IEEE 802.11s Power Save Modes Support Implementation for Linux

Ivan Bezyazychnyy, Michael Krinkin, Dmitry Yuranov, Pavel Zubarev, Alexander Vereshchagin, Kirill Krinkin
Open source and Linux Lab
Saint-Petersburg, Russia
{ivan.bezyazychnyy, krinkin.m.u, d.yuranov, pavel.zubarev, aleverr}@gmail.com, kirill.krinkin@fruct.org

Abstract

Wireless mesh networks (WMN) have attracted considerable interest in recent years as a convenient, flexible and low-cost alternative to wired communication infrastructures in many contexts. However, in this type of networks mobile nodes use batteries and their energy resources are very limited. For this reason IEEE 802.11s mesh standard has power save mechanisms which aim is to utilize energy efficiently. In this paper, we describe the power save modes implementation of these mechanisms in Linux kernel. We displayed our current state of implementation and discussed future plans.

Index Terms: IEEE 802.11s, mesh, power save, linux, wireless.

I. Introduction

Wireless networks (WLAN) are based on the IEEE 802.11 standards and become increasingly popular in the computing industry. They are organized in two ways. The first one is known as infrastructure networks, i.e. those networks have fixed bridges with wired gateway. The bridges of these networks are known as base stations. A mobile node within these networks connects to, and communicates with, the nearest base station that is within its communication radius.

The second type of mobile wireless network is the infrastructure-less mobile network, commonly known as ad hoc networks. All nodes in these networks are capable of movement and can be connected dynamically in an arbitrary manner.

A special type of wireless ad hoc networks are wireless mesh networks (WMN). IEEE 802.11s is a draft IEEE 802.11 amendment for mesh networking. It defines how wireless devices can interconnect to create a WLAN mesh network, which may be used for static topologies and ad hoc networks. A wireless mesh network often has a more planned configuration, and may be deployed to provide dynamic and cost effective connectivity over a certain geographic area. It is often assumed that all nodes in a wireless mesh network are immobile, but this may not be the case. The mesh routers may be mobile, and be moved according to specific demands arising in the network.

Wireless mesh architecture is the first step towards providing cost effective and dynamic high-bandwidth networks over a specific coverage area.

Example of three types of wireless mesh network:

- Infrastructure wireless mesh networks: Mesh routers form an infrastructure for clients.
- Client wireless mesh networks: Client nodes constitute the actual network to perform routing and configuration functionalities.
- Hybrid wireless mesh networks: Mesh clients can perform mesh functions with other mesh clients as well as accessing the network.

An 802.11s device is labeled as Mesh Station (mesh STA). Mesh STAs form mesh links with one another, over which mesh paths can be established using a protocol. 802.11s defines a default mandatory routing protocol (Hybrid Wireless Mesh Protocol or HWMP), yet allows vendors to operate using alternate protocols. HWMP is inspired by a combination of AODV and tree-based routing.

Mesh STAs are individual devices using mesh services to communicate with other devices in the network. They can also collocate with 802.11 Access Points (APs) and provide access to the mesh network to 802.11 stations (STAs), which have broad market availability. Also, mesh STAs can collocate with an 802.11 portal that implements the role of a gateway and provides access to one or more non-802.11 networks. In both cases, 802.11s provides a proxy mechanism to provide addressing support for non-mesh 802 devices, allowing end-points to be cognizant of external addresses. 802.11s also includes mechanisms to provide deterministic network access, a framework for congestion control and power save.

In WMNs, battery energy at the nodes is a very limited resource that needs to be utilized efficiently. The failure of some nodes' operation can greatly impede performance of the network and even affect the basic availability of the network. The potential problem in current protocols for WMNs is that they find the shortest path and use that path for every communication. However, that is not the best thing to do for network lifetime. Using the shortest path frequently leads to energy depletion of the nodes along that path and in the worst case may lead to network partitioning.

Below, in Section 2, we briefly describe the key features of the power saving modes in IEEE 802.11s networks. In Section 3 we provide some details of our implementation of power saving modes support in Linux kernel. And in Section 4 we introduce our testing results of the our power saving implementation.

II. DESCRIPTION OF IEEE 802.11s POWER SAVE MODES

In this section, we briefly describe the key features of the power saving modes in IEEE 802.11s networks.

A mesh STA can be in one of two different power states: *Awake* and *Doze* states.

The manner in which a mesh STA transitions between power states is determined by the combination of mesh peering specific mesh power modes. A mesh STA shall be in Awake state if any of its mesh peerings requires operation in Awake state.

Mesh power mode represents activity level of a mesh STA per peering. Three power modes are defined in IEEE 802.11s [1] standard:

- Active mode;
- Light sleep mode;
- Deep sleep mode.

