Numbat – WiMax, DHCPv6 implementation in Omnet++

Andrzej Bojarski
Maciej Jureko
Tomasz Mrugalski

2009-01-18

0.0.0

Contents

1 Introduction

Numbat is a simulation model for IEEE 802.16 [2], better known under its commercial name Mobile WiMax. As its original purpose was to simulate IPv6 mobile node, with focus on the (re)configuration processes, stateless (RA, DAD) and stateful (DHCPv6) configuration methods are also part of this model.

Following features are supported in the WiMAX layer:

- based on IEEE 802.16-2005 (aka 802.16e)
- Simple radio/PHY layer - unicast (SS to BS) and multicast (BS to SS) transmissions
- OFDMA transmission (including CDMA codes, radio frame slots/symbols)
- Bandwidth management: BWR, CDMA codes for ranging and BWR transmission)
- Supported traffic classes: Best Effort (BE) and Unsolicited Grant Service (UGS)
- Control plane: Network entry(RNG-REQ, RNG-RSP, SBC-REQ, SBC-RSP, PKM-REQ, PKM-RSP, REG-REQ, REG-RSP messages)
- Control plane: Service flow creation/mgmt (DSA-REQ, DSA-RSP, DSA-ACK)
- Control plane: Scanning (SCN-REQ, SCN-RSP)
- Control plane: Handover (BSHO-REQ, MSHO-RSP, HO-IND)
- Several traffic models: fixed, handover after timeout, distance based handover
- Simple event signalling, so you don't have to understand whole stack operation to get some notification (similar to MIH 802.21-style events, but easier)
- Several optimization allowed by 802.16-2005 standard are configurable (you can enable or disable them)
- multiple SSes support (optimizations can be enabled on a per SS basis)
- Connection management and queuing
- multiple BSes support
- Handover between BSes can be simulated easily

On top of WiMAX layer, there is a working IPv6 stack. As primary purpose of this simulation environment is focused automatic configuration and DHCPv6 in particular, related areas are the most developed:

- IPv6Node: traffic source/sink, with statistics and various traffic generation models
- RAGen/RArcv: Router Advertisement generator and receiver, used to configure IPv6 nodes in stateless mode
- DHCPv6 client: implementation of the DHCPv6 client, with several proposed enhancements
- DHCPv6 server: as you probably guessed, it is used for providing configuration. Supports relays and several extra enhancements
- MobileIPv6 Mobile Node: Simple implementation of the MN
- MobileIPv6 Home Agent
- IPv6 dispatcher, that inteligently redirects packets to specific modules. It is also mobility aware.

To better model complex environment, this simulation also provides event based FSM (Finite State Machine) implementation. For each state, there are up to 3 functions: onEnter(), onEvent() and onExit(). States can be transitive or stationary. Also there's a well defined list of inputs for each FSM.

Numbat code is available under GNU GPL (version 2 or later) licence. It means that it can be downloaded, compiled, used, modified and even redistributed by all users, including commercial purposes. Numbat uses the Omnet++ as simulation environment [3][1]

[1] Omnet++ is distributed under other license. See its homepage for details.

2 Installation and usage

Numbat can be used in Windows and Linux systems. Ways of installation are different. As Numbat is not a end user software, but rather research environment, you need to gain a bit of expertise to use it. Do not expect to have download and ready to use product anytime soon.

2.1 Linux compilation

2.1.1 Requirements

To compile and run Numbat you will need properly installed Omnet++ (tested with version 3.3). Omnet++ packages and installation notes are available on the Omnet Community website: http://omnetpp.org.

2.1.2 Installation

First you need to obtain Numbat sources from http://klub.com.pl/projects/numbat/. You can download latest snapshot or sources directly from svn repository.

Before compilation you must prepare Makefile, execute command opp_makemake may be used. However, as sources are distributed over serval directories, it is better to use provided script. Then make all targets by simply running make:

```
./rebuild-makefiles
make
```

We suppose that there weren't any errors, so you can run and enjoy simulation:

```
cd wimax
./wimax
```

2.2 Windows compilation

Note: This section is based on document "Omnet++ installation and configuration guide in WindowsÂŽ environment." by Andrzej Bojarski.

This guide shows how to install and configure Omnet++ in Windows environment. This guide is mainly based on official Omnet++ manual, but also contains some knowledge based on author's own experiences. It does not cover all possible installation scenarios. Though it should be treated as helpful source rather than complete guide.

First thing to consider is that Windows version of Omnet++ is optimized for and should be used with Visual C++. As full version it is a commercial product. However there are free versions like Visual Studio 2005 Express Edition which are fully sufficient for Omnet++ uses. Also installing Platform SDK, which is available free too, is necessary.

