UNIVERSIDAD CATÓLICA ANDRÉS BELLO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES

CÁLCULO DE CAPACIDAD DE UNA RED FIJA BASADA EN WIMAX IEEE 802.16d

APÉNDICES Y ANEXOS

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar al título de

INGENIERO EN TELECOMUNICACIONES

REALIZADO POR *Rosell Carolina Barrios Benitez*

PROFESOR GUIA *José Javier Barrios A.*

FECHA *24 de Septiembre de 2007*

APÉNDICE B

Código Fuente de la Herramienta

```
function varargout = modulo_presentacion(varargin)
% MODULO_PRESENTACION M-file for modulo_presentacion.fig
%MODULO_PRESENTACION, by itself, creates a new MODULO_PRESENTACION %or
%raises the existing
%singleton*.
%
% H = MODULO_PRESENTACION returns the handle to a new
%MODULO_PRESENTACION or the handle to the existing singleton*.
%
%MODULO_PRESENTACION('CALLBACK',hObject,eventData,handles,...) %calls the
%local function named CALLBACK in MODULO_PRESENTACION.M with the given
%input arguments.
%MODULO_PRESENTACION('Property','Value',...) creates a new
%MODULO_PRESENTACION or raises the existing singleton*.  Starting from
the left, property value pairs are applied to the GUI before
%modulo_presentacion_OpeningFunction gets %called.  An unrecognized
%property name or invalid value makes property %application
%stop.  All inputs are passed to modulo_presentacion_OpeningFcn via
%varargin.
%
% See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help modulo_presentacion

% Last Modified by GUIDE v2.5 12-Sep-2007 12:50:19

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @modulo_presentacion_OpeningFcn, ...
                   'gui_OutputFcn',  @modulo_presentacion_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before modulo_presentacion is made visible.
function modulo_presentacion_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to modulo_presentacion (see VARARGIN)
```

```
set(handles.axes3,'Xtick',[]);
set(handles.axes3,'Ytick',[]);


set(handles.Logo,'Visible','on');
[x,MAP]=imread('telecom.jpg');
axes(handles.Logo);
imshow(x,MAP);

set(handles.Logo,'Xtick',[]);
set(handles.Logo,'Ytick',[]);


set(handles.Logo2,'Visible','on');
[x,MAP]=imread('ucab.jpg');
axes(handles.Logo2);
imshow(x,MAP);

set(handles.Logo2,'Xtick',[]);
set(handles.Logo2,'Ytick',[]);


% Choose default command line output for modulo_presentacion
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes modulo_presentacion wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = modulo_presentacion_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in inicio.
function inicio_Callback(hObject, eventdata, handles)
% hObject    handle to inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all;
modulo_transmisor
```

```
% --- Executes on button press in ESCAPE.
function ESCAPE_Callback(hObject, eventdata, handles)
% hObject    handle to ESCAPE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all;

function varargout = modulo_transmisor(varargin)
% MODULO_TRANSMISOR M-file for modulo_transmisor.fig
%      MODULO_TRANSMISOR, by itself, creates a new MODULO_TRANSMISOR or
%      raises the existing singleton*.
%
%      H = MODULO_TRANSMISOR returns the handle to a new
%MODULO_TRANSMISOR or the handle to the existing singleton*.
%
%      MODULO_TRANSMISOR('CALLBACK',hObject,eventData,handles,...) calls
%      the local function named CALLBACK in MODULO_TRANSMISOR.M with the
%      given input arguments.
%
%      MODULO_TRANSMISOR('Property','Value',...) creates a new
%MODULO_TRANSMISOR or raises the
%      existing singleton*.  Starting from the left, property value pairs
%      are applied to the GUI before modulo_transmisor_OpeningFunction %
%      gets called.  An
%      unrecognized property name or invalid value makes property %   % %
%      application
%      stop.  All inputs are passed to modulo_transmisor_OpeningFcn via %
%      varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only %
%      one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help modulo_transmisor

% Last Modified by GUIDE v2.5 14-Sep-2007 09:15:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @modulo_transmisor_OpeningFcn, ...
                   'gui_OutputFcn',  @modulo_transmisor_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{i});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before modulo_transmisor is made visible.
function modulo_transmisor_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to modulo_transmisor (see VARARGIN)
 set(handles.diagrama,'Xtick',[]);
 set(handles.diagrama,'Ytick',[]);


set(handles.antena,'Visible','on');
[x,MAP]=imread('antenas.jpg');
axes(handles.antena);
imshow(x,MAP);

set(handles.antena,'Xtick',[]);
set(handles.antena,'Ytick',[]);

 % Choose default command line output for modulo_transmisor
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes modulo_transmisor wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = modulo_transmisor_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function editPOTENCIA_Callback(hObject, eventdata, handles)
% hObject    handle to editPOTENCIA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editPOTENCIA as text
```

```
%        str2double(get(hObject,'String')) returns contents of
editPOTENCIA as a double


% --- Executes during object creation, after setting all properties.
function editPOTENCIA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editPOTENCIA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editGANANCIA_Callback(hObject, eventdata, handles)
% hObject    handle to editGANANCIA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editGANANCIA as text
%        str2double(get(hObject,'String')) returns contents of
editGANANCIA as a double


% --- Executes during object creation, after setting all properties.
function editGANANCIA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editGANANCIA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editFIGURAruido_Callback(hObject, eventdata, handles)
% hObject    handle to editFIGURAruido (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editFIGURAruido as
text
```

```
%        str2double(get(hObject,'String')) returns contents of
editFIGURAruido as a double


% --- Executes during object creation, after setting all properties.
function editFIGURAruido_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editFIGURAruido (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editFRECUENCIA_Callback(hObject, eventdata, handles)
% hObject    handle to editFRECUENCIA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editFRECUENCIA as text
%        str2double(get(hObject,'String')) returns contents of
editFRECUENCIA as a double


% --- Executes during object creation, after setting all properties.
function editFRECUENCIA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editFRECUENCIA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a
double
```

```
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a
double
```

```
% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --------------------------------------------------------------------
function uipanel2_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


if (hObject==handles.uno)

    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('60g.bmp');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


elseif(hObject==handles.dos)

    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('90g.bmp');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


elseif(hObject==handles.tress)

    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('120g.jpg');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);
```

```
elseif(hObject==handles.cuatro)


    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('180g.jpg');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


else


    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('240g.bmp');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


end


% --- Executes on button press in salir.
function salir_Callback(hObject, eventdata, handles)
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 close all;


% --- Executes on button press in salvar.
function salvar_Callback(hObject, eventdata, handles)
% hObject    handle to salvar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

p=get(handles.salvar,'value');
if p==1,

    global P;
    P=str2double(get(handles.editPOTENCIA,'String'));
    P=P*10^-3;
    P

    global Gt;
```

```
    Gt=str2double(get(handles.editGANANCIA,'String'));
    Gt

    global NF;
    NF=str2double(get(handles.editFIGURAruido,'String'));
    NF


     global Trafico;
    Trafico=str2double(get(handles.edit8,'String'));
    Trafico

    global F;
    F=str2double(get(handles.editFRECUENCIA,'String'));
    F

    global D;
    D=str2double(get(handles.edit10,'String'));
    D


end


% --- Executes on button press in continuar.
function continuar_Callback(hObject, eventdata, handles)
% hObject    handle to continuar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all;
modulo_receptor


 % --- Executes on button press in anterior.
function anterior_Callback(hObject, eventdata, handles)
% hObject    handle to anterior (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all;
modulo_presentacion



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as
a double
```

```
% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function varargout = modulo_receptor(varargin)
% MODULO_RECEPTOR M-file for modulo_receptor.fig
%      MODULO_RECEPTOR, by itself, creates a new MODULO_RECEPTOR or
raises the existing
%      singleton*.
%
%      H = MODULO_RECEPTOR returns the handle to a new MODULO_RECEPTOR or
the handle to
%      the existing singleton*.
%
%      MODULO_RECEPTOR('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in MODULO_RECEPTOR.M with the given input
arguments.
%
%      MODULO_RECEPTOR('Property','Value',...) creates a new
MODULO_RECEPTOR or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before modulo_receptor_OpeningFunction gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to modulo_receptor_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help modulo_receptor

% Last Modified by GUIDE v2.5 12-Sep-2007 17:31:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @modulo_receptor_OpeningFcn, ...
                   'gui_OutputFcn',  @modulo_receptor_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before modulo_receptor is made visible.
function modulo_receptor_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to modulo_receptor (see VARARGIN)

set(handles.diagrama,'Xtick',[]);
 set(handles.diagrama,'Ytick',[]);


set(handles.antena,'Visible','on');
[x,MAP]=imread('antena2.jpg');
axes(handles.antena);
imshow(x,MAP);

set(handles.antena,'Xtick',[]);
set(handles.antena,'Ytick',[]);

% Choose default command line output for modulo_receptor
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes modulo_receptor wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = modulo_receptor_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function ganancia_Callback(hObject, eventdata, handles)
% hObject    handle to ganancia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ganancia as text
%        str2double(get(hObject,'String')) returns contents of ganancia
as a double


% --- Executes during object creation, after setting all properties.
function ganancia_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ganancia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function perdidas_Callback(hObject, eventdata, handles)
% hObject    handle to perdidas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of perdidas as text
%        str2double(get(hObject,'String')) returns contents of perdidas
as a double


% --- Executes during object creation, after setting all properties.
function perdidas_CreateFcn(hObject, eventdata, handles)
% hObject    handle to perdidas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in salir.
function salir_Callback(hObject, eventdata, handles)
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all

% --- Executes on button press in salvar.
function salvar_Callback(hObject, eventdata, handles)
% hObject    handle to salvar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

p=get(handles.salvar,'value');
if p==1,

    global Gr;
    Gr=str2double(get(handles.ganancia,'String'));
    Gr

    global Ltr;
    Ltr=str2double(get(handles.perdidas,'String'));
    Ltr

    global BWrx;
    BWrx=str2double(get(handles.edit7,'String'));
    BWrx


end

% --- Executes on button press in continuar.
function continuar_Callback(hObject, eventdata, handles)
% hObject    handle to continuar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close all;
modulo_propagacion

% --- Executes on button press in anterior.
function anterior_Callback(hObject, eventdata, handles)
% hObject    handle to anterior (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close all;
modulo_transmisor
```

```
% --------------------------------------------------------------------
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (hObject==handles.uno)

    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('60g.bmp');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


elseif(hObject==handles.dos)

    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('90g.bmp');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


elseif(hObject==handles.tres)

    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('120g.jpg');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


elseif(hObject==handles.cuatro)


    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('180g.jpg');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);

else
```

```
    set(handles.diagrama,'Visible','on');
    [x,MAP]=imread('240g.bmp');
    axes(handles.diagrama);
    imshow(x,MAP);

    set(handles.diagrama,'Xtick',[]);
    set(handles.diagrama,'Ytick',[]);


end


function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function varargout = modulo_propagacion(varargin)
