

1. Functions of the traffic monitor module.

The traffic monitor module counts the used data value in certain time, gives a warning when the used data value reaches the warning percent, and disconnects the network when all available data is used up.

1.1 Ways of Traffic Management

The module supports three ways to manage the traffic: basic monthly; idle time; period of time.

1.1.1 Basic Monthly.

The used data is counted by month. And the month may be by the natural month or month with a pay day.

1.1.2 Idle time

The data flow used in idle time is much cheaper than in non idle time. The idle time is from one hour to another hour, e.g. 23:00~7:00.

The data used in idle time is counted by the natural or month with a pay day.

1.1.3 Period of time

The period of time is from the start date to the end date, e.g. one season, half a year.

1.2 Interfaces

1.2.1. Get the current used data

The module supplies APIs to get the current used data in basic month, in idle time (if on), in period of time.(if on)

1.2.2. Set available data

The module supplies APIs to set available data in basic month, in idle time and in period of time.

1.2.3. Set warning on/off

Users can set a warning percent: when used data reaches this percent, a warning flag will be set.

1.2.4. Adjust the used value

The data counted by this module may be not match with actual value. This module supplies APIs to set the accurate value.

1.2.5. Set disconnected function when used up.

When the used value is used up, the network can be disconnected or not according to the settings.

1.2.6. Get the daily used value.

The daily used data value includes the data used in this day; no matter the data is counted into basic monthly, idle time or period of time.

1.2.7 Set idle time settings (on/off, start/end hour).

1.2.8 Set the period of time settings. (On/off, start/end date)

1.2.9. Get the speed rate at given interval.

2. Get Traffic Statistics from OpenWRT

The data which go through one net interface(e.g. etho) can be read from "sys/class/net/if_name/statistics/"

```
root@OpenWrt:/sys/class/net# ls
br-lan  ccinet1  ccinet3  ccinet5  ccinet7  lo      uap0
ccinet0  ccinet2  ccinet4  ccinet6  ip6tnl0  tunl0  usbnet0
root@OpenWrt:/sys/class/net#
```

The files under '/sys/class/net/if_name/statistics':

```
root@openwrt:/sys/devices/virtual/net/ccinet4/statistics# ls
collisions      rx_frame_errors  tx_compressed
multicast        rx_length_errors tx_dropped
rx_bytes         rx_missed_errors tx_errors
rx_compressed    rx_over_errors   tx_fifo_errors
rx_crc_errors    rx_packets        tx_heartbeat_errors
rx_dropped       tx_aborted_errors tx_packets
rx_errors        tx_bytes          tx_window_errors
rx_fifo_errors   tx_carrier_errors
root@openwrt:/sys/devices/virtual/net/ccinet4/statistics#
```

rx_bytes: received bytes

tx_bytes: transferred bytes

rx_packets: received packets

tx_packets: transferred packets

These files are refreshed by kernel.

The wan interface created by Dialer is named as ccinet*. And there may be several net interface (multiple PDP). The traffic module gets the current active net interface created by PDP from dialer module.

3. Data Base/Config — UCI file

3.1 Daily used data(basic month + idle time + period of time).

The daily used value is stored in the file '/etc/stat/daily/year-mon'

config daily_use

option day '1' /*the day*/

option value '3213' /*rx_bytes + tx_bytes*/

config daily_ues

option day '2' /*the day*/

option value '32421' /*rx_bytes + tx_bytes*/

3.2 daily used data in idle time

The daily used value is stored in the file '/etc/stat/idle_daily/year-mon'

The format of UCI file is the same as files in chapter 3.1.

3.3 daily used data in period of time

The daily used value is stored in the file '/etc/stat/period_daily/year-mon'

The format of UCI file is the same as files in chapter 3.1.

