

Despite what appear to be claims to the contrary, busy channel detection is a complicated process and current implementations have serious unacknowledged flaws when used in an ARQ system. It is not possible, without complicated auxiliary listening technologies, to determine if a wider bandwidth transition will cause interference with adjacent signals, so current implementations do not check **once a connection has been established**. It is the intention of the team developing ARDOP to implement the next best thing, which is to determine maximum currently available bandwidth as part of the initial connection negotiation. Another thing that hams tend to overlook is that ARQ modes usually do not have a means of decoding the existing traffic on a frequency, so users of the associated software do not know if the signals that are blocking their connection attempts are simply others using the same fixed station or one side of a QSO in progress, so any interference that results in a “busted QSO” is not necessarily intentional on the part of the users of automatic stations. They also may not have the time or power resources to spare to run the computer constantly and monitor with applications that can determine these things.

The following code sample is the actual VB.NET code used as the busy channel detector in ARDOP 0.7.2 up to 0.8.1. Similar code is used in WinMOR and other software modems written by KN6KB. I also have a C# version of the original VB.NET busy channel detector code from ARDOP posted on Github. With the release of ARDOP Chat 0.8.2 and ARDOP 0.8.2, we have added Busy Channel Blocking which ends a connection attempt if the receiving station has detected local channel activity that cannot be detected at the other end. Pactor does have a busy channel detector as well. Other modes can have identical functionality, but most developers don't seem to see the need to implement this due to dependence on use of visual guides that may not adequately prevent interference such as waterfall displays.

```
'Function to implement a busy detector based on 1024 point FFT
Public Function BusyDetect2(ByRef dblMag() As Double, intStart As Integer, intStop As Integer) As Boolean
    ' each bin is about 12000/1024 or 11.72 Hz
    ' this only called while searching for leader ...once leader detected, no longer called.
    ' First sort signals and look at high signals:baseline ratio..
    Dim dblAVGSignalPerBinNarrow, dblAVGSignalPerBinWide, dblAVGBaselineNarrow,
    dblAVGBaselineWide As Double
    Dim dblFastAlpha As Double = 0.3
    Dim dblSlowAlpha As Double = 0.1
    Dim dblAvgStoNNarrow, dblAvgStoNWide As Double
    Dim intNarrow As Int32 = 8 * 8 * 11.72 Hz about 94 z
    Dim intWide As Int32 = Round((intStop - intStart) * 0.66)
    Static dblAvgStoNSlowNarrow As Double
    Static dblAvgStoNFastNarrow As Double
    Static dblAvgStoNSlowWide As Double
    Static dblAvgStoNFastWide As Double
```

```

Static intLastStart As Integer = 0
Static intLastStop As Integer = 0

' First narrow band (~94Hz)
SortSignals(dblMag, intStart, intStop, intNarrow, dblAVGSignalPerBinNarrow,
dblAVGBaselineNarrow)
If intLastStart = intStart And intLastStop = intStop Then
    dblAvgStoNSlowNarrow = (1 - dblSlowAlpha) * dblAvgStoNSlowNarrow + dblSlowAlpha *
dblAVGSignalPerBinNarrow / dblAVGBaselineNarrow
    dblAvgStoNFastNarrow = (1 - dblFastAlpha) * dblAvgStoNFastNarrow + dblFastAlpha *
dblAVGSignalPerBinNarrow / dblAVGBaselineNarrow
Else
    dblAvgStoNSlowNarrow = dblAVGSignalPerBinNarrow / dblAVGBaselineNarrow
    dblAvgStoNFastNarrow = dblAVGSignalPerBinNarrow / dblAVGBaselineNarrow
    intLastStart = intStart : intLastStop = intStop
End If
dblAvgStoNNarrow = Max(dblAvgStoNSlowNarrow, dblAvgStoNFastNarrow) ' computes
fast attack, slow release

' Wide band (66% ofr current bandwidth)
SortSignals(dblMag, intStart, intStop, intWide, dblAVGSignalPerBinWide,
dblAVGBaselineWide)
If intLastStart = intStart And intLastStop = intStop Then
    dblAvgStoNSlowWide = (1 - dblSlowAlpha) * dblAvgStoNSlowWide + dblSlowAlpha *
dblAVGSignalPerBinWide / dblAVGBaselineWide
    dblAvgStoNFastWide = (1 - dblFastAlpha) * dblAvgStoNFastWide + dblFastAlpha *
dblAVGSignalPerBinWide / dblAVGBaselineWide
Else
    dblAvgStoNSlowWide = dblAVGSignalPerBinWide / dblAVGBaselineWide
    dblAvgStoNFastWide = dblAVGSignalPerBinWide / dblAVGBaselineWide
    intLastStart = intStart : intLastStop = intStop
End If
dblAvgStoNNarrow = Max(dblAvgStoNSlowNarrow, dblAvgStoNFastNarrow) ' computes
fast attack, slow release
dblAvgStoNWide = Max(dblAvgStoNSlowWide, dblAvgStoNFastWide) ' computes fast
attack, slow release
'Debug.WriteLine("[BusyDetect2: StoN Narrow = " & Format(dblAvgStoNNarrow, "#.0") & "
StoN Wide = " & Format(dblAvgStoNWide, "#.0"))

' Preliminary calibration...future a function of bandwidth and squelch.
Select Case MCB.ARQBandwidth
Case "200MAX", "200FORCED"
    If (dblAvgStoNNarrow > 1.4 * MCB.Squelch) Or (dblAvgStoNWide > 2 *