Each mode defines how a mesh STA shall operate with other peer nodes. In *active mode* a mesh STA shall be in Awake state all the time. In *light* and *deep sleep modes* a mesh STA alternates between Awake and Doze states, as determined by the frame transmission and reception rules. But in deep sleep mode a mesh STA may choose not to listen to the beacons from its peer mesh STA. Operation in light or in deep sleep mode is optional.

The used mesh power mode shall be indicated by the Power Management field and Mesh Power Save Level field. Values of these fields are represented in Table 1.

TABLE I
MESH POWER MODES

Activity level	Mesh Power Mode	Power Management field	Mesh Power Save Level field
Highest	Active Mode	0	Reserved
Middle	Light sleep Mode	1	0
Lowest	Deep sleep Mode	1	1

To change its mesh power mode for a link to a higher activity level a mesh STA may use group addressed or individually addressed Mesh Data or QoS Null frames. Individually addressed frames may be used to temporarily raise the activity level of the mesh STA for a mesh peering.

A mesh STA shall use acknowledged individually addressed Mesh Data or QoS Null frames to change its mesh power mode for a link to a lower activity level . If a mesh decided to change its *own* power mode to a lower level it has to use proper values of Power Management and Mesh Power Save field in *all* transmitted group addressed frames.

A mesh STA shall always indicate its own mesh power mode for each mesh peering node and obtains the mesh power nodes of its peer mesh STAs. A mesh STA shall not arbitrary transmit frames to mesh STAs operating in a light or deep sleep mode, but shall buffer frames and transmit them only at designated times.

If a mesh STA wants to identify that there is a pending or buffered traffic for the peering mesh STA operating in sleep mode it uses the TIM element. There are two different TIM types: TIM and DTIM. A mesh STA shall transmit a TIM with every Beacon frame. And every DTIMPeriod, a DTIM element is transmitted with a Beacon frame. After sending a DTIM the mesh STA shall send the buffered group addressed MSDUs and MMPDUs, before transmitting any individually addressed frames. For transmitting a buffered frames STA uses Mesh Awake Window and Peer Service Period.

A mesh STA in light or deep sleep mode shall be in Awake state during its own Mesh Awake Window. Mesh Awake Window is a period of time where the mesh STA operates in Awake state after its Beacon Probe Response frame transmission that contained the Mesh Awake Window element. Beacon frame with Mesh Awake Window element from a peer STA indicates that there are a buffered traffic for a mesh STA.

The start of the Mesh Awake Window is measured from the end of the Beacon or Probe Response transmission and the duration of its period is defined by the value of the Mesh Awake Window field in this element.

When a mesh STA turns into Awake state during Mesh Awake Window, it may send a trigger frame in order to initiate *a peer service period.* This contiguous period is used for successfull transmission buffered frames towards mesh STAs that operates in light or deep

sleep mode. Peer service periods are not used in frames exchanges towards mesh STAs operating in active mode.

There are three phases of a peer service period: initiation, operation and termination. One mesh STA operates as transmitter in peer service period, transmits frames and initiates the termination of the peer service period. The other mesh STA operates as receiver in peer service period and receives the frames. There are multiple peer service periods towards peer mesh STAs ongoing in parallel. At most one peer service period can be set up in each direction with each peer mesh STA.

To initiate a peer service period a mesh STA may use a Mesh Data frame or a QoS Null frame that requires acknowledgment as a peer trigger frame. The phase of peer service period is determined by combination of RSPI and EOSP fields combination in peer trigger frame.

The peer service period may be initiated in the following cases:

- The mesh STA in sleep mode, during Mesh Awake Window, receives a peer trigger frame with defined combination of RSPI and EOSP fields;
- The mesh STA in active mode receives a peer trigger frame from the peer mesh STA in sleep mode;

During the peer service period, the transmitter and receiver STAs shall operate in Awake state. After a successful transmission of frames the peer service period may be terminated. It can be accomplished by a successfully acknowledged QoS Null or Mesh Data frame with EOPS bit set to 1 from the transmitter of the peer service period. If the mesh STA does not receive an acknowledgment, the transmitter shall retransmit that frame at least once within the same peer service period. The maximum number of retransmissions is the lesser of the Max Retry Limit (not defined in the standard).

Based on latest IEEE 802.11s standard the power saving mechanism can be sorted out as next phases of implementation:

- indication of mesh STA' power modes in frames;
- a buffering and identification sending frames in mesh STA;
- Mesh Awake Window implementation;
- Peer service period;

III. Implementation details

Our ultimate goal was to implement power save modes in Linux kernel according to IEEE802.11s standard. After we compared Linux kernel sources and IEEE802.11s standard we discovered issues in power save modes implementation and defined the ways to solve them.