Omnet++ comes as an ordinary .exe installation file so it is obvious for every windows user how to make use of it. Only thing that needs explanation is environment selection. You have to choose VC++ release you are using. If it's Visual C++ 2003 (7.1) you choose vc-71m, if it's Visual C++ 2005 (8.1) you choose vc-81. It is not recommended to use Visual Studio 6.0 because some of Omnet++ extensions have problem to compile with it.

Installation directory should not have blank spaces as it may cause compilation problems.

2.2.1 Pre-compilation setup

Before making any operations using Omnet++ it is obligatory to set right system variables. They must be set for every new command line window used for compilation. In other words, first thing You have to do after opening the command line window is setting the system variables. It is done by executing vcvarsall.bat or vcvars32.bat files from VC++ installation directories. These files can usually found in `C:\Program Files\Microsoft Visual Studio 8\VC` or similar (depending on VC++ release) directory. It is a good practice to make a .bat file executing system variables setting to save time wasted for walking through directories.

On this stage Omnet++ should be fully functional, however sometimes there is a compilation and/or linker problem with libraries placement. That was my case. Moving all VC++ libraries to Omnet++ include and lib directories turned out to be helpful. However there are other problems with configuration based on library compatibility issues. If you get such errors, visit http://www.omnetpp.org/pmwiki/index.php?n=Main.OmnetppInstallation to get latest

istallation tips. Main advice for such incompatibility is to change version of VC++ as some versions tend to work incorrectly with Omnet++.

2.2.2 Actual compilation

To create Makefile.vc which is VC++ file used for further compilation run following command:

```
\your\working\directory>opp\nmakemake -f
```

Â This will create new Makefile.vc, overwriting previous existing. Next, compile files into executable:

```
\your\working\directory>nmake -f Makefile.vc
```

Assuming you won't get any errors, .exe file should be made.

3 Features HOWTO

This section describes various aspects of the Numbat implementation.

3.1 Finite State Machine

Numbat does not use FSM framework provided by the Omnet++ environment. Decision to go ahead with own FSM implementation was based on several factors:

- Omnet's FSM are does not allow to stay in the same state. Existing workaround to exit and enter the same state is not sufficient.
- There are no clearly defined events in Omnet's FSM.
- Omnet's FSM must be implemented as object instantiated inside another object, which is derived from cSimpleModule.
- There are not useful timer support.

There are several things that must be done in the header (.h) file:

- Declare class (e.g. WMaxCtrlSS) derived from a Fsm class.
- Define list of states (as an enum).
- For each stationary state define onEventState, which will decide what to do, when FSM is in a particular state.
- It is possible to define 2 additional functions: onEnterState and onExitState.
- If the state is transitive (i.e. FSM does not end in this state, but rather does some action and then switches to another state immediately), onEnterState and onExitState is mandatory.
- Define list of events (also a enum).
- (optional) Define timers. Each defined timer can be started, when certain conditions are met (e.g. frame is sent and something must be done after some time). When timeout is reached, self message will be sent. Timers can also be stopped at any time.

Below an example of a simple FSM machine implementation is presented.

```
// WMaxCtrlSs.h file
#include "fsm.h"

class WMaxCtrlSS : public Fsm
{
public:
    WMaxCtrlSS();
```

```
    void initialize();
    void handleMessage(cMessage *msg);

protected:
    void fsmInit();

    // --- STATES ---
    typedef enum {
        STATE_OPERATIONAL,       // normal operation
        STATE_SEND_MSHO_REQ,     // send MSHO-REQ message
        STATE_WAIT_BSHO_RSP,     // wait for BSHO-RSP message
        STATE_SEND_HO_IND,       // send HO-IND message
        STATE_HANDOVER_COMPLETE, // handover complete
        STATE_NUM
    } State;

    // operational state
    static FsmStateType onEventState_Operational(Fsm * fsm, FsmEventType s, cMessage *msg);

    // send MSHO-REQ state
    static FsmStateType onEnterState_SendMshoReq(Fsm *fsm);

    // handover complete state
    static FsmStateType onEventState_HandoverComplete(Fsm * fsm, FsmEventType s, cMessage *msg);

    // --- EVENTS ---
    typedef enum {
        EVENT_HANDOVER_START,
        EVENT_BSHO_RSP_RECEIVED,
        EVENT_NUM
    } Event;

    // --- TIMERS ---
    TIMER_DEF(Handover);
};
```

When the class declaration is complete, several things must be specified in the .cc file:

- In the fsmInit() method: `statesEventsInit(X, Y, Z)` must be called. X denotes maximum number of states, Y number of events and Z is a initial state of the state machine.
- Initialize all defined states. There are 2 possible state types: stationary and transitive.
- For each stationary state, name and onEventState function must be defined. That function will decide what to do when certain event has been received.
- For each transitive state, it is necessary to define name and target state. Also `onEnterState` and `onExitState` functions must be defined (those function are optional for stationary states).
- Timers can also be initialized here.
- handleMessage() method contains a „dispatcher". It translates received messages into events, e.g. if received frame is of type `WMaxMsgBSHORSP`, then execute `onEvent(EVENT_BSHO_RSP_RECEIVED)`.
- initialize() method should call fsmInit() and also may contain additonal initialization code, e.g. timers can be started here.
- For each stationary state, there should be method called onEventState. It defines what exactly should be done when some event has occured. That method returns a newState. If no state change is necessary, it must contain following entry: `return fsm->State();`