3.4 The config of traffic monitor module

The config file of this module is stored in the file '/etc/config/traffic'

config basic_mon

option enable 1

option avl_data '13245' /*available data value*/

config period

option enable 0

```

option    avl_data '43234'
option    start_date '2015-4-7'
option    end_date '2015-7-7'
config    idle
option    enable '0'
option    avl_data '0'
option    start_hour '23'
option    end_hour '7'
config    general
option    warn_enable '1'
option    warn_percent '90'
option    dis_enable '1'
option    'dis_percent' '100'

```

4. Data Struct

```

/*this struct is used to controll statistics warning*/
struct _stat_ctrl_warn{
    int _warn_enable; // 0 -- needn't; 1 -- need
    int _warn_percent; //0~100, warning percent
    int _warninged; // have warned
    int *_warn_handler(); //when warning, call this handler
    int *_reset_handler(); // reset waring, could warn again
    int *_check_warn_handle(); // check whether reached warning data
};

/*this struct is used to controll statistics disconnect/reconnect*/
struct _stat_ctrl_disconn{
    int _disconn_enable; // 0 -- needn't; 1 -- need
    int _disconn_percent; //0~100, disconn percent
    int *_disconn_handler(); // when reaching limit, call this handler
    int *_reset_handler(); // when _disconn_need or _disconn_percent is changed
                                // call this handler, decide to redial or not
    int *_check_disconn_handler(); // check wheter reached disconn data
};

/*This struct is used to store month data*/
struct _stat_data_month{
    long long _avaliable_data; /*read from UCI file*/
    long long _used_basic_data; /*updated during running*/
    long long _used_total_data;
    long long _adjusted_value;
    int _payday; /*read from UCI file*/
    int status; /*On/Off*/
}

```

```

};

/*This struct is used to store period data*/
struct _stat_data_period
{
    long long _available_data; /*read from UCI file*/
    long long _used_data; /*updated during running*/
    long long _mon_used_data;
    long long _adjusted_value;
    struct tm start_date;
    struct tm end_date;
    int status; //on/off
};

struct _stat_data_idle{
    idle_data_type data_type;
    long long _available_data; //total available data
    long long _used_data;
    long long _adjusted_value;
    struct tm start_hour;
    struct tm end_hour;
    int status; //on/off
};

/*the _traffic_stat supply APIs*/
struct _traffic_stat{
    /*data from power on ( bytes), begin*/
    long long *get_mobile_rx_bytes();
    long long *get_mobile_tx_bytes();
    long long *get_mobile_tx_rx_bytes();
    /*data from power on (bytes), end*/

    /*data used this mon (all), begin*/
    long long *get_month_used_mobile_bytes();
    long long *get_month_left_mobile_bytes();
    /*data used this mon (all), end*/

    /*data used this mon (basic), begin*/
    long long *get_basic_mon_used_mobile_bytes();
    long long *get_basic_mon_left_mobile_bytes();
    /*data used this mon (basic), end*/

    long long *get_period_used_mobile_bytes();
    long long *get_period_left_mobile_bytes();

    long long *get_idle_used_mobile_bytes();

```

```

long long *get_idle_left_mobile_bytes();

long long *get_period_this_mon_used_bytes();
};

struct _stat_monitor{
    struct _stat_data_month data_month;
    struct _stat_data_period data_period;
    struct _stat_data_idle data_idle;
    struct _stat_ctrl_warn stat_warn;
    struct _stat_ctrl_disconn stat_disconn;
    struct _traffic_stat traffic_api;
    stat_handle_state_t handle_status;
    long long last_rd_data;
    long long      unsaved_data;
    int           unsaved_cycle ;
};

```

5. Main Structure

There're two processes:

1. **Save_Monito_Process:** this process is responsible for updating and saving statistics value to UCI file, and deciding whether or not to warn or disconnect the network.

do

{

...

ret = select(maxfdp, &fds, &fds, NULL, &timeout)

...

}while(1);

The timeout is 30 seconds.

This process will run a loop every 30 seconds. So the statistics data value are updated every 30 seconds if no fd is ready.