% MODULO_PROPAGACION M-file for modulo_propagacion.fig
%      MODULO_PROPAGACION, by itself, creates a new MODULO_PROPAGACION or
%      raises the existing singleton*.
%
%      H = MODULO_PROPAGACION returns the handle to a new
%MODULO_PROPAGACION or the handle to the existing singleton*.
%
%      MODULO_PROPAGACION('CALLBACK',hObject,eventData,handles,...) calls
%      the local function named CALLBACK in MODULO_PROPAGACION.M with the
%      given input arguments.
%
%      MODULO_PROPAGACION('Property','Value',...) creates a new
MODULO_PROPAGACION or raises the existing singleton*.  Starting from the
%      left, property value pairs are applied to the GUI before % % % %
%      modulo_propagacion_OpeningFunction gets called.  An
```

```
%      unrecognized property name or invalid value makes property  % % %
%      application
%      stop.  All inputs are passed to modulo_propagacion_OpeningFcn via
%      varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only %
%      one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help modulo_propagacion

% Last Modified by GUIDE v2.5 14-Sep-2007 21:57:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @modulo_propagacion_OpeningFcn, ...
                   'gui_OutputFcn',  @modulo_propagacion_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before modulo_propagacion is made visible.
function modulo_propagacion_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to modulo_propagacion (see VARARGIN)

 set(handles.axes1,'Xtick',[]);
 set(handles.axes1,'Ytick',[]);

 set(handles.axes2,'Xtick',[]);
 set(handles.axes2,'Ytick',[]);

set(handles.axes1,'Visible','on');
[x,MAP]=imread('timotes7.jpg');
axes(handles.axes1);
imshow(x,MAP);

 set(handles.edit7,'Visible','off');
```

```
set(handles.text32,'Visible','off');

 M = moviein(50);
                        t=1/8000:1/8000:4;
                        for j=1:100,
                        amplitud=20;
                        x=amplitud*sin(2*(pi/4)*10*(t+j));
                        y=amplitud*cos(2*(pi/4)*10*(t+j));
                        set(handles.axes2,'Visible','on');
                        axes(handles.axes2);
                        plot(t,x,'g');
                        hold on
                        plot(t,y,'m');
                        hold off
                        grid
                        AXIS([0 1 min(x) max(x)])
                        M(:,j) = getframe;
                        end


set(handles.salvar2,'Visible','off');
set(handles.text3,'Visible','off');
set(handles.text5,'Visible','off');
set(handles.text7,'Visible','off');
set(handles.text9,'Visible','off');
set(handles.text11,'Visible','off');
set(handles.text13,'Visible','off');

% Choose default command line output for modulo_propagacion
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes modulo_propagacion wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = modulo_propagacion_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in salvar.
function salvar_Callback(hObject, eventdata, handles)
% hObject    handle to salvar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
p=get(handles.salvar,'value');
if p==1,

    global fc;
    fc=str2double(get(handles.fc,'String'));
    fc

    global hBS;
    hBS=str2double(get(handles.hBS,'String'));
    hBS

    global hMS;
    hMS=str2double(get(handles.hMS,'String'));
    hMS


     global d;
    d=str2double(get(handles.d,'String'));
    d


end

% --------------------------------------------------------------------
function panelESCENARIO_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to panelESCENARIO (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if (hObject==handles.uno)

    set(handles.text3,'Visible','on');
    set(handles.boton1,'Visible','on');
    set(handles.PEL,'Visible','on');

    set(handles.text5,'Visible','off');
    set(handles.text7,'Visible','off');
    set(handles.text9,'Visible','off');
    set(handles.text11,'Visible','off');
    set(handles.text13,'Visible','off');


end
if(hObject==handles.dos)

    set(handles.boton2,'Visible','on');
    set(handles.boton3,'Visible','on');
    set(handles.boton4,'Visible','on');
    set(handles.boton5,'Visible','on');
    set(handles.text5,'Visible','on');
    set(handles.text6,'Visible','on');
    set(handles.text7,'Visible','on');
    set(handles.text8,'Visible','on');
```

```
    set(handles.text9,'Visible','on');
    set(handles.text10,'Visible','on');
    set(handles.text11,'Visible','on');
    set(handles.text12,'Visible','on');

    set(handles.text3,'Visible','off');
    set(handles.text13,'Visible','off');

end
 if(hObject==handles.tres)


    set(handles.boton6,'Visible','on');
    set(handles.text13,'Visible','on');
    set(handles.text14,'Visible','on');

    set(handles.text3,'Visible','off');
    set(handles.text5,'Visible','off');
    set(handles.text7,'Visible','off');
    set(handles.text9,'Visible','off');
    set(handles.text11,'Visible','off');

 end

% --- Executes on button press in boton1.
function boton1_Callback(hObject, eventdata, handles)
% hObject    handle to boton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


global P;
X=1*10^-3;
S=P/X;
PTm=10*log10(S);
PT=10*log10(P);

global D;
distancia=20*log10(D);

global F;
f=20*log10(F);
global PERdIDAS

PERdIDAS = 92.5 + distancia + f;
set(handles.PEL,'String',PERdIDAS);



% --- Executes on button press in boton2.
function boton2_Callback(hObject, eventdata, handles)
% hObject    handle to boton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global fc;
global d;
d
fc
global PL_COST231_WI;
PL_COST231_WI=42.6+26*log10(d)+20*log10(fc)
set(handles.text6,'String',PL_COST231_WI);


% --- Executes on button press in salvar2.
function salvar2_Callback(hObject, eventdata, handles)
% hObject    handle to salvar2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global hroof;
hroof=str2double(get(handles.edit7,'String'));
set(handles.edit7,'Visible','off');
set(handles.text32,'Visible','off');
set(handles.salvar2,'Visible','off');



% --- Executes on button press in boton3.
function boton3_Callback(hObject, eventdata, handles)
% hObject    handle to boton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global d;
global fc;
global hBS;

C=menu('El Modelo Xia tiene tres modalidades, seleccione una por
favor','ANTENAS DE ESTACION BASE PROXIMAS AL NIVEL MEDIO DE LOS
EDIFICIOS','ANTENAS DE ESTACION BASE POR ENCIMA DEL NIVEL MEDIO DE LOS
EDIFICIOS', 'ANTENAS DE ESTACION BASE POR DEBAJO DEL NIVEL MEDIO DE LOS
EDIFICIOS');
modelo= [1,2,3];
M=modelo(C);

if M==1

    global PL_XIA;
    PL_XIA=40*log10(d)+30*log10(fc)+49 %en dB

    set(handles.text8,'String',PL_XIA);

elseif M==2

    set(handles.salvar2,'Visible','on');
    set(handles.text32,'Visible','on');
    set(handles.edit7,'Visible','on');
    global hroof;
    global PL_XIA;
    PL_XIA=38*log10(d)-18*log10(hBS-hroof)+21*log10(fc)+81.5 %en dB
```

```
    set(handles.text8,'String',PL_XIA);

else
    global PL_XIA;
    PL_XIA=40*log10(d)-40*log10(fc)+35 %en
    set(handles.text8,'String',PL_XIA);
end


% --- Executes on button press in boton4.
function boton4_Callback(hObject, eventdata, handles)
% hObject    handle to boton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global hBS;
global hMS;
global fc;
global d;

FC=fc*10^6;
if FC<=200000000
    a1(hMS)=[8.29*((log10(1.54*hMS))*(log10(1.54*hMS)))]-1.1
else
   a1(hMS)=[3.2*((log10(11.75*hMS))*(log10(11.75*hMS)))]-4.97
end

A1=46.3+33.9*log10(fc)-13.82*log10(hBS)+a1(hMS);

B=44.9-6.55*log10(hBS);
C=menu('Para medios con alta densidad de arboles marque 0, para centros
metropolitanos marque 3:','C = 0','C = 3');
centro= [0,3];
Centro=centro(C);
C

global PL_COST231_HATA1;
PL_COST231_HATA1=A1+B*log10(d)+C

set(handles.text10,'String',PL_COST231_HATA1);


% --- Executes on button press in boton5.
function boton5_Callback(hObject, eventdata, handles)
% hObject    handle to boton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global hBS;
global hMS;
global fc;
global d;

if fc<=200000000
    a1(hMS)=[8.29*((log10(1.54*hMS))*(log10(1.54*hMS)))]-1.1
else
```

```
   a1(hMS)=[3.2*((log10(11.75*hMS))*(log10(11.75*hMS)))-4.97
end


C=menu('Para este tipo de Escenarios el Modelo Okumura-Hata tiene dos
modalidades, seleccione una por favor','Modelo Urbano','Modelo
SubUrbano');
modelo= [1,2];
M=modelo(C);

if M==1

    global PL_HATA_TU1;
    PL_HATA_TU1=69.55+26.16*log10(fc)-13.82*log10(hBS)-a1(hMS)+[44.9-
6.55*log10(hBS)]*log10(d)% EN dB
    set(handles.text12,'String',PL_HATA_TU1);

else

    PL_HATA_TU1=69.55+26.16*log10(fc)-13.82*log10(hBS)-a1(hMS)+[44.9-
6.55*log10(hBS)]*log10(d)% EN dB
    global PL_HATA_TS1;
    PL_HATA_TS1=PL_HATA_TU1-2*[(log10(fc/28))*(log10(fc/28))]-5.4% en dB

    set(handles.text12,'String',PL_HATA_TS1);

end

% --- Executes on button press in boton6.
function boton6_Callback(hObject, eventdata, handles)
% hObject    handle to boton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global hBS;
global hMS;
global fc;
global d;

if fc<=200000000
    a1(hMS)=[8.29*((log10(1.54*hMS))*(log10(1.54*hMS)))]-1.1
else
   a1(hMS)=[3.2*((log10(11.75*hMS))*(log10(11.75*hMS)))]-4.97
end

    PL_HATA_TU1=69.55+26.16*log10(fc)-13.82*log10(hBS)-a1(hMS)+[44.9-
6.55*log10(hBS)]*log10(d)% EN dB
    global PL_HATA_RU1;
    PL_HATA_RU1=PL_HATA_TU1-
4.78*[(log10(fc))*(log10(fc))]+18.33*log10(fc)-40.94 % en dB

    set(handles.text14,'String',PL_HATA_RU1);


function fc_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to fc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of fc as text
%        str2double(get(hObject,'String')) returns contents of fc as a
double


% --- Executes during object creation, after setting all properties.
function fc_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function hBS_Callback(hObject, eventdata, handles)
% hObject    handle to hBS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of hBS as text
%        str2double(get(hObject,'String')) returns contents of hBS as a
double


% --- Executes during object creation, after setting all properties.
function hBS_CreateFcn(hObject, eventdata, handles)
% hObject    handle to hBS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function hMS_Callback(hObject, eventdata, handles)
% hObject    handle to hMS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of hMS as text
%        str2double(get(hObject,'String')) returns contents of hMS as a
double


% --- Executes during object creation, after setting all properties.
function hMS_CreateFcn(hObject, eventdata, handles)
% hObject    handle to hMS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function d_Callback(hObject, eventdata, handles)
% hObject    handle to d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of d as text
%        str2double(get(hObject,'String')) returns contents of d as a
double


% --- Executes during object creation, after setting all properties.
function d_CreateFcn(hObject, eventdata, handles)
% hObject    handle to d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in continuar.
function continuar_Callback(hObject, eventdata, handles)
% hObject    handle to continuar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close all;
modulo_resultados
```

```
% --- Executes on button press in salir.
function salir_Callback(hObject, eventdata, handles)
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close all;

% --- Executes on button press in anterior.
function anterior_Callback(hObject, eventdata, handles)
% hObject    handle to anterior (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all;
modulo_receptor


function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function entornos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entornos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


% --- Executes on selection change in entorno.
function entorno_Callback(hObject, eventdata, handles)
% hObject    handle to entorno (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns entorno contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
entorno
switch entorno

    case 1

        set(handles.text3,'Visible','on');
        set(handles.boton1,'Visible','on');
        set(handles.PEL,'Visible','on');

    case 2
        set(handles.boton2,'Visible','on');
        set(handles.boton3,'Visible','on');
        set(handles.boton4,'Visible','on');

        set(handles.text5,'Visible','on');
        set(handles.text6,'Visible','on');
        set(handles.text7,'Visible','on');
        set(handles.text8,'Visible','on');
        set(handles.text9,'Visible','on');
        set(handles.text10,'Visible','on');


    case 3
        set(handles.text11,'Visible','on');
        set(handles.text12,'Visible','on');
        set(handles.boton5,'Visible','on');

    case 4
        set(handles.boton6,'Visible','on');
        set(handles.text13,'Visible','on');
        set(handles.text14,'Visible','on');

end

% --- Executes during object creation, after setting all properties.
function entorno_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entorno (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in modelos.
function modelos_Callback(hObject, eventdata, handles)
% hObject    handle to modelos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns modelos contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
modelos


% --- Executes during object creation, after setting all properties.
function modelos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to modelos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function varargout = modulo_resultados(varargin)
% MODULO_RESULTADOS M-file for modulo_resultados.fig
%      MODULO_RESULTADOS, by itself, creates a new MODULO_RESULTADOS or %
%      raises the existing singleton*.
%
%      H = MODULO_RESULTADOS returns the handle to a new
%MODULO_RESULTADOS or the handle to the existing singleton*.
%
%      MODULO_RESULTADOS('CALLBACK',hObject,eventData,handles,...) calls