The first issue was to add indication of peer and non-peer mesh power modes in relevant frames. This has been done in mac80211 subsystem. In transmission function ieee80211_xmit power mode is checked and Power Management field of frame control and Mesh Power Save Level field of QoS control field are set if necessary to indicate peer power mode. To indicate non-peer power mode Power Management field of frame control is set up in ieee80211_beacon_get_tim function during header initialization if power mode is not active and Mesh Power Save Level field is set up in mesh_mgmt_ies_add function during mesh capability installation if power mode is the deep sleep.

Unicast data frames and QoS Null frames are tracked to record current peer power mode in ieee80211_rx_h_sta_process function. This function is one of the handlers called in receive path flow. Power Management field of frame control and Mesh Power Save Level field of QoS control field are tested and peer power mode is indicated in sta_info structure. sta_info structure represents and any station (peer) in mac80211. This structure could represent mesh peer, IBSS peer, AP, WDS peer. To keep peer power save mode for mesh stations peer_ps_mode field is added to this structure.

When a new peer link opens initial values for local and peer power modes has to be set. Local link-specific power mode for this new link is set up equal to local non-peer power mode value. Peer link-specific power mode for this new link is supposed equal to peer's non-peer power mode value. This settings are done in mesh_neighbour_update function.

Changes also has been done in cfg80211 subsystem and in nl80211 interface. cfg80211 is the configuration API for 802.11 devices in Linux and a subsystem of Linux kernel. It bridges userspace and drivers, and offers some utility functionality associated with 802.11. By means of cfg80211 Linux kernel offers a consistent API through nl80211. Changes to this subsystem are aimed to:

- setting up local link-specific power modes;
- getting and setting local and peer link-specific mesh power modes;
- getting and setting mesh non-peer power mode.

Despite the kernel space there is also a user space from where we would like to track and to manage our power modes. iw is a nl80211 based command line interface configuration utility for wireless devices. It uses libnl to interact with kernel through nl80211 interface. As we have new parameters in cfg80211 component we had to add ability to work with them from userspace using iw utility. Changes can be divided in two parts: working with non-peer power mode and working with link-specific power modes.

To work with non-peer power modes with iw utility _mesh_param_descrs array of mesh_param_descr structures was added by new one. This structure defines the name and the nl80211 type of new parameter. Also two functions to parse and to print power modes was written and used in this structure. To work with link-specific power modes station handler defined in station.c file has been updated.

Nevertheless a lot of work is to be done. Mesh station should buffer mesh data frames for sleeping peers, make indication of buffered unicast frames in TIM bitmap and group addressed buffered frames in DTIM. It should deliver group addressed buffered frames after DTIM. Power saving scheme proposed be IEEE 80211s standard in this case closely resemble infrastructure mode scheme. Our approach is to look at infrastructure mode realization and to reuse parts of code. Another not implemented issue is mesh station should support and indicate the Mesh Awake Window. Also peer service period logic described in standard need to be added.

IV. TESTING RESULTS

This section describes a current results of our mesh power save implementation. All changes we made to Linux Kernel must be verified. To make sure that our mesh power save implementation works properly we need to test a mesh network. Testing process contains several steps. First of all it is needed to build and configure Linux kernel containing all out patches. After the kernel successfully installed it is need to setup mesh network. The last step was testing power save modes using iw utility.

A. Build kernel

Mesh support comes from 2.6.26 Linux kernel, so it's able to use latest promoted kernel. Hwsim tool is used for testing the mesh stack. Now the hwsim userspace extension have been accepted in wireless-testing kernel development tree. For making things easy it is needed to check /boot/ directory for available kernel's configuration:

```
zps@thinkers:~$ ls -la /boot/ | grep config
-rw-r--r--   1 root root   109918 May 21 21:31 config-2.6.35.13-92.fc14.x86_64
-rw-r--r--   1 root root   109918 Aug 17 01:20 config-2.6.35.14-95.fc14.x86_64
-rw-r--r--   1 root root   109918 Sep  1 16:04 config-2.6.35.14-96.fc14.x86_64
```

These configuration are used on your PC and can be used for new kernel. After the configuration of the kernel is finished it is need to make sure that next features are checked.

```
zps@thinkers:~/ws/original_kernel/linux$ grep MAC80211 .config
CONFIG_MAC80211=m
CONFIG_MAC80211_MESH=y
CONFIG_MAC80211_HWSIM=m
```

For testing purposes it is needed to apply “Send-To-Self (loop)” [2] patch which implements routing of traffic between local IP addresses externally via ethernet interfaces. After that the kernel was build and installed on system.