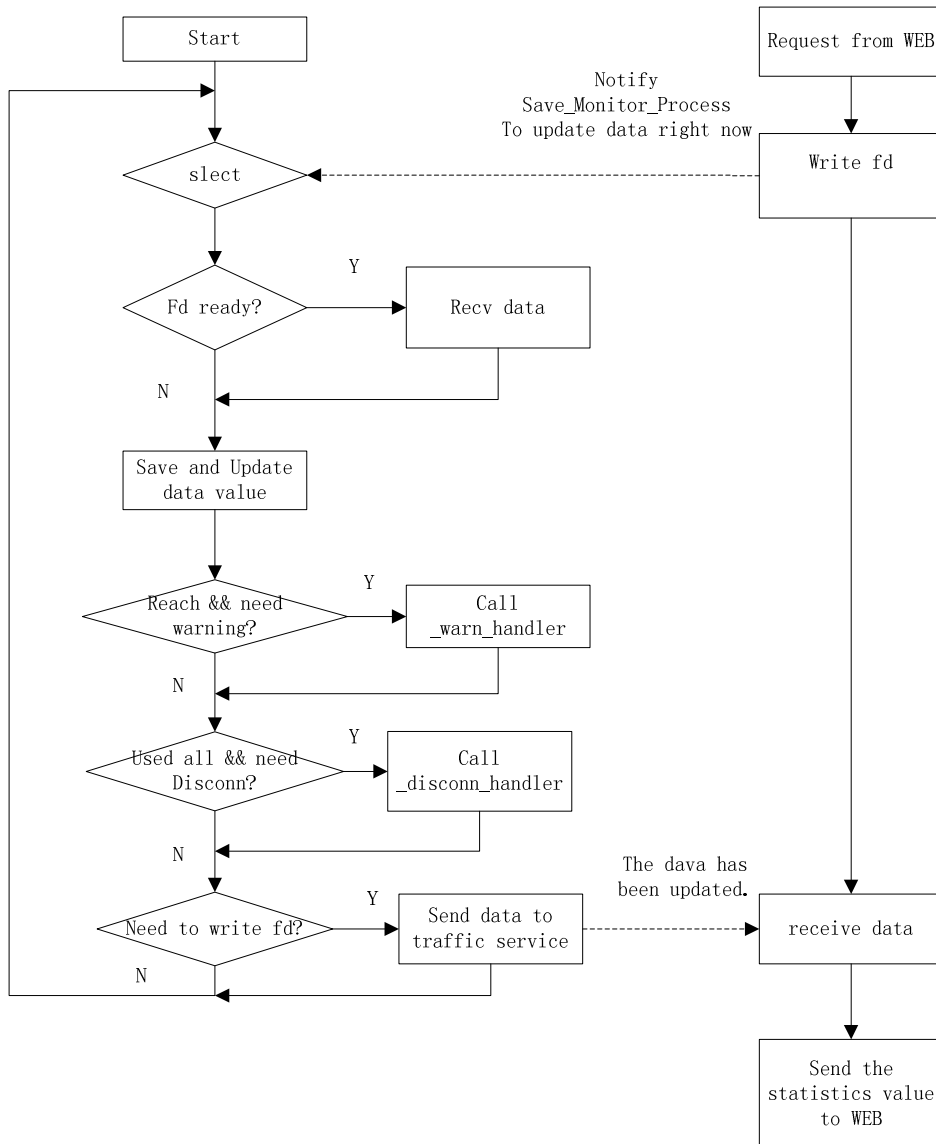
2. **Traffic_Service_Process:** this process receives request from CGI, collects data, transfers the data to blobmsg format, sends the blobmsg to CGI.

3. The two processes communicates by sockets.

When the Traffic_Service_Process receives request from CGI, and wants to get the current statistics value, because the Save_Monitor_Process updates the value every 30 seconds, the Traffic_Service_Process needs to invoke the Save_Monitor_Process by writing command to the socket.