%      the local
%      function named CALLBACK in MODULO_RESULTADOS.M with the given %
%      input arguments.
%
%      MODULO_RESULTADOS('Property','Value',...) creates a new
MODULO_RESULTADOS or raises the existing singleton*.  Starting from the %
%      left, property value pairs are
%      applied to the GUI before modulo_resultados_OpeningFunction gets %
%      called.  An unrecognized property name or invalid value makes %  %
%      property application
```

```
%      stop.  All inputs are passed to modulo_resultados_OpeningFcn via %
%      varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only %
%      one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help modulo_resultados

% Last Modified by GUIDE v2.5 20-Sep-2007 18:48:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @modulo_resultados_OpeningFcn, ...
                   'gui_OutputFcn',  @modulo_resultados_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before modulo_resultados is made visible.
function modulo_resultados_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to modulo_resultados (see VARARGIN)

set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);


set(handles.axes1,'Visible','on');
[x,MAP]=imread('resultados.jpg');
axes(handles.axes1);
imshow(x,MAP);


% Choose default command line output for modulo_resultados
handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes modulo_resultados wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = modulo_resultados_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in boton1.
function boton1_Callback(hObject, eventdata, handles)
% hObject    handle to boton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PEL;
global D;
global P;
global Gt;
global Gr;
global F;
global Ltr;
global NF;
global BWrx;
X=1*10^-3;
S=P/X;
PTm=10*log10(S)
PT=10*log10(P)
Ltt=2;
Lint=0;
PR= PTm - Ltt + Gt - PEL + Gr - Lint - Ltr;
BWC=BWrx*1000000;
Nrx= -174 + NF +10*log10(BWC);
global SNR;
SNR= PR-Nrx
if SNR<-0
    SNR=SNR*-1;
end
set(handles.snr2,'String',SNR);


 % --- Executes on button press in boton2.
function boton2_Callback(hObject, eventdata, handles)
% hObject    handle to boton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
M=menu('Desea mirar el patron de Probabilidad de Error de las
modulaciones usadas por WIMAX?','SI','NO');
mostrar= [1,2];
MOSTRAR=mostrar(M);

if MOSTRAR==1

   mostrar_pe

elseif MOSTRAR==2


        global SNR;
        if SNR>=6.4 & SNR<9.4

        set(handles.modul,'String','BPSK');
        else
        if SNR>=9.4 & SNR<16.4
            set(handles.modul,'String','QPSK');
        else
        if SNR>=16.4 & SNR<22.7
            set(handles.modul,'String','16-QAM');
        else
        if SNR>=22.7
            set(handles.modul,'String','64-QAM');
        end
        end
        end
        end


end

global SNR;
if SNR>=6.4 & SNR<9.4

     set(handles.modul,'String','BPSK');
   else
    if SNR>=9.4 & SNR<16.4
    set(handles.modul,'String','QPSK');
else
    if SNR>=16.4 & SNR<22.7
    set(handles.modul,'String','16-QAM');
else
    if SNR>=22.7
    set(handles.modul,'String','64-QAM');
   end
   end
   end
end
global y;
y=SNR;

% --- Executes on button press in boton3.
function boton3_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to boton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global D;
global P;
global Gt;
global Gr;
global F;
global Ltr;
global NF;
global BWrx;
global PEL;
global SNR;
Ltt=2;
Lint=0;

if   SNR>=6.4 & SNR<9.4
global TASA;
    fprintf('WIMAX UTILIZA MODULACION ADAPTATIVA Y DADO EL VALOR DE LA
SNR \n');
    fprintf('LA MODULACION CORRESPONDIENTE ES: \n\n');
    fprintf('BPSK \n\n');
    fprintf('LA TASA DE CODIFICACION USADA CON BPSK ES DE: \n\n');
    fprintf('1/2 \n\n');
    TASA=1/2;
else
    if SNR>=9.4 & SNR<16.4
    global TASA;
    fprintf('WIMAX UTILIZA MODULACION ADAPTATIVA Y DADO EL VALOR DE LA
SNR \n');
    fprintf('LA MODULACION CORRESPONDIENTE ES: \n\n');
    fprintf('QPSK \n\n');
    fprintf('A CONTINUACION ESCOJA LA TASA DE CODIFICACION \n');
    T= menu('Para QPSK WIMAX posee 2 tasas de Codificacion de Canal,
Escoja una:','t = 1/2','t = 3/4');
    tasa = [1/2,3/4];
    TASA=tasa(T); % ES NECESARIO PARA EL CALCULO DE EL THROUGHPUT !!!!!
else
    if SNR>=16.4 & SNR<22.7
    global TASA;
    fprintf('WIMAX UTILIZA MODULACION ADAPTATIVA Y DADO EL VALOR DE LA
SNR \n');
    fprintf('LA MODULACION CORRESPONDIENTE ES: \n\n');
    fprintf('16-QAM \n\n');
    fprintf('A CONTINUACION ESCOJA LA TASA DE CODIFICACION \n');
    T= menu('Para 16-QAM WIMAX posee 2 tasas de Codificacion de Canal,
Escoja una:','t = 1/2','t = 3/4');
    tasa = [1/2,3/4];
    TASA=tasa(T); % ES NECESARIO PARA EL CALCULO DE EL THROUGHPUT !!!!!
else
    if SNR>=22.7
    global TASA;
    fprintf('WIMAX UTILIZA MODULACION ADAPTATIVA Y DADO EL VALOR DE LA
SNR \n');
    fprintf('LA MODULACION CORRESPONDIENTE ES: \n\n');
    fprintf('64-QAM \n\n');
    fprintf('A CONTINUACION ESCOJA LA TASA DE CODIFICACION \n');
```

```
    T= menu('Para 64-QAM WIMAX posee 2 tasas de Codificacion de Canal,
Escoja una:','t = 2/3','t = 3/4');
    tasa = [2/3,3/4];
    TASA=tasa(T); % LO DEBO USAR PARA EL THROUGHPUT !!!!!
   end
   end
   end
end

set(handles.tasacodif,'String',TASA);


Trafico=str2double(get(handles.edit10,'String'));
Througput = (Trafico*(1+0.2))/(0.9*TASA)


 % --- Executes on button press in boton4.
function boton4_Callback(hObject, eventdata, handles)
% hObject    handle to boton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global D;
global P;
global Gt;
global Gr;
global F;
global Ltr;
global NF;
global BWrx;
global PEL;
global SNR;
global SNR;
Ltt=2;
Lint=0;
global C;
BWC2=BWrx*1000000;
C=2*BWC2*log2(1+SNR)

set(handles.capacidad2,'String',C);

% --- Executes on button press in boton5.
function boton5_Callback(hObject, eventdata, handles)
% hObject    handle to boton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Trafico;
global TASA;
global Througput2;
Througput = (Trafico*(1+0.2))/(0.9*TASA);
Througput2=Througput/10;
set(handles.trougput,'String',Througput2);


% --- Executes on button press in continuar.
function continuar_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to continuar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close all;
SIMULACION1

% --- Executes on button press in boton6.
function
% hObject    handle to boton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

      'WIMAX puede usar dos tipos de capas fisicas. Escoja una para
continuar:','Capa fisica de tipo OFDM','Capa fisica de tipo OFDMA');
 capa= [1,2];
 CAPA=capa(C);

 if CAPA==1

    global
    BWC2=BWrx*1000000;
    NutilFFT=196;
    NFFT=256;
    Tg=0.0114;
    bn=1;

    f=(8*BWC2)/7;
    N=NFFT+(NFFT/Tg);
    R=1/N;

    global
    TB=f*bn*Cr*NutilFFT*R; % tasa binaria
                        'String'


 else CAPA==2
    global
    BWC2=BWrx*1000000;
    NutilFFT=1536;
    NFFT=2048;
    Tg=0.0114;
    bn=1;

    f=(8*BWC2)/7;
    N=NFFT+(NFFT/Tg);
    R=1/N;

    global
    TB=f*bn*Cr*NutilFFT*R; % tasa binaria
                        'String'


 end
```

```
% --- Executes on button press in boton7.
function
% hObject    handle to boton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global
global
global


m=35;
                          % en MHz
                          % en microsegundos
BWtotal=Frecuencia_muestreo*Tiempo_util_simbolo


global
NU=(m*(BWtotal-2*BWguard)/BWc)
                                    'String',nu);
% --------------------------------------------------------------------
function
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


    if


        global
        global
        PEL = PERdIDAS;
        global
                'Modelo de Propagacion en el Espacio Libre'

    end
    if


        global
        global
        PEL=PL_COST231_WI;
        global
                'Modelo Cost231-Walfisch-Ikegami'

    end

    if


        global
        global
        PEL=PL_XIA;
```

```
        global
        'Modelo Xia'

    end

    if

        global
        global
        PEL-PL COST231 HATA1;
        global
            'Modelo Cost231-Hata'

    end

    if

        global
        global
        PEL-PL HATA TU1;
        global
            'Modelo Empirico Okumura-Hata - Urbano'

    end

    if

        global
        global
        PEL-PL HATA TS1;
        global
            'Modelo Empirico Okumura-Hata - Suburbano'

    end

    if

        global
        global
        PEL-PL HATA RU1;
        global
            'Modelo Empirico Okumura-Hata - Rural'

    end

% --- Executes on button press in salvar.
function
% hObject    handle to salvar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global



%--- Executes on button press in pushbutton11.
function
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    all

% --- Executes on button press in anterior.
function
% hObject    handle to anterior (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    all
modulo_propagacion



% --- Executes on button press in boton8.
function
% hObject    handle to boton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global
global
global

m=30;
BWc=BWguard/1000;
global
ND=(m*(BWtotal-2*BWguard)/BWc);
ND=ND/10^6;
                'String'



% --- Executes on button press in reporte.
function
% hObject    handle to reporte (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

                'value'
if
    global
    global
    global
```

```
    global
    global
    global
    global
    global
    global
    global
    global
    global
    global
    global
    global
    global
            'REPORTE.txt'  'wt'
            ' UNIVERSIDAD CATÓLICA ANDRÉS BELLO \n'
            ' FACULTAD DE INGENIERIA \n'
            ' ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES \n\n\n'
            '\b\b\b HERRAMIENTA DE CÁLCULO DE CAPACIDAD DE UNA RED FIJA
\n'
            '\b\b\b BASADA EN WIMAX \n\n\n\'
            '\b\b REPORTE DE RESULTADOS: \n\n'
            '\b\b DATOS DEL ENLACE: \n'
            '\b\b Distancia del Enlace (Km): %6.2f\n'
            '\b\b Frecuencia del Enlace (GHz): %6.2f\n'
            '\b\b Ganancia de la Antena Tx (dBi): %6.2f\n'
            '\b\b Ganancia de la Antena Rx (dBi): %6.2f\n'
            '\b\b Potencia de Tx (mW): %6.2f\n'
            '\b\b Modelo de Propagación seleccionado: %s\n\n'
            '\b\b CÁLCULOS DEL ENLACE: \n'
            '\b\b Pérdidas Basicas calculadas (dB): %e\n'
            '\b\b Relación Señal a Ruido (dB): %e\n'
            '\b\b CAPACIDAD MÁXIMA DEL ENLACE (bps): %e\n'
            '\b\b THROUGHPUT (bps): %e\n'
            '\b\b TASA BINARIA (bps): %e\n'
            '\b\b NUMERO DE CANALES EN EL UPLINK: %e\n'
            '\b\b NUMERO DE CANALES EN EL DOWNLINK: %e\n\n\n\n\n\n'
            '\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b Profesor Guía: José Javier
Barrios\n\n'
            '\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b Herramienta Elaborada por: Roseli
Carolina Barrios Benitez\n\n'
fclose(fid)
end

function
% MOSTRAR_PE M-file for mostrar_pe.fig
%      MOSTRAR_PE, by itself, creates a new MOSTRAR_PE or raises the %
%      existing singleton*.
%
%      H = MOSTRAR_PE returns the handle to a new MOSTRAR_PE or the %  %
%      handle to the existing singleton*.
%
%      MOSTRAR_PE('CALLBACK',hObject,eventData,handles,...) calls the % %
%      local function named CALLBACK in MOSTRAR_PE.M with the given input
%      arguments.
%
```

```
%      MOSTRAR_PE('Property','Value',...) creates a new MOSTRAR_PE or % %
%      raises the
%      existing singleton*.  Starting from the left, property value pairs
%      are applied to the GUI before mostrar_pe_OpeningFunction gets %  %
%      called.  An unrecognized property name or invalid value makes % %
%      property application
%      stop.  All inputs are passed to mostrar_pe_OpeningFcn via % % % %
%      varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only %
%      one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help mostrar_pe

% Last Modified by GUIDE v2.5 21-Sep-2007 22:49:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
                  'gui_Name'                    ...
                  'gui_Singleton'                  ...
                  'gui_OpeningFcn'                          ...
                  'gui_OutputFcn'                          ...
                  'gui_LayoutFcn'          ...
                  'gui_Callback'
if
    gui_State.gui_Callback = str2func(varargin{1});
end

if
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before mostrar_pe is made visible.
function
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to mostrar_pe (see VARARGIN)

                  'Xtick'
                  'Ytick'

   EbNoVec = [4:20];
     %QPSK
     for
         M=4;
```

```
    end
    axes(handles.axes1)
    grid
        'E/N (dB)'                'PE'
    axes(handles.axes2)
    grid
     'E/N (dB)'                'PE'

                    'string' 'Pe = 2Q(raiz(d*d/eta))'

on

%QPSK
    for
        M=2;
        end

                        '-ro'

for
        M=4;
        end

                        '-bo'

    for
        M=8;
        end
                        '-go'
     for
        M=16;
        end
                        '-co'

                'QPSK M=2' 'PSK M=4' 'PSK M=8' 'PSK M=16'

    off
```

```
% Choose default command line output for mostrar_pe
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes mostrar_pe wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = mostrar_pe_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in regresar.
function regresar_Callback(hObject, eventdata, handles)
% hObject    handle to regresar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close mostrar_pe


% --- Executes on button press in continuar.
function continuar_Callback(hObject, eventdata, handles)
% hObject    handle to continuar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close all;
```