- In the switch() command in the onEventState method, it seems reasonable to add `default: CASE_IGNORE(e)` line, which will print appropriate information in case of reception of an event, which is not supported in that state.
- It is possible to defined, what actions should be performed when entering (onEnter) or leaving (onExit) particular state.
- At any moment, `TIMER_START()` or `TIMER_STOP()` can be executed to start or stop timer.

```
Define_Module(WMaxCtrlSS);

WMaxCtrlSS::WMaxCtrlSS()
{

}

void WMaxCtrlSS::fsmInit() {

    // initialize number of states, number of elements and set initial state
    statesEventsInit(WMaxCtrlSS::STATE_NUM, WMaxCtrlSS::EVENT_NUM, STATE_OPERATIONAL);

    // state init
    stateInit(STATE_OPERATIONAL,   "Operational", onEventState_Operational);
    stateInit(STATE_SEND_MSHO_REQ, "Sending MSHO-REQ", STATE_WAIT_BSHO_RSP,
                                   onEnterState_SendMshoReq, 0);
    stateInit(STATE_WAIT_BSHO_RSP, "Waiting for BSHO-RSP", onEventState_WaitForBshoRsp);
    stateInit(STATE_SEND_HO_IND,   "Sending HO-IND", STATE_HANDOVER_COMPLETE,
                                   onEnterState_SendHoInd, 0);
    stateInit(STATE_HANDOVER_COMPLETE,"Handover complete", onEventState_HandoverComplete);
    stateVerify();

    // event init
    eventInit(EVENT_CDMA_CODE, "CDMA code received");
    eventInit(EVENT_HANDOVER_START, "Begin handover procedure");
    eventInit(EVENT_BSHO_RSP_RECEIVED, "BSHO-RSP received");
    eventVerify();

    TIMER(Handover, 0.1, "Start handover");
}

void WMaxCtrlSS::initialize() {
    // initiate FSM
    fsmInit();

    // Start handover timer
    TIMER_START(Handover);
}

void WMaxCtrlSS::handleMessage(cMessage *msg)
{
    //IF_TIMER(name, EVENT_TIMEOUT);
    if (msg==TimerHandover) {
        onEvent(EVENT_HANDOVER_START, msg);
        return;
    }

    if (dynamic_cast<WMaxMsgBSHORSP*>(msg)) {
```

```
        onEvent(EVENT_BSHO_RSP_RECEIVED, msg);
        return;
    }
}


FsmStateType WMaxCtrlSS::onEventState_Operational(Fsm * fsm, FsmEventType e, cMessage *msg)
{
    switch (e) {
    case EVENT_HANDOVER_START:
        return STATE_SEND_MSHO_REQ;
    default:
        CASE_IGNORE(e);
    }
}

// send MSHO-REQ state
FsmStateType WMaxCtrlSS::onEnterState_SendMshoReq(Fsm *fsm)
{
    WMaxMsgMSHOREQ * mshoReq = new WMaxMsgMSHOREQ("MSHO-REQ");
    mshoReq->setName("MSHO-REQ");
    ev << fsm->fullName() << "Sending MSHO-REQ message." << endl;
    fsm->send(mshoReq, "macOut");
    return fsm->State();
}

// wait for BSHO-RSP state
FsmStateType WMaxCtrlSS::onEventState_WaitForBshoRsp(Fsm * fsm, FsmEventType e, cMessage *msg)
{
    switch (e) {
    case EVENT_BSHO_RSP_RECEIVED:
        return STATE_SEND_HO_IND;
    default:
        CASE_IGNORE(e);
    }
    return fsm->State();
}

// sent HO-IND state
FsmStateType WMaxCtrlSS::onEnterState_SendHoInd(Fsm *fsm)
{
    WMaxMsgHOIND * hoInd = new WMaxMsgHOIND();
    hoInd->setName("HO-IND");
    fsm->send(hoInd, "macOut");
    return fsm->State();
}

// handover complete state
FsmStateType WMaxCtrlSS::onEventState_HandoverComplete(Fsm * fsm, FsmEventType s, cMessage *msg)
{
    return fsm->State();
}
```

References

[1] IEEE, "Part 16:Air Interface for Fixed Broadband Wireless Access Systems", IEEE Std 802.16d-2004, IETF, October 2004

[2] IEEE, "Part 16:Air Interface for Fixed and Mobile Broadband Wireless Access Systems", IEEE Std 802.16e-2005, IETF, February 2006

[3] http://www.omnetpp.org/, December 2006