Numbat – WiMax, DHCPv6 implementation in Omnet++

Andrzej Bojarski
Maciej Jureko
Tomasz Mrugalski

2007-08-23

0.0.0

Contents

1 Introduction

Numbat is a simulation model for IEEE 802.16 [2], better known under its commercial name Mobile WiMax. Due to full scale mobility simulation, also DHCPv6 is also part of this model. It is implemented in the Omnet++ environment [3].

The whole Numbat code is available under GNU GPL (version 2 or later) licence. It means that it can be downloaded, compiled, used and modifed by all users, including commercial purposes.

2 Installation and usage

Numbat can be used in Windows and Linux systems. Ways of installation are different, but both quite simple.

2.1 Linux installation

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 -f
```

To build Numbat, type:

```
make
```

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

```
./wimax
```

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