ANEXO A

Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**	
Title	**IEEE Standard 802.16: A Technical Overview of the WirelessMAN™ Air Interface for Broadband Wireless Access**	
Date Submitted	**2002-06-04**	
Source(s)	Roger Marks NIST 325 Broadway Boulder, CO 80305	Voice: +1-303-497-3037 Fax: +1-303-497-3037 mailto:r.b.marks@ieee.org
Re:	IEEE Std 802.16	
Abstract	The broadband wireless access industry, which provides high-rate network connections to stationary sites, has matured to the point at which it now has a standard for second-generation wireless metropolitan area networks. IEEE Standard 802.16, with its WirelessMAN™ air interface, sets the stage for widespread and effective deployments worldwide. This article overviews the technical medium access control and physical layer features of this new standard. This article was written by Carl Eklund, Roger B. Marks, Kenneth L. Stanwood, and Stanley Wang. It was published in *IEEE Communications Magazine*, June 2002, pp. 98-107." For more details, see: **<http://www.comsoc.org/ci1/Public/2002/Jun/index.html>**	
Purpose	This document will improve the effectiveness of participants in IEEE Working Group 802.16 by providing an overview of the published base standard upon which the Working Group is currently developing several amendments.	
Notice	This document has been prepared to assist IEEE 802.16. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.	
Patent Policy and Procedures	The contributor is familiar with the IEEE 802.16 Patent Policy and Procedures (Version 1.0) <http://ieee802.org/16/ipr/patents/policy.html>, including the statement "IEEE standards may include the known use of patent(s), including patent applications, if there is technical justification in the opinion of the standards-developing committee and provided the IEEE receives assurance from the patent holder that it will license applicants under reasonable terms and conditions for the purpose of implementing the standard." Early disclosure to the Working Group of patent information that might be relevant to the standard is essential to reduce the possibility for delays in the development process and increase the likelihood that the draft publication will be approved for publication. Please notify the Chair <mailto:r.b.marks@ieee.org> as early as possible, in written or electronic form, of any patents (granted or under application) that may cover technology that is under consideration by or has been approved by IEEE 802.16. The Chair will disclose this notification via the IEEE 802.16 web site <http://ieee802.org/16/ipr/patents/notices>.	

IEEE Standard 802.16: A Technical Overview of the WirelessMAN™ Air Interface for Broadband Wireless Access

Carl Eklund, Nokia Research Center
Roger B. Marks, National Institute of Standards and Technology
Kenneth L. Stanwood and Stanley Wang, Ensemble Communications Inc.

ABSTRACT

The broadband wireless access industry, which provides high-rate network connections to stationary sites, has matured to the point at which it now has a standard for second-generation wireless metropolitan area networks. IEEE Standard 802.16, with its WirelessMAN™ air interface, sets the stage for widespread and effective deployments worldwide. This article overviews the technical medium access control and physical layer features of this new standard.

INTRODUCTION AND MARKET OPPORTUNITIES

IEEE Standard 802.16-2001 [1], completed in October 2001 and published on 8 April 2002, defines the WirelessMAN™ air interface specification for wireless metropolitan area networks (MANs). The completion of this standard heralds the entry of broadband wireless access as a major new tool in the effort to link homes and businesses to core telecommunications networks worldwide.

As currently defined through IEEE Standard 802.16, a wireless MAN provides network access to buildings through exterior antennas communicating with central radio base stations (BSs). The wireless MAN offers an alternative to cabled access networks, such as fiber optic links, coaxial systems using cable modems, and digital subscriber line (DSL) links. Because wireless systems have the capacity to address broad geographic areas without the costly infrastructure development required in deploying cable links to individual sites, the technology may prove less expensive to deploy and may lead to more ubiquitous broadband access. Such systems have been in use for several years, but the development of the new standard marks the maturation of the industry and forms the basis of new industry success using second-generation equipment.

In this scenario, with WirelessMAN technology bringing the network to a building, users inside the building will connect to it with conventional in-building networks such as, for data, Ethernet (IEEE Standard 802.3) or wireless LANs (IEEE Standard 802.11). However, the fundamental design of the standard may eventually allow for the efficient extension of the WirelessMAN networking protocols directly to the individual user. For instance, a central BS may someday exchange medium access control (MAC) protocol data with an individual laptop computer in a home. The links from the BS to the home receiver and from the home receiver to the laptop would likely use quite different physical layers, but design of the WirelessMAN MAC could accommodate such a connection with full quality of service (QoS). With the technology expanding in this direction, it is likely that the standard will evolve to support nomadic and increasingly mobile users. For example, it could be suitable for a stationary or slow-moving vehicle.

IEEE Standard 802.16 was designed to evolve as a set of air interfaces based on a common MAC protocol but with physical layer specifications dependent on the spectrum of use and the associated regulations. The standard, as approved in 2001, addresses frequencies from 10 to 66 GHz, where extensive spectrum is currently available worldwide but at which the short wavelengths introduce significant deployment challenges. A new project, currently in the balloting stage, expects to complete an amend-

0163-6804/02/$17.00 © 2002 IEEE

ment denoted IEEE 802.16a [2] before the end of 2002. This document will extend the air interface support to lower frequencies in the 2–11 GHz band, including both licensed and license-exempt spectra. Compared to the higher frequencies, such spectra offer the opportunity to reach many more customers less expensively, although at generally lower data rates. This suggests that such services will be oriented toward individual homes or small to medium-sized enterprises.

The 802.16 Working Group

Development of IEEE Standard 802.16 and the included WirelessMAN™ air interface, along with associated standards and amendments, is the responsibility of IEEE Working Group 802.16 on Broadband Wireless Access (BWA) Standards (http://WirelessMAN.org). The Working Group's initial interest was the 10–66 GHz range. The 2–11 GHz amendment project that led to IEEE 802.16a was approved in March 2000. The 802.16a project primarily involves the development of new physical layer specifications, with supporting enhancements to the basic MAC. In addition, the Working Group has completed IEEE Standard 802.16.2 [3] ("Recommended Practice for Coexistence of Fixed Broadband Wireless Access Systems") to address 10–66 GHz coexistence and, through the amendment project 802.16.2a, is expanding its recommendations to include licensed bands from 2 to 11 GHz.

Historically, the 802.16 activities were initiated at an August 1998 meeting called by the National Wireless Electronics Systems Testbed (N-WEST) of the U.S. National Institute of Standards and Technology. The effort was welcomed in IEEE 802, which opened a Study Group. The 802.16 Working Group has held weeklong meetings at least bimonthly since July 1999. Over 700 individuals have attended a session. Membership, which is granted to individuals based on their attendance and participation, currently stands at 130. The work has been closely followed; for example, the IEEE 802.16 Web site received over 2.8 million file requests in 2000.

Technology Design Issues

Medium Access Control

The IEEE 802.16 MAC protocol was designed for point-to-multipoint broadband wireless access applications. It addresses the need for very high bit rates, both uplink (to the BS) and downlink (from the BS). Access and bandwidth allocation algorithms must accommodate hundreds of terminals per channel, with terminals that may be shared by multiple end users. The services required by these end users are varied in their nature and include legacy time-division multiplex (TDM) voice and data, Internet Protocol (IP) connectivity, and packetized voice over IP (VoIP). To support this variety of services, the 802.16 MAC must accommodate both continuous and bursty traffic. Additionally, these services expect to be assigned QoS in keeping with the traffic types. The 802.16 MAC provides a wide range of service types analogous to the classic asynchronous transfer mode (ATM) service categories as well as newer categories such as guaranteed frame rate (GFR).