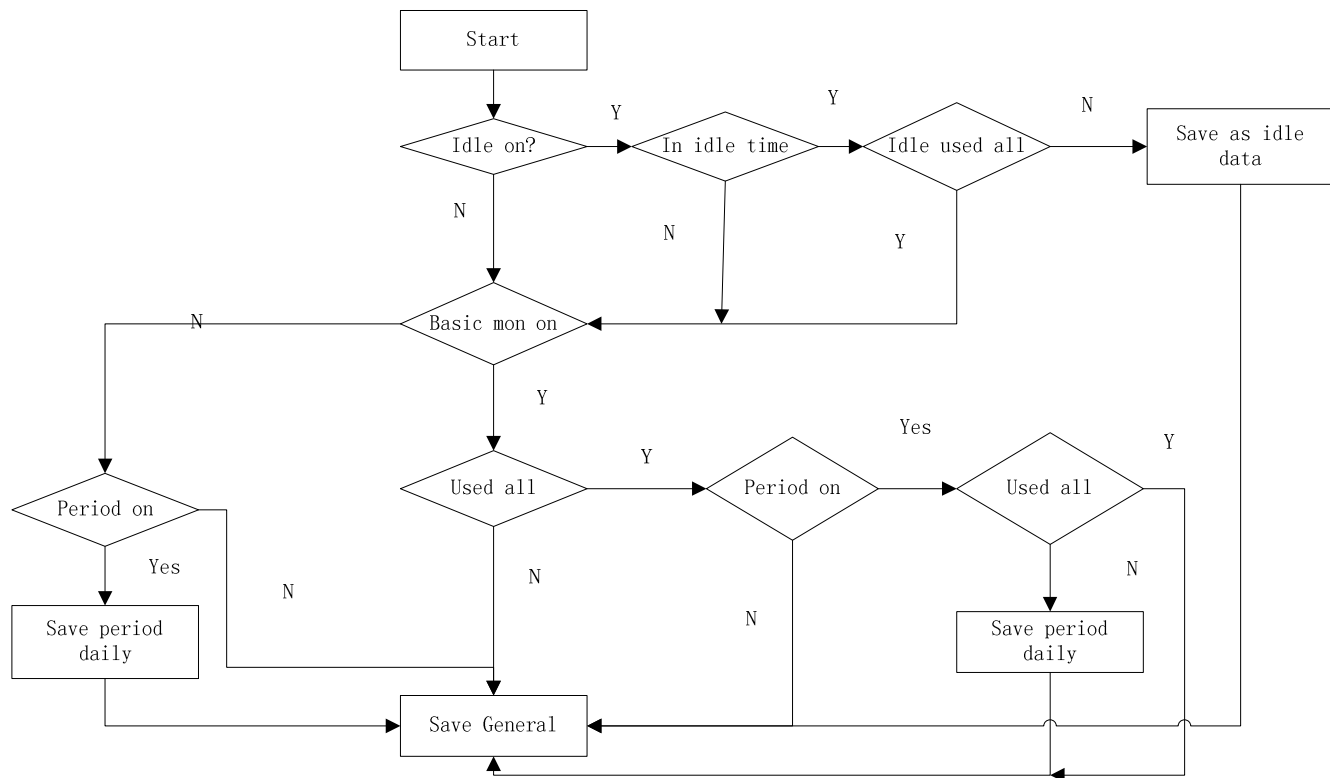
Save_Monitor_Process

Traffic Service



Save and Update flow:

Idle time→basic month→Period



Simple state machine is used to handle this flow:

```

stat_set_state( monitor,STAT_INIT_STATUS);
while (done == 0)
{
    switch (monitor->handle_status)
    {
        case STAT_INIT_STATUS:
        {
            if (monitor->data_idle.status == 1) /*idle is on*/
            {
                stat_set_state(monitor, STAT_IDLE_TIME_ON_STATUS)
            }
            else
            {
                stat_set_state(monitor,STAT_IDLE_TIME_OFF_STATUS);
            }
            break;
        }
        case STAT_IDLE_TIME_ON_STATUS:
        {