```

```

MCB.Squelch) Then Return True
    Case "500MAX", "500FORCED"
        If (dblAvgStoNNarrow > 1.4 * MCB.Squelch) Or (dblAvgStoNWide > 2 *
MCB.Squelch) Then Return True
        Case "1000MAX", "1000FORCED"
            If (dblAvgStoNNarrow > 1.4 * MCB.Squelch) Or (dblAvgStoNWide > 2 *
MCB.Squelch) Then Return True
            Case "2000MAX", "2000FORCED"
                If (dblAvgStoNNarrow > 1.4 * MCB.Squelch) Or (dblAvgStoNWide > 2 *
MCB.Squelch) Then Return True
                End Select
            Return False
        End Function ' BusyDetect2

```

```

Private Sub SortSignals(ByRef dblMag() As Double, intStartBin As Integer, intStopBin As
Integer, intNumBins As Integer, ByRef dblAVGSignalPerBin As Double, ByRef
dblAVGBaselinePerBin As Double)

```

```

    ' puts the top intNumber of bins between intStartBin and intStopBin into dblAVGSignal, the
rest into dblAVGBaseline

```

```

    ' for decent accuracy intNumBins should be < 75% of intStopBin-intStartBin)

```

```

    Dim dblAVGSignal(intNumBins - 1)

```

```

    Dim dblAVGBaseline(intStopBin - intStartBin - intNumBins - 1)

```

```

    Dim dblSigSum As Double

```

```

    Dim dblTotalSum As Double

```

```

    Dim intSigPtr As Integer = 0

```

```

    Dim intBasePtr As Integer = 0

```

```

    For i As Integer = 0 To intNumBins - 1

```

```

        For j As Integer = intStartBin To intStopBin

```

```

            If i = 0 Then

```

```

                dblTotalSum += dblMag(j)

```

```

                If dblMag(j) > dblAVGSignal(i) Then dblAVGSignal(i) = dblMag(j)

```

```

            Else

```

```

                If dblMag(j) > dblAVGSignal(i) And dblMag(j) < dblAVGSignal(i - 1) Then

```

```

                dblAVGSignal(i) = dblMag(j)

```

```

            End If

```

```

        Next j

```

```

    Next

```

```

    For k As Integer = 0 To intNumBins - 1

```

```

        dblSigSum += dblAVGSignal(k)

```

```

    Next

```

```

    dblAVGSignalPerBin = dblSigSum / intNumBins

```

```

    dblAVGBaselinePerBin = (dblTotalSum - dblSigSum) / (intStopBin - intStartBin -

```

```
intNumBins + 1)  
End Sub
```