The 802.16 MAC protocol must also support a variety of backhaul requirements, including both asynchronous transfer mode (ATM) and packet-based protocols. Convergence sublayers are used to map the transport-layer-specific traffic to a MAC that is flexible enough to efficiently carry any traffic type. Through such features as payload header suppression, packing, and fragmentation, the convergence sublayers and MAC work together to carry traffic in a form that is often more efficient than the original transport mechanism.

Issues of transport efficiency are also addressed at the interface between the MAC and the physical layer (PHY). For example, the modulation and coding schemes are specified in a burst profile that may be adjusted adaptively for each burst to each subscriber station. The MAC can make use of bandwidth-efficient burst profiles under favorable link conditions but shift to more reliable, although less efficient, alternatives as required to support the planned 99.999 percent link availability.

The request-grant mechanism is designed to be scalable, efficient, and self-correcting. The 802.16 access system does not lose efficiency when presented with multiple connections per terminal, multiple QoS levels per terminal, and a large number of statistically multiplexed users. It takes advantage of a wide variety of request mechanisms, balancing the stability of contentionless access with the efficiency of contention-oriented access.

While extensive bandwidth allocation and QoS mechanisms are provided, the details of scheduling and reservation management are left unstandardized and provide an important mechanism for vendors to differentiate their equipment.

Along with the fundamental task of allocating bandwidth and transporting data, the MAC includes a privacy sublayer that provides authentication of network access and connection establishment to avoid theft of service, and it provides key exchange and encryption for data privacy.

To accommodate the more demanding physical environment and different service requirements of the frequencies between 2 and 11 GHz, the 802.16a project is upgrading the MAC to provide automatic repeat request (ARQ) and support for mesh, rather than only point-to-multipoint, network architectures.

The Physical Layer

10–66 GHz — In the design of the PHY specification for 10–66 GHz, line-of-sight propagation was deemed a practical necessity. With this condition assumed, single-carrier modulation was easily selected; the air interface is designated "WirelessMAN-SC." Many fundamental design challenges remained, however. Because of the point-to-multipoint architecture, the BS basically transmits a TDM signal, with individual subscriber stations allocated time slots serially. Access in the uplink direction is by time-division multiple access (TDMA). Following extensive discussions regarding duplexing, a

While extensive bandwidth allocation and QoS mechanisms are provided, the details of scheduling and reservation management are left unstandardized and provide an important mechanism for vendors to differentiate their equipment.

The PHY specification defined for 10–66 GHz uses burst single-carrier modulation with adaptive burst profiling in which transmission parameters, including the modulation and codling schemes, my be adjusted individually to each subscriber station on a frame-by-frame basis. Both TDD and burst FDD variants are defined.

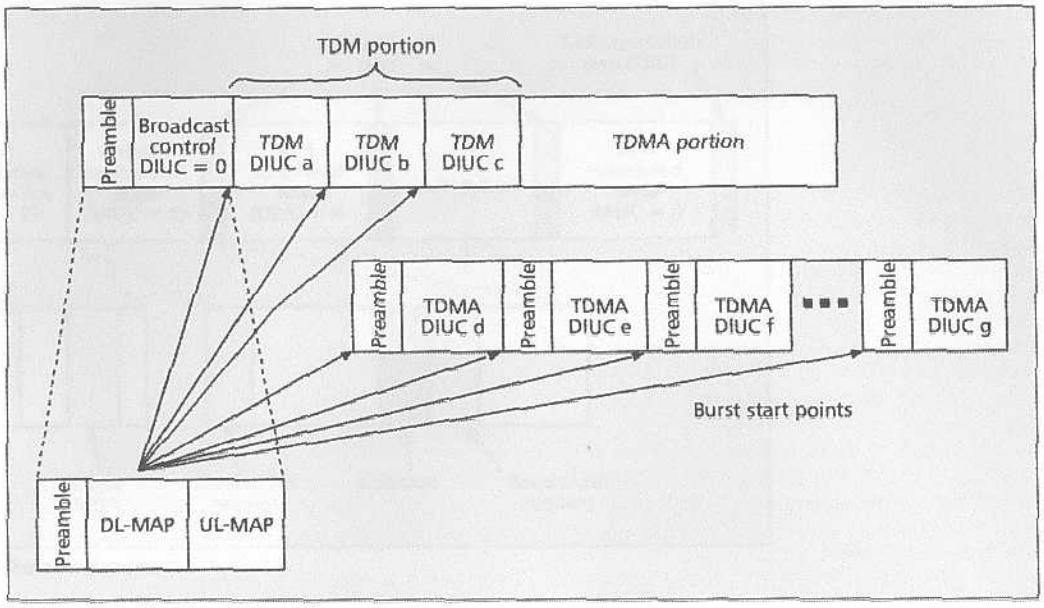

Figure 1. *The downlink subframe structure.*

burst design was selected that allows both time-division duplexing (TDD), in which the uplink and downlink share a channel but do not transmit simultaneously, and frequency-division duplexing (FDD), in which the uplink and downlink operate on separate channels, sometimes simultaneously. This burst design allows both TDD and FDD to be handled in a similar fashion. Support for half-duplex FDD subscriber stations, which may be less expensive since they do not simultaneously transmit and receive, was added at the expense of some slight complexity. Both TDD and FDD alternatives support adaptive burst profiles in which modulation and coding options may be dynamically assigned on a burst-by-burst basis.

2–11 GHz — The 2–11 GHz bands, both licensed and license-exempt, are addressed in IEEE Project 802.16a. The standard is in ballot but is not yet complete. The draft currently specifies that compliant systems implement one of three air interface specifications, each of which provides for interoperability. Design of the 2–11 GHz physical layer is driven by the need for non-line-of-sight (NLOS) operation. Because residential applications are expected, rooftops may be too low for a clear sight line to a BS antenna, possibly due to obstruction by trees. Therefore, significant multipath propagation must be expected. Furthermore, outdoor-mounted antennas are expensive due to both hardware and installation costs.

The three 2–11 GHz air interface specifications in 802.16a Draft 3 are:

- WirelessMAN-SC2: This uses a single-carrier modulation format.
- WirelessMAN-OFDM: This uses orthogonal frequency-division multiplexing with a 256-point transform. Access is by TDMA. This air interface is mandatory for license-exempt bands.
- WirelessMAN-OFDMA: This uses orthogonal frequency-division multiple access with a 2048-point transform. In this system, multiple access is provided by addressing a subset of the multiple carriers to individual receivers.

Because of the propagation requirements, the use of advanced antenna systems is supported.

It is premature to speculate on further specifics of the 802.16a amendment prior to its completion. While the draft seems to have reached a level of maturity, the contents could change significantly in balloting. Modes could even be deleted or added.

Physical Layer Details

The PHY specification defined for 10–66 GHz uses burst single-carrier modulation with adaptive burst profiling in which transmission parameters, including the modulation and coding schemes, may be adjusted individually to each subscriber station (SS) on a frame-by-frame basis. Both TDD and burst FDD variants are defined. Channel bandwidths of 20 or 25 MHz (typical U.S. allocation) or 28 MHz (typical European allocation) are specified, along with Nyquist square-root raised-cosine pulse shaping with a rolloff factor of 0.25. Randomization is performed for spectral shaping and to ensure bit transitions for clock recovery.

The forward error correction (FEC) used is Reed-Solomon GF(256), with variable block size and error correction capabilities. This is paired with an inner block convolutional code to robustly transmit critical data, such as frame control and initial accesses. The FEC options are paired with quadrature phase shift keying (QPSK), 16-state quadrature amplitude modulation (16-QAM), and 64-state QAM (64-QAM) to form burst profiles of varying robustness and efficiency. If the last FEC block is not filled, that block may be shortened. Shortening in both the uplink

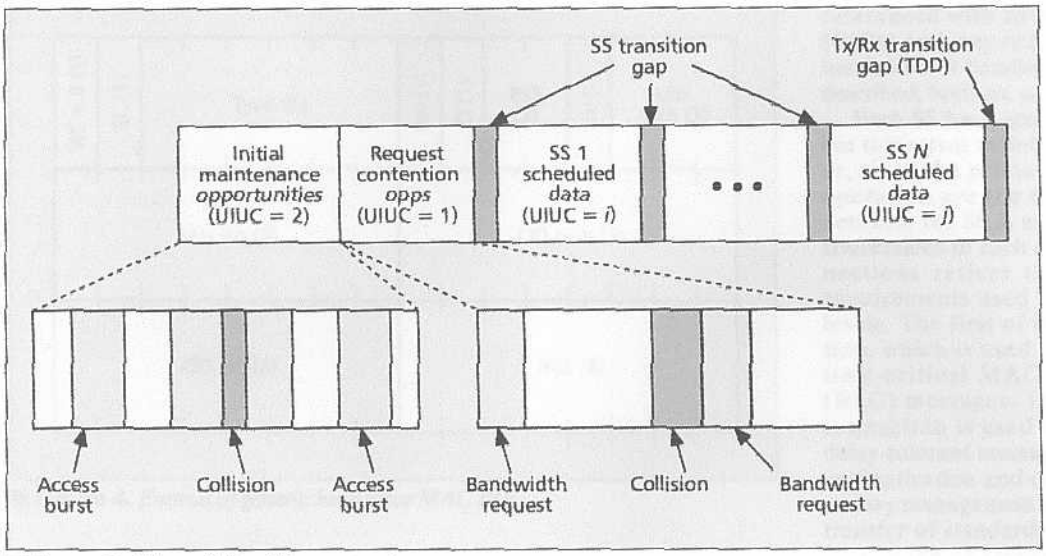

Figure 2. *The uplink subframe structure.*

and downlink is controlled by the BS and is implicitly communicated in the uplink map (UL-MAP) and downlink map (DL-MAP).

The system uses a frame of 0.5, 1, or 2 ms. This frame is divided into physical slots for the purpose of bandwidth allocation and identification of PHY transitions. A physical slot is defined to be 4 QAM symbols. In the TDD variant of the PHY, the uplink subframe follows the downlink subframe on the same carrier frequency. In the FDD variant, the uplink and downlink subframes are coincident in time but are carried on separate frequencies. The downlink subframe is shown in Fig. 1.

The downlink subframe starts with a frame control section that contains the DL-MAP for the current downlink frame as well as the UL-MAP for a specified time in the future. The downlink map specifies when physical layer transitions (modulation and FEC changes) occur within the downlink subframe. The downlink subframe typically contains a TDM portion immediately following the frame control section. Downlink data are transmitted to each SS using a negotiated burst profile. The data are transmitted in order of decreasing robustness to allow SSs to receive their data before being presented with a burst profile that could cause them to lose synchronization with the downlink.

In FDD systems, the TDM portion may be followed by a TDMA segment that includes an extra preamble at the start of each new burst profile. This feature allows better support of half-duplex SSs. In an efficiently scheduled FDD system with many half-duplex SSs, some may need to transmit earlier in the frame than they receive. Due to their half-duplex nature, these SSs lose synchronization with the downlink. The TDMA preamble allows them to regain synchronization.