B. Setup Mesh

In our testbed we created three devices using this simple script:

```
#!/bin/bash

modprobe mac80211
modprobe mac80211_hwsim radios=3

for i in `seq 1 3`
do
    iw phy phy$i interface add mesh$i type mp mesh_id osll
    echo 1 > /proc/sys/net/ipv4/conf/mesh$i/loop
    ifconfig mesh$i 192.168.70.1$i
done
```

Each created device doesn't have any hardware, but is represented in the system with according device. This script uses physical devices from 1 til 3 because phy0 is usually a wireless card on a PC.

C. Testing the network

To test the network *iw* utility was used. Several commands were running to make sure that nodes have connection between each other.

```
[root@thinkers original_kernel]# iw dev mesh7 station dump | grep -e Sta -e plink
Station 42:00:00:00:02:00 (on mesh7)
            mesh plink:     ESTAB
```

```
Station 42:00:00:00:01:00 (on mesh7)
        mesh plink:     ESTAB
[root@thinkers original_kernel]# iw dev mesh8 station dump | grep -e Sta -e plink
Station 42:00:00:00:02:00 (on mesh8)
        mesh plink:     ESTAB
Station 42:00:00:00:00:00 (on mesh8)
        mesh plink:     ESTAB
[root@thinkers original_kernel]# iw dev mesh9 station dump | grep -e Sta -e plink
Station 42:00:00:00:01:00 (on mesh9)
        mesh plink:     ESTAB
Station 42:00:00:00:00:00 (on mesh9)
        mesh plink:     ESTAB
```

ESTAB means that nodes is connected with its peers. After that an ability of sending packets between nodes was checked.

```
[root@thinkers original_kernel]# iw dev mesh7 mpath dump
DEST ADDR         NEXT HOP          IFACE      SN      METRIC  QLEN    EXPTIME         DTIM
DRET    FLAGS
42:00:00:00:02:00 42:00:00:00:02:00 mesh7      2       8193    0       1404557608      0       0
0x14
42:00:00:00:01:00 42:00:00:00:01:00 mesh7      1       911     0       1404557608      0       0
0x10
[root@thinkers original_kernel]# ping -I mesh7 192.168.70.19  -f -c 100
PING 192.168.70.19 (192.168.70.19) from 192.168.70.17 mesh7: 56(84) bytes of data.

--- 192.168.70.19 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 0.022/0.035/0.777/0.076 ms, ipg/ewma 0.069/0.023 ms
```

And final step was to test that power save modes are set. Before that it is needed to check that enum nl80211_meshconf_params in nl80211.h in iw and the kernel are identical.

```
[root@thinkers iw]# ./iw dev mesh5 get mesh_param mesh_power_mode
active
```

Running this command will show what power mode is used by mesh STA.

As we can see creating and testing a mesh network is boring and tedious task which can be automated. In order to automate process of testing we have developed a testing script. The script gets one optional parameter - the number of mesh points to use. By default it uses three mesh points. If testing succeeds script returns zero value and it exits with nonzero error code if an error has occurred during testing process. Our script performs the following actions. Firstly, it reloads mac80211_hwsim kernel module creating required number of "physical" devices. Then it sets up mesh interface for each device and links these devices to mesh network. After that, it checks that every mesh point is connected and able to ping each other. Finally, it tests power save modes for all created devices.

V. Conclusion

In this paper we described power save modes support implementation of IEEE 802.11s networks in Linux kernel.

The main goal of our project is to implement energy saving in IEEE 802.11s Mesh networks. One of the main issue was to determine how the power saving mechanisms in Mesh networks work. At the moment support of these mechanisms haven't implemented yet.

As the base of exploring this problem we chose the mesh implementation in Linux kernel. The latest Linux kernel has a working version of Mesh network that is maintained by several network drivers including virtual wireless driver mac80211_hwsim.

Based on the standard we have implemented the first part - indication of mesh STA' power modes in frames. We are currently implementing the support buffering frames in mesh STA and ability a mesh STA to identify peer nodes about presence a buffered frames.

In the next phase we plan to extend our implementation with Mesh Awake Window and Peer Service Period. The last and final phase will be developing a driver notification system which aims to schedule Awake and Doze states for active and sleep modes.

REFERENCES

[1] IEEE, “Draft amendment: ESS mesh networking,” IEEE P802.11s Draft 12.00, June 2011.
[2] Linux Patches http://www.ssi.bg/~ja/#loop
[3] mac80211_hwsim tool http://linuxwireless.org/en/users/Drivers/mac80211_hwsim
[4] iw utility http://linuxwireless.org/en/users/Documentation/iw