البرنامج إذا كان من الضروري توسيع مجال الإرسال إلى شبكات أخرى ضمن الشبكة المحلية أم لا ويتم ذلك كما يلي كمثال:

UDP Socket – Set The Number Of TTL In a Multicasting Socket:

```
// Set The Number Of TTL in the SetSocketOption
MySocket.SetSocketOption(SocketOptionLevel.IP,
                          SocketOptionName.MulticastTimeToLive, 1);
```

أما عملية الإرسال فيمكن أن تتم بطريقتين الأولى من خلال الدالة Send حيث يلزم عمل Connct مع Multicast Group والثانية الإرسال مباشرة من خلال الدالة SendTo حيث يتم تمرير البيانات المراد إرسالها وعنوان Multicast Group الذي نريد أن نرسل إليه كما يلي كمثال:

UDP Socket – Sending In Multicasting:

```
Byte[] buffer =
    Encoding.Unicode.GetBytes("Hello My Group");

IPEndPoint IPEnd = new IPEndPoint(Multicast_IP,
    Port_Number);

// The First Way (Send After Joining)
MySocket.Connect(IPEnd);
MySocket.Send(buffer);

// The Seond Way (Send Directly)
MySocket.SendTo(buffer, IPEnd);
```

أما الاستقبال فيمكن أن يتم بطريقتين أيضا أما من خلال الـ Receive أو الـ ReceiveFrom ونستفيد من الأخيرة بإمكانية معرفة عنوان مرسل الرسالة.

UDP Socket – Receiving In Multicasting:

```
// Binding
IPEndPoint ipend = new IPEndPoint(IPAddress.Any,
    Port_Number);
MySocket.Bind(ipend);

// Join To The Multicast Group
IPAddress Multicast_Group = IPAddress.Parse("224.0.0.1");
MySocket.SetSocketOption(SocketOptionLevel.IP,
    SocketOptionName.AddMembership, Multicast_Group);

byte[] buffer = new byte[1024];
EndPoint ip = (EndPoint)ipend;

// The First Way if we need to know the sender address:
MySocket.ReceiveFrom(buffer, ref ip);
string ReceivedFrom = ip.ToString();

// The Second Way:
MySocket.Receive(buffer);

string ReceivedMessage =
    UnicodeEncoding.Unicode.GetString(buffer);
```

3.3.3: معمارية التراسل باستخدام الـ UDP كـ Broadcasting:

تعني عملية الإرسال كـ Broadcasting الإرسال إلى كل من هو على الشبكة بغض النظر عن الـ IP الذي يملكه المستقبل ولا تختلف عملية الإرسال كـ Broadcasting كثيرا عن الـ Unicasting حيث أن الاختلاف فقط بالسماح بتفعيل الـ Broadcasting على الـ Socket وتمرير عنوان الـ Broadcast إلى الـ IPEndPoint عند الإرسال أما الاستقبال فهو تماما كالاستقبال في الـ Uncasting.

UDP Socket – Sending as Broadcasting:

```
// Normal UDP Socket declaration
Socket MySocket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);

// Declare the EndPoint destination as Broadcast
IPEndPoint iep = new
IPEndPoint(IPAddress.Broadcast, Port_Number);

// Enable The Broadcasting at The Socket Level
MySocket.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.Broadcast, true);

// Send The Data Normally
byte[] data = Encoding.ASCII.GetBytes("Hello All");
MySocket.SendTo(data, iep);

// Finally Close The Socket
MySocket.Close();
```

في الـ IPEndPoint يمكن تمرير عنوان الـ Broadcast الخاص بشبكته بشكل يدوي كوضع العنوان "255.255.255.255" كمثال (لمزيد من المعلومات حول الـ Broadcast وكيفية حسابه أنظر الفصل الأول) أو يمرر بشكل آلي من خلال تمرير الـ IPAddress.Broadcast.

أما الدالة SetSocketOption فقد قمنا بتمرير قيمة الـ Socket Level في الباروميتر الأول وفي الثاني حددنا نوع العملية المطلوبة وهي تفعيل الـ Broadcast على الـ Socket و في الباروميتر الثالث مررنا قيمة true للسماح بالإرسال كـ Broadcast.

UDP Socket – Receiving In Broadcasting:

```
// Normal UDP Socket declaration
Socket MySocket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);

EndPoint iep = new IPEndPoint(IPAddress.Any,
Port_Number);
MySocket.Bind(iep);

EndPoint ep = (EndPoint) iep;

byte[] buffer = new byte[1024];
MySocket.ReceiveFrom(buffer, ref ep);

string ReceivedMessage =
Encoding.ASCII.GetString(buffer);

string ReceivedFrom = ep.ToString();
```

الخلاصة:

بينما في هذا الفصل طريقة التعامل مع الـ Synchronous Socket بجميع الطرق في الـ TCP والـ UDP ومعمارية الإرسال بالـ Socket كـ Unicast و Multicast وأخيرا كـ Broadcast كما وبيننا كيفية التحكم بخصائص الـ Socket من خلال الدالة SetSocketOption. سيتم الحديث في الفصل التالي عن الـ Asynchronous Socket وبرمجتها بجميع الطرق في الـ TCP والـ UDP.