Due to the dynamics of bandwidth demand for the variety of services that may be active, the mixture and duration of burst profiles and the presence or absence of a TDMA portion vary dynamically from frame to frame. Since the recipient SS is implicitly indicated in the MAC

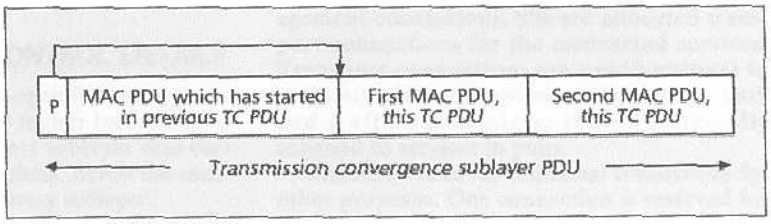

Figure 3. *TC PDU format.*

headers rather than in the DL-MAP, SSs listen to all portions of the downlink subframe they are capable of receiving. For full-duplex SSs, this means receiving all burst profiles of equal or greater robustness than they have negotiated with the BS.

A typical uplink subframe for the 10–66 GHz PHY is shown in Fig. 2. Unlike the downlink, the UL-MAP grants bandwidth to specific SSs. The SSs transmit in their assigned allocation using the burst profile specified by the Uplink Interval Usage Code (UIUC) in the UL-MAP entry granting them bandwidth. The uplink subframe may also contain contention-based allocations for initial system access and broadcast or multicast bandwidth requests. The access opportunities for initial system access are sized to allow extra guard time for SSs that have not resolved the transmit time advance necessary to offset the round-trip delay to the BS.

Between the PHY and MAC is a transmission convergence (TC) sublayer. This layer performs the transformation of variable length MAC protocol data units (PDUs) into the fixed length FEC blocks (plus possibly a shortened block at the end) of each burst. The TC layer has a PDU sized to fit in the FEC block currently being filled. It starts with a pointer indicating where the next MAC PDU header starts within the FEC block. This is shown in Fig. 3.

The TC PDU format allows resynchronization to the next MAC PDU in the event that the previous FEC block had irrecoverable errors.

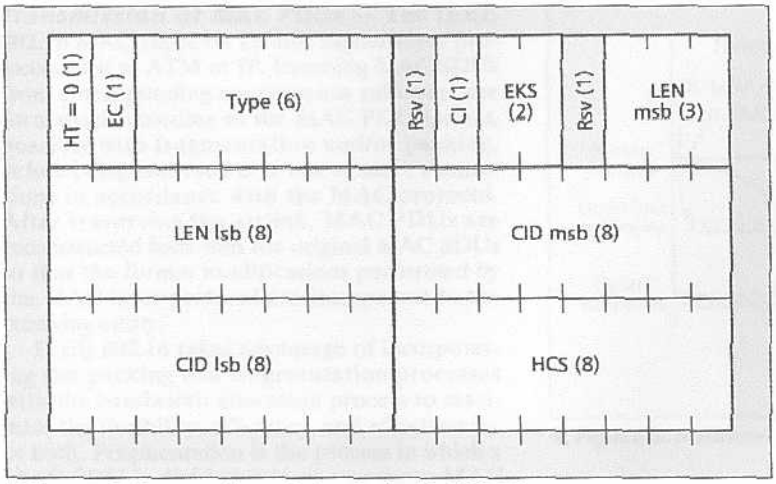

Figure 4. *Format of generic header for MAC PDU.*

Without the TC layer, a receiving SS or BS would potentially lose the entire remainder of a burst when an irrecoverable bit error occurred.

Medium Access Control Details

The MAC includes service-specific convergence sublayers that interface to higher layers, above the core MAC common part sublayer that carries out the key MAC functions. Below the common part sublayer is the privacy sublayer.

Service-Specific Convergence Sublayers

IEEE Standard 802.16 defines two general service-specific convergence sublayers for mapping services to and from 802.16 MAC connections. The ATM convergence sublayer is defined for ATM services, and the packet convergence sublayer is defined for mapping packet services such as IPv4, IPv6, Ethernet, and virtual local area network (VLAN). The primary task of the sublayer is to classify service data units (SDUs) to the proper MAC connection, preserve or enable QoS, and *enable bandwidth allocation. The mapping* takes various forms depending on the type of service. In addition to these basic functions, the convergence sublayers can also perform more sophisticated functions such as payload header suppression and reconstruction to enhance airlink efficiency.

Common Part Sublayer

Introduction and General Architecture — In general, the 802.16 MAC is designed to support *a point-to-multipoint architecture with a central* BS handling multiple independent sectors simultaneously. On the downlink, data to SSs are multiplexed in TDM fashion. The uplink is shared between SSs in TDMA fashion.

The 802.16 MAC is connection-oriented. All services, including inherently connectionless services, are mapped to a connection. This provides a mechanism for requesting bandwidth, associating QoS and traffic parameters, transporting and routing data to the appropriate convergence sublayer, and all other actions associated with the contractual terms of the service. Connections are referenced with 16-bit connection identifiers (CIDs) and may require continuously granted bandwidth or bandwidth on demand. As will be described, both are accommodated.

Each SS has a standard 48-bit MAC address, but this serves mainly as an equipment identifier, since the primary addresses used during operation are the CIDs. Upon entering the network, the SS is assigned three management connections in each direction. These three connections reflect the three different QoS requirements used by different management levels. The first of these is the basic connection, which is used for the transfer of short, time-critical MAC and radio link control (RLC) messages. The primary management connection is used to transfer longer, more delay-tolerant messages such as those used for authentication and connection setup. The secondary management connection is used for the transfer of standards-based management messages such as Dynamic Host Configuration Protocol (DHCP), Trivial File Transfer Protocol (TFTP), and Simple Network Management Protocol (SNMP). In addition to these management connections, SSs are allocated transport connections for the contracted services. Transport connections are unidirectional to facilitate different uplink and downlink QoS and traffic parameters; they are typically assigned to services in pairs.

The MAC reserves additional connections for other purposes. One connection is reserved for contention-based initial access. Another is reserved for broadcast transmissions in the downlink as well as for signaling broadcast contention-based polling of SS bandwidth needs. Additional connections are reserved for multicast, rather than broadcast, contention-based polling. SSs may be instructed to join multicast polling groups associated with these multicast polling connections.

MAC PDU Formats — The MAC PDU is the data unit exchanged between the MAC layers of the BS and its SSs. A MAC PDU consists of a fixed-length MAC header, a variable-length payload, and an optional cyclic redundancy check (CRC). Two header formats, distinguished by the HT field, are defined: the generic header (Fig. 4) and the bandwidth request header.

Except for bandwidth request MAC PDUs, which contain no payload, MAC PDUs contain either MAC management messages or convergence sublayer data.

Three types of MAC subheader may be present. The grant management subheader is used by an SS to convey bandwidth management needs to its BS. The fragmentation subheader contains information that indicates the presence and orientation in the payload of any fragments of SDUs. The packing subheader is used to indicate the packing of multiple SDUs into a single PDU. The grant management and fragmentation subheaders may be inserted in MAC PDUs immediately following the generic header if so indicated by the Type field. The packing subheader may be inserted before each MAC SDU if so indicated by the Type field. More details are provided below.

Transmission of MAC PDUs — The IEEE 802.16 MAC supports various higher-layer protocols such as ATM or IP. Incoming MAC SDUs from corresponding convergence sublayers are formatted according to the MAC PDU format, possibly with fragmentation and/or packing, before being conveyed over one or more connections in accordance with the MAC protocol. After traversing the airlink, MAC PDUs are reconstructed back into the original MAC SDUs so that the format modifications performed by the MAC layer protocol are transparent to the receiving entity.

IEEE 802.16 takes advantage of incorporating the packing and fragmentation processes with the bandwidth allocation process to maximize the flexibility, efficiency, and effectiveness of both. Fragmentation is the process in which a MAC SDU is divided into one or more MAC SDU fragments. Packing is the process in which multiple MAC SDUs are packed into a single MAC PDU payload. Both processes may be initiated by either a BS for a downlink connection or an SS for an uplink connection.

IEEE 802.16 allows simultaneous fragmentation and packing for efficient use of the bandwidth.

PHY Support and Frame Structure — The IEEE 802.16 MAC supports both TDD and FDD. In FDD, both continuous and burst downlinks are supported. Continuous downlinks allow for certain robustness enhancement techniques, such as interleaving. Burst downlinks (either FDD or TDD) allow the use of more advanced robustness and capacity enhancement techniques, such as subscriber-level adaptive burst profiling and advanced antenna systems.

The MAC builds the downlink subframe starting with a frame control section containing the DL-MAP and UL-MAP messages. These indicate PHY transitions on the downlink as well as bandwidth allocations and burst profiles on the uplink.

The DL-MAP is always applicable to the current frame and is always at least two FEC blocks long. The first PHY transition is expressed in the first FEC block, to allow adequate processing time. In both TDD and FDD systems, the UL-MAP provides allocations starting no later than the next downlink frame. The UL-MAP can, however, allocate starting in the current frame as long as processing times and round-trip delays are observed. The minimum time between receipt and applicability of the UL-MAP for an FDD system is shown in Fig. 5.

Radio Link Control — The advanced technology of the 802.16 PHY requires equally advanced radio link control (RLC), particularly the capability of the PHY to transition from one burst profile to another. The RLC must control this capability as well as the traditional RLC functions of power control and ranging.

RLC begins with periodic BS broadcast of the burst profiles that have been chosen for the uplink and downlink. The particular burst profiles used on a channel are chosen based on a number of factors, such as rain region and equipment capabilities. Burst profiles for the downlink are each tagged with a Downlink Interval Usage Code (DIUC). Those for the uplink are each tagged with an Uplink Interval Usage Code (UIUC).

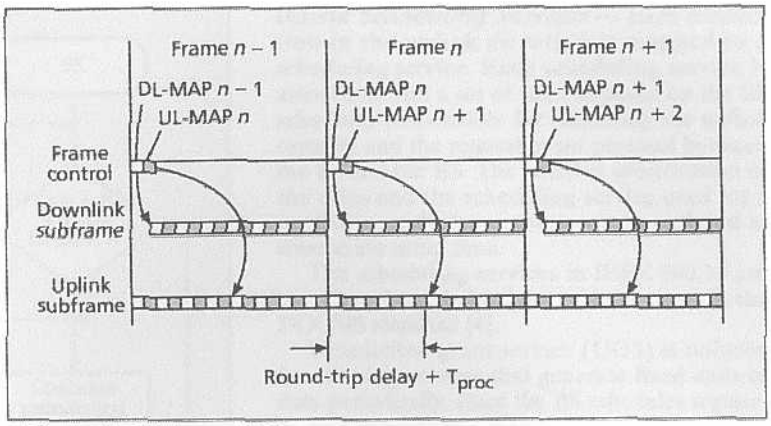

Figure 5. *Minimum FDD map relevance.*

During initial access, the SS performs initial power leveling and ranging using ranging request (RNG-REQ) messages transmitted in initial maintenance windows. The adjustments to the SS's transmit time advance, as well as power adjustments, are returned to the SS in ranging response (RNG-RSP) messages. For ongoing ranging and power adjustments, the BS may transmit unsolicited RNG-RSP messages commanding the SS to adjust its power or timing.

During initial ranging, the SS also requests to be served in the downlink via a particular burst profile by transmitting its choice of DIUC to the BS. The choice is based on received downlink signal quality measurements performed by the SS before and during initial ranging. The BS may confirm or reject the choice in the ranging response. Similarly, the BS monitors the quality of the uplink signal it receives from the SS. The BS commands the SS to use a particular uplink burst profile simply by including the appropriate burst profile UIUC with the SS's grants in UL-MAP messages.