```

```

if (get_cur_time_idle_status() == OUT_IDLE)
{
    if (monitor->data_month.status == 1)
    {
        stat_set_state(monitor, STAT_BASIC_MON_ON_STATUS);
    }
    else
    {
        stat_set_state(monitor, STAT_BASIC_MON_OFF_STATUS);
    }
}
else //in idle time
{
    if (monitor->data_idle._used_data < monitor->data_idle._available_data)
    {
        stat_set_state(monitor, STAT_IDLE_TIME_AVL_STATUS);
    }
    else
    {
        stat_set_state(monitor, STAT_IDLE_TIME_FULL_STATUS);
    }
}
break;
}
case STAT_IDLE_TIME_OFF_STATUS:
{
    if (monitor->data_month.status == 1)
    {
        stat_set_state(monitor, STAT_BASIC_MON_ON_STATUS);
    }
    else
    {
        stat_set_state(monitor, STAT_BASIC_MON_OFF_STATUS);
    }
}
break;
}
case STAT_IDLE_TIME_AVL_STATUS:
{
    /*add data used in this cycle to data file*/
    ...
    /*update idle used data*/
    ...
    stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
    break;
}

```



```

}
case STAT_IDLE_TIME_FULL_STATUS:
{
    if (monitor->data_month.status == 1)
    {
        stat_set_state(monitor, STAT_BASIC_MON_ON_STATUS);
    }
    else
    {
        stat_set_state(monitor, STAT_BASIC_MON_OFF_STATUS);
    }
    break;
}
case STAT_BASIC_MON_ON_STATUS:
{
    if(monitor->data_month._used_basic_data < monitor->data_month._available_data)
    {
        stat_set_state(monitor, STAT_BASIC_MON_AVL_STATUS);
    }
    else
    {
        stat_set_state(monitor, STAT_BASIC_MON_FULL_STATUS);
    }
    break;
}
case STAT_BASIC_MON_AVL_STATUS:
{
    stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
    break;
}
case STAT_BASIC_MON_FULL_STATUS:
{
    if(monitor->data_period.status == 1)
    {
        stat_set_state(monitor, STAT_PERIOD_ON_STATUS);
    }
    else
    {
        stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
    }
    break;
}
case STAT_BASIC_MON_OFF_STATUS:
{

```

```

        if(monitor->data_period.status == 1)
        {
            stat_set_state(monitor, STAT_PERIOD_ON_STATUS);
        }
        else
        {
            stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
        }
        break;
    }
    case STAT_PERIOD_ON_STATUS:
    {
        if(get_cur_time_period_status() == OUT_PERIOD)
        {
            stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
        }
        else
        {
            if(monitor->data_period.used_data < monitor->data_period.avaliable_data)
            {
                stat_set_state(monitor, STAT_PERIOD_AVL_STATUS);
            }
            else
            {
                stat_set_state(monitor, STAT_PERIOD_FULL_STATUS);
            }
        }
        break;
    }
    case STAT_PERIOD_OFF_STATUS:
    {
        stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
        break;
    }
    case STAT_PERIOD_AVL_STATUS:
    {
        /*add data used in this cycle to data file*/
        stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
        break;
    }
    case STAT_PERIOD_FULL_STATUS:
    {
        if(monitor->data_month.status == 0)
        {

```

```

        /*add data used in this cycle to data file*/
        ....
    }
    stat_set_state(monitor, STAT_FINAL_RECORD_STATUS);
}
case STAT_FINAL_RECORD_STATUS:
{
    //add the data used in this cycle to data file
    ...
    done = 1;
    break;
}
}
}

```

6. Use Crontab to update and save statistics value at the critical time.

1. At the end of this month
2. At the begin,end of one period
3. At the end of the day
4. At the begin,end of the idle time.

Call ubus method→Traffic_Service_Process→Save_Monitor_Process