After initial determination of uplink and downlink burst profiles between the BS and a particular SS, RLC continues to monitor and control the burst profiles. Harsher environmental conditions, such as rain fades, can force the SS to request a more robust burst profile. Alternatively, exceptionally good weather may allow an SS to temporarily operate with a more efficient burst profile. The RLC continues to adapt the SS's current UL and DL burst profiles, ever striving to achieve a balance between robustness and efficiency. Because the BS is in control and directly monitors the uplink signal quality, the protocol for changing the uplink burst profile for an SS is simple: the BS merely specifies the profile's associated UIUC whenever granting the SS bandwidth in a frame. This eliminates the need for an acknowledgment, since the SS will always receive either both the UIUC and the grant or neither. Hence, no chance of uplink burst profile mismatch between the BS and SS exists.

In the downlink, the SS is the entity that monitors the quality of the receive signal and therefore knows when its downlink burst profile

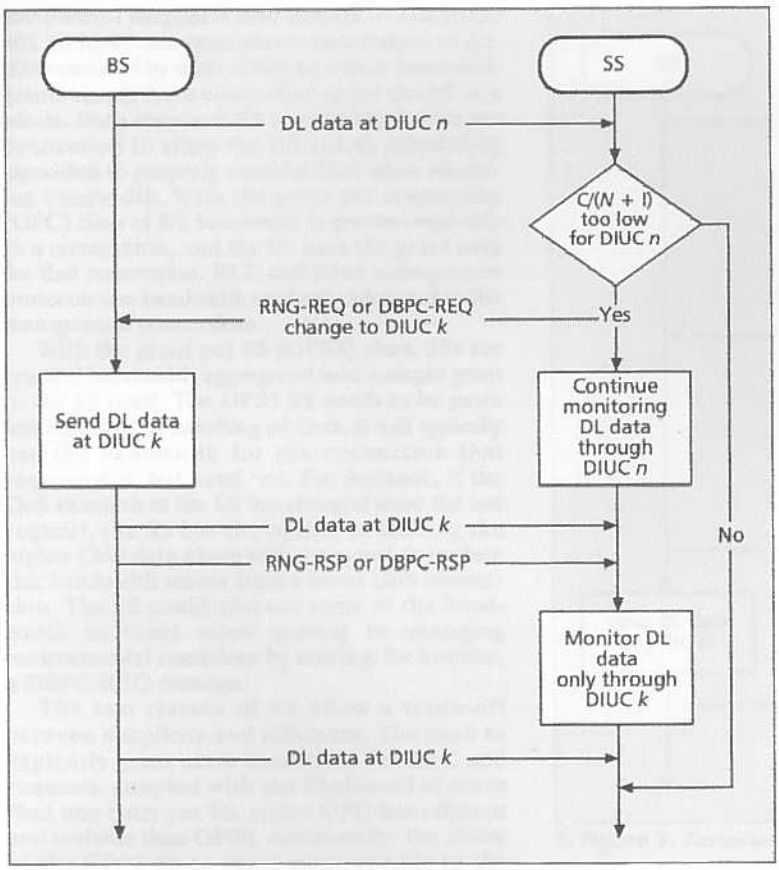

Figure 6. *Transition to a more robust burst profile.*

should change. The BS, however, is the entity in control of the change. There are two methods available to the SS to request a change in downlink burst profile, depending on whether the SS operates in the grant per connection (GPC) or grant per SS (GPSS) mode (see "Bandwidth Requests and Grants"). The first method would typically apply (based on the discretion of the BS scheduling algorithm) only to GPC SSs. In this case, the BS may periodically allocate a station maintenance interval to the SS. The SS can use the RNG-REQ message to request a change in downlink burst profile. The preferred method is for the SS to transmit a downlink burst profile change request (DBPC-REQ). In this case, which is always an option for GPSS SSs and can be an option for GPC SSs, the BS responds with a downlink burst profile change response (DBPC-RSP) message confirming or denying the change.

Because messages may be lost due to irrecoverable bit errors, the protocols for changing an SS's downlink burst profile must be carefully structured. The order of the burst profile change actions is different when transitioning to a more robust burst profile than when transitioning to a less robust one. The standard takes advantage of the fact that an SS is always required to listen to more robust portions of the downlink as well as the profile that was negotiated. Figure 6 shows a transition to a more robust burst profile. Figure 7 shows a transition to a less robust burst profile.

Uplink Scheduling Services — Each connection in the uplink direction is mapped to a *scheduling service*. Each scheduling service is associated with a set of rules imposed on the BS scheduler responsible for allocating the uplink capacity and the request-grant protocol between the SS and the BS. The detailed specification of the rules and the scheduling service used for a particular uplink connection is negotiated at connection setup time.

The scheduling services in IEEE 802.16 are based on those defined for cable modems in the DOCSIS standard [4].

Unsolicited grant service (UGS) is tailored for carrying services that generate fixed units of data periodically. Here the BS schedules regularly, in a preemptive manner, grants of the size negotiated at connection setup, without an explicit request from the SS. This eliminates the overhead and latency of bandwidth requests in order to meet the delay and delay jitter requirements of the underlying service. A practical limit on the delay jitter is set by the frame duration. If more stringent jitter requirements are to be met, output buffering is needed. Services that typically would be carried on a connection with UGS service include ATM constant bit rate (CBR) and E1/T1 over ATM.

When used with UGS, the grant management subheader includes the poll-me bit (see "Bandwidth Requests and Grants") as well as the slip indicator flag, which allows the SS to report that the transmission queue is backlogged due to factors such as lost grants or clock skew between the IEEE 802.16 system and the outside network. The BS, upon detecting the slip indicator flag, can allocate some additional capacity to the SS, allowing it to recover the normal queue state. Connections configured with UGS are not allowed to utilize random access opportunities for requests.

The real-time polling service is designed to meet the needs of services that are dynamic in nature, but offers periodic dedicated request opportunities to meet real-time requirements. Because the SS issues explicit requests, the protocol overhead and latency is increased, but this capacity is granted only according to the real need of the connection. The real-time polling service is well suited for connections carrying services such as VoIP or streaming video or audio.

The non-real-time polling service is almost identical to the real-time polling service except that connections may utilize random access transmit opportunities for sending bandwidth requests. Typically, services carried on these connections tolerate longer delays and are rather insensitive to delay jitter. The non-real-time polling service is suitable for Internet access with a minimum guaranteed rate and for ATM GFR connections.

A best effort service has also been defined. Neither throughput nor delay guarantees are provided. The SS sends requests for bandwidth in either random access slots or dedicated transmission opportunities. The occurrence of dedicated opportunities is subject to network load, and the SS cannot rely on their presence.

Bandwidth Requests and Grants — The IEEE 802.16 MAC accommodates two classes of SS, differentiated by their ability to accept bandwidth grants simply for a connection or for the SS as a whole. Both classes of SS request bandwidth per connection to allow the BS uplink scheduling algorithm to properly consider QoS when allocating bandwidth. With the grant per connection (GPC) class of SS, bandwidth is granted explicitly to a connection, and the SS uses the grant only for that connection. RLC and other management protocols use bandwidth explicitly allocated to the management connections.

With the grant per SS (GPSS) class, SSs are granted bandwidth aggregated into a single grant to the SS itself. The GPSS SS needs to be more intelligent in its handling of QoS. It will typically use the bandwidth for the connection that requested it, but need not. For instance, if the QoS situation at the SS has changed since the last request, the SS has the option of sending the higher QoS data along with a request to replace this bandwidth stolen from a lower QoS connection. The SS could also use some of the bandwidth to react more quickly to changing environmental conditions by sending, for instance, a DBPC-REQ message.

The two classes of SS allow a trade-off between simplicity and efficiency. The need to explicitly grant extra bandwidth for RLC and requests, coupled with the likelihood of more than one entry per SS, makes GPC less efficient and scalable than GPSS. Additionally, the ability of the GPSS SS to react more quickly to the needs of the PHY and those of connections enhances system performance. GPSS is the only class of SS allowed with the 10–66 GHz PHY.

With both classes of grants, the IEEE 802.16 MAC uses a self-correcting protocol rather than an acknowledged protocol. This method uses less bandwidth. Furthermore, acknowledged protocols can take additional time, potentially adding delay. There are a number of reasons the bandwidth requested by an SS for a connection may not be available:

- The BS did not see the request due to irrecoverable PHY errors or collision of a contention-based reservation.
- The SS did not see the grant due to irrecoverable PHY errors.
- The BS did not have sufficient bandwidth available.
- The GPSS SS used the bandwidth for another purpose.

In the self-correcting protocol, all of these anomalies are treated the same. After a timeout appropriate for the QoS of the connection (or immediately, if the bandwidth was stolen by the SS for another purpose), the SS simply requests again. For efficiency, most bandwidth requests are incremental; that is, the SS asks for more bandwidth for a connection. However, for the self-correcting bandwidth request/grant mechanism to work correctly, the bandwidth requests must occasionally be aggregate; that is, the SS informs the BS of its total current bandwidth needs for a connection. This allows the BS to reset its perception of the SS's needs without a complicated protocol acknowledging the use of granted bandwidth.

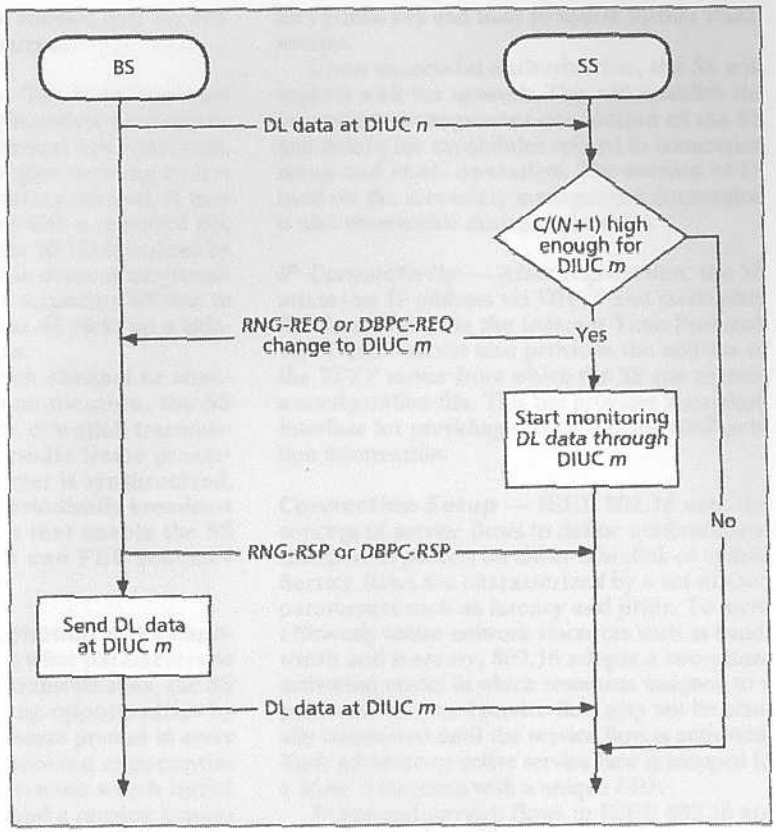

Figure 7. *Transition to a less robust burst profile.*

The SS has a plethora of ways to request bandwidth, combining the determinism of unicast polling with the responsiveness of contention-based requests and the efficiency of unsolicited bandwidth. For continuous bandwidth demand, such as with CBR T1/E1 data, the SS need not request bandwidth; the BS grants it unsolicited.

To short-circuit the normal polling cycle, any SS with a connection running UGS can use the poll-me bit in the grant management subheader to let the BS know it needs to be polled for bandwidth needs on another connection. The BS may choose to save bandwidth by polling SSs that have unsolicited grant services only when they have set the poll-me bit.

A more conventional way to request bandwidth is to send a bandwidth request MAC PDU that consists of simply the bandwidth request header and no payload. GPSS SSs can send this in any bandwidth allocation they receive. GPC terminals can send it in either a request interval or a data grant interval allocated to their basic connection. A closely related method of requesting data is to use a grant management subheader to piggyback a request for additional bandwidth for the same connection within a MAC PDU.

In addition to polling individual SSs, the BS may issue a broadcast poll by allocating a request interval to the broadcast CID. Similarly, the standard provides a protocol for forming multicast groups to give finer control to contention-based polling. Due to the nondeterministic delay that can be caused by collisions and retries, con-

In general, service flows in IEEE 802.16 are preprovisioned, and setup of the service flows is initiated by the BS during SS initialization. However, service flows can also be dynamically established by either the BS or the SS.

tention-based requests are allowed only for certain lower QoS classes of service.

Channel Acquisition — The MAC protocol includes an initialization procedure designed to eliminate the need for manual configuration. Upon installation, an SS begins scanning its frequency list to find an operating channel. It may be programmed to register with a specified BS, referring to a programmable BS ID broadcast by each. This feature is useful in dense deployments where the SS might hear a secondary BS due to selective fading or when the SS picks up a sidelobe of a nearby BS antenna.

After deciding on which channel or channel pair to attempt communication, the SS tries to synchronize to the downlink transmission by detecting the periodic frame preambles. Once the physical layer is synchronized, the SS will look for the periodically broadcast DCD and UCD messages that enable the SS to learn the modulation and FEC schemes used on the carrier.

Initial Ranging and Negotiation of SS Capabilities — Upon learning what parameters to use for its initial ranging transmissions, the SS will look for initial ranging opportunities by scanning the UL-MAP messages present in every frame. The SS uses a truncated exponential backoff algorithm to determine which initial ranging slot it will use to send a ranging request message. The SS will send the burst using the minimum power setting and will try again with increasingly higher transmission power if it does not receive a ranging response.

Based on the arrival time of the initial ranging request and the measured power of the signal, the BS commands a timing advance and a power adjustment to the SS in the ranging response. The response also provides the SS with the basic and primary management CIDs. Once the timing advance of the SS transmissions has been correctly determined, the ranging procedure for fine-tuning the power can be performed using invited transmissions.

All transmissions up to this point are made using the most robust, and thus least efficient, burst profile. To avoid wasting capacity, the SS next reports its PHY capabilities, including the modulation and coding schemes it supports and whether, in an FDD system, it is half-duplex or full-duplex. The BS, in its response, can deny the use of any capability reported by the SS.

SS Authentication and Registration — Each SS contains both a manufacturer-issued factory-installed X.509 digital certificate and the certificate of the manufacturer. These certificates, which establish a link between the 48-bit MAC address of the SS and its public RSA key, are sent to the BS by the SS in the Authorization Request and Authentication Information messages. The network is able to verify the identity of the SS by checking the certificates and can subsequently check the level of authorization of the SS. If the SS is authorized to join the network, the BS will respond to its request with an Authorization Reply containing an Authorization Key (AK) encrypted with the SS's public key and used to secure further transactions.

Upon successful authorization, the SS will register with the network. This will establish the secondary management connection of the SS and determine capabilities related to connection setup and MAC operation. The version of IP used on the secondary management connection is also determined during registration.

IP Connectivity — After registration, the SS attains an IP address via DHCP and establishes the time of day via the Internet Time Protocol. The DHCP server also provides the address of the TFTP server from which the SS can request a configuration file. This file provides a standard interface for providing vendor-specific configuration information.

Connection Setup — IEEE 802.16 uses the concept of service flows to define unidirectional transport of packets on either downlink or uplink. Service flows are characterized by a set of QoS parameters such as latency and jitter. To most efficiently utilize network resources such as bandwidth and memory, 802.16 adopts a two-phase activation model in which resources assigned to a particular admitted service flow may not be actually committed until the service flow is activated. Each admitted or active service flow is mapped to a MAC connection with a unique CID.

In general, service flows in IEEE 802.16 are preprovisioned, and setup of the service flows is initiated by the BS during SS initialization. However, service flows can also be dynamically established by either the BS or the SS. The SS typically initiates service flows only if there is a dynamically signaled connection, such as a switched virtual connection (SVC) from an ATM network. The establishment of service flows is performed via a three-way handshaking protocol in which the request for service flow establishment is responded to and the response acknowledged.

In addition to dynamic service establishment, IEEE 802.16 also supports dynamic service changes in which service flow parameters are renegotiated. Like dynamic service flow establishment, service flow changes also follow a similar three-way handshaking protocol.

Privacy Sublayer — IEEE 802.16's privacy protocol is based on the Privacy Key Management (PKM) protocol of the DOCSIS BPI+ specification [5] but has been enhanced to fit seamlessly into the IEEE 802.16 MAC protocol and to better accommodate stronger cryptographic methods, such as the recently approved Advanced Encryption Standard.

Security Associations — PKM is built around the concept of security associations (SAs). The SA is a set of cryptographic methods and the associated keying material; that is, it contains the information about which algorithms to apply, which key to use, and so on. Every SS establishes at least one SA during initialization. Each connection, with the exception of the basic and primary management connections, is mapped to an SA either at connection setup time or dynamically during operation.

Cryptographic Methods — Currently, the PKM protocol uses X.509 digital certificates with RSA public key encryption for SS authentication and authorization key exchange. For traffic encryption,the Data Encryption Standard (DES) running in the cipher block chaining (CBC) mode with 56-bit keys is currently mandated. The CBC initialization vector is dependent on the frame counter and differs from frame to frame. To reduce the number of computationally intensive public key operations during normal operation, the transmission encryption keys are exchanged using 3DES with a key exchange key derived from the authorization key.

The PKM protocol messages themselves are authenticated using the Hashed Message Authentication Code (HMAC) protocol [6] with SHA-1 [7]. In addition, message authentication in vital MAC functions, such as the connection setup, is provided by the PKM protocol.

Summary and Conclusion

The WirelessMAN™ air interface specified in IEEE Standard 802.16 provides a platform for the development and deployment of standards-based metropolitan area networks providing broadband wireless access in many regulatory environments. The standard is intended to allow for multiple vendors to produce interoperable equipment. However, it also allows for extensive vendor differentiation. For instance, the standard provides the base station with a set of tools to implement efficient scheduling. However, the scheduling algorithms that determine the overall efficiency will differ from vendor to vendor and may be optimized for specific traffic patterns. Likewise, the adaptive burst profile feature allows great control to optimize the efficiency of the PHY transport. Innovative vendors will introduce clever schemes to maximize this opportunity while maintaining interoperability with compliant subscriber stations.

The publication of IEEE Standard 802.16 is a defining moment in which broadband wireless access moves to its second generation and begins its establishment as a mainstream alternative for broadband access. Through the dedicated service of many volunteers, the IEEE 802.16 Working Group succeeded in quickly designing and forging a standard based on forward-looking technology. IEEE Standard 802.16 is the foundation of the wireless metropolitan area networks of the next few decades.

Acknowledgments

As lead PHY editors of IEEE Standard 802.16-2001, Jay Klein and Lars Lindh played key roles in the completion of the 10–66 GHz physical layer discussed here. Mr. Klein also chaired the PHY Task Group that led its development. Mr. Lindh and Ken Peirce each provided a helpful technical review of this manuscript.

References

[1] IEEE 802.16-2001, "IEEE Standard for Local and Metropolitan Area Networks — Part 16: Air Interface for Fixed Broadband Wireless Access Systems," Apr. 8, 2002.
[2] *IEEE P802.16a/D3-2001: "Draft Amendment to IEEE Standard for Local and Metropolitan Area Networks — Part 16: Air Interface for Fixed Wireless Access Systems — Medium Access Control Modifications and Additional Physical Layers Specifications for 2–11 GHz,"* Mar. 25, 2002.
[3] IEEE 802.16.2-2001, "IEEE Recommended Practice for Local and Metropolitan Area Networks — Coexistence of Fixed Broadband Wireless Access Systems," Sept. 10, 2001.
[4] SCTE DSS 00-05, Data-Over-Cable Service Interface Specification (DOCSIS) SP-RFIv1.1-I05-000714, "Radio Frequency Interface 1.1 Specification," July 2000.
[5] *SCTE DSS 00-09, DOCSIS SP-BPI+-I06-001215, "Baseline* Privacy Plus Interface Specification," Dec. 2000.
[6] *H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-*Hashing for Message Authentication," IETF RFC 2104, Feb. 1997.
[7] Federal Information Processing Standards Publication 180–1, "Secure Hash Standard," Apr. 1995.

Biographies

CARL EKLUND (carl.eklund@nokia.com) is a senior research engineer with Nokia Research Center, Helsinki, Finland. He chaired the MAC Task Group that developed the IEEE 802.16 medium access control protocol and served as a lead MAC editor of IEEE Standard 802.16-2001. He received his M.Sc. in engineering physics from Helsinki University of Technology in 1996. He is currently a guest researcher at the National Institute of Standards and Technology (NIST), Boulder, Colorado.

ROGER B. MARKS [F] (marks@nist.gov) is with NIST, Boulder, Colorado. In 1998 he initiated the effort that led to the IEEE 802.16 Working Group on Broadband Wireless Access, chairing it since inception. He served as technical editor of IEEE Standards 802.16-2001 and 802.16.2-2001. He received his A.B. in physics in 1980 from Princeton University and his Ph.D. in applied physics in 1988 from Yale University. Author of over 80 publications, his awards include the 1995 IEEE Morris E. Leeds Award (an IEEE Technical Field Award) and the Broadband Wireless Hall of Fame. He developed the IEEE Radio and Wireless Conference and chaired it from 1996 through 1999.

KENNETH L. STANWOOD (ken@ensemble.com) is currently principal member of technical staff and manager of systems engineering at Ensemble Communications, San Diego, California, where he was the primary designer of the MAC and transmission convergence layers of Ensemble's proprietary Adaptix™ broadband wireless access system. He graduated with a B.S. degree in mathematical sciences from Oregon State University in 1983 and an M.S. in computer science from Stanford University in 1986. He has been heavily involved in the IEEE 802.16 10–66 GHz project (serving as a lead MAC editor) as well as its European counterpart, ETSI BRAN HIPERACCESS. He is technical working group chair for the Worldwide Interoperability for Microwave Access (WiMAX) Forum, which is dedicated to producing test specifications and system option profiles to ensure interoperability of systems built to IEEE Standard 802.16. He chairs 802.16's Task Group c, which is creating profiles for 10–66 GHz 802.16 systems.

STANLEY WANG (stanley@reddotwireless.com) is currently a director of RedDot Wireless Inc., San Jose, California, where he is leading the development and implementation of MAC protocols. Before joining RedDot Wireless, he was with Ensemble Communications Inc., where he worked on IEEE 802.16 standards, serving as a lead MAC editor of IEEE Standard 802.16-2001. He received his Ph.D. in computer engineering in 1994 from the University of Southern California. He was with the faculty of the Computer Science Department at California State University-San Marcos from 1994 through 2000, where he received the Harry E. Brakebill Outstanding Professor Awards in 1996 and his tenure in 1999.

Through the dedicated service of many volunteers, the IEEE 802.16 Working Group succeeded in quickly designing and forging a standard based on forward-looking technology. IEEE Standard 802.16 is the foundation of the wireless metropolitan area networks of the next few decades.