Self-hosted home SMS gateway

How to set up Gammu on an old ThinkPad with a 3G modem and listen for MQTT messages to send them as text messages in case your internet is down but you still need to alert.

But... why?

I wanted to be able to alert in problematic situations, such as power loss at home. It has never happened before for longer than minutes, but if it would, it's a nasty surprise: I once managed to accidentally power off my freezer before going away for holiday. I never want to feel that smell ever again. If I don't have power I don't have internet, so I had to come up with something that has backup power and can still alert me.

Europe is small and we have a ridiculous amount of mobile service providers. Before reasonable international roaming prices the only viable option between countries was SMS, so text messages got quite big in Europe.

But there is a deeper reason: text messages are essentially GSM level service status messages and they are small enough to travel through the moment you have any connectivity, including 2G. Rural England has a terrible coverage when it comes to mobile signals and yes, there are places where you're happy to have a plain old GSM connection running. Connecting to web services on GSM is painful, truly, really painful - *an experience everyone should try once, especially if you forgot or never tried what is was like with 56K modems* -, therefore alerting via SMS is a simple, surprisingly reliable way here. It's also cheap: I never had to pay to receive text messages (it's an outrageous idea) and for 5GBP/month I can send unlimited texts within network.

This setup is a little more complicated than it could be. Originally I ran a gammu-smsd and used shell scripts to communicate with it, but that requires an ssh access from remote processes. This way I can use the MQTT as a central hub, allowing other services to send alerts as well. For example if I ever manage to put a Z-Wave sensor network together, I can alert the smoke alarm, the motion sensors, etc as well with the system.

Get the server

First of all we need a hub: in our case, my home server is a Lenovo ThinkPad T400. *In case you need HDMI, get either something newer or a T500, because the T400, the X200 and the X201 doesn't come with digital video port.* If you want something small, quiet, really cheap, and you don't care about the video signal, look for and X200; it's cheaper, than a Raspberry Pi with a battery backup and a GSM shield, and you get a full-blown home server for peanuts.

They usually come with either a 3G modem installed already or a 3G option. Unfortunately not all of them so for the modem or for '3G ready' options, otherwise they won't have the necessary antenna cables. Also check the battery life.

I'm running a Debian Stretch on the machine, so the instructions are according to this. I'm not going to cover how to install linux; there are nice tutorials out there for this.

Optional: Linrunner TLP[^1]

TLP is an advanced power management utility that lets you, for example, undervolt your Core2Duo CPU and limit the battery charge/discharge thresholds. This latter is important, it will help your battery to live significantly longer.

```bash
apt install tlp
```

Part of `/etc/default/tlp` for battery thresholds:

```
START_CHARGE_THRESH_BAT0=60
STOP_CHARGE_THRESH_BAT0=80
```

WARNING: with newer ThinkPads keep the `SATA_LINKPWR_ON_AC` at `maximum_performance` and the `SATA_LINKPWR_ON_BAT` on `medium_power`. If you let them go lower, sometime the SATA devices vanishes, which is really unhealthy. It should look like this:

```
SATA_LINKPWR_ON_AC=maximum_performance
SATA_LINKPWR_ON_BAT=medium_power
```

Set up the SIM card and Gammu

Getting a SIM card

In case you're in the UK, I'd recommend getting a giffgaff pre-paid SIM[^2]. It's cheap and it's working.

Setting up `Gammu`

Gammu is the software that communicates with the modem.

```bash
apt install gammu
```

Create `udev` rules:

`/etc/udev/rules.d/99-ericsson.rules`

```
# Ericsson F3507g
ATTRS{idVendor}=="0bdb", ATTRS{idProduct}=="1900",
ENV{ID_USB_INTERFACE_NUM}=="01", SYMLINK+="f3507g_modem",
GROUP="dialout", MODE="0660"
ATTRS{idVendor}=="0bdb", ATTRS{idProduct}=="1900",
ENV{ID_USB_INTERFACE_NUM}=="03", SYMLINK+="f3507g_data",
GROUP="dialout", MODE="0660"
```

and restart udev:

```bash
sudo udevadm trigger
```

Configure Gammu

`/etc/gammurc`

```bash
[gammu]

port = /dev/f3507g_modem
```

```
model =
connection = at
synchronizetime = yes
logfile = /var/log/ericssonf3507g
logformat = text
use_locking =
gammuloc =
```

Initialise the device

Apparently you need to enable the device and the network before you can use it. **You need gammu 1.38.1 at least for this to work.**

If you don't do this, the modem might refuse to start and start switching between being available on ttyACM0 and ttyACM1.

```bash
gammu -c /etc/gammurc setautonetworklogin
gammu -c /etc/gammurc setpower ON
```

These lines are added to the systemd unit file below, but if you prefer not to use that, you need to take care of this.

Set up Mosquitto MQTT server

MQTT is a lightweight messaging protocol: it has a server (a hub), which collects all the incoming data; publishers pushing the data, and subscribers, reading topics. *I already have this as part of my central bus for sensor data publishing, so extending with alert messages seemed trivial.*

Install Mosquitto

```bash
sudo apt install mosquitto mosquitto-clients
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

Keep in mind that **this will load your MQTT server without authentication and authorization** on port `1883`, so don't ever do this on an internet facing device. For that protect it with password and maybe with TLS encryption as well.

To test it:

```bash
mosquitto_sub -h 127.0.0.1 -p 1883 -u 'your-mqtt-user-if-any' -P 'your-mqtt-password-if-any' -t '#' -v
```

The `#` is to subscribe to everything.

Add an ini for the MQTT clients

This is not part of Mosquitto, but we're going to use it for our publishing and subscribing Python and Bash scripts.

`/etc/mqtt.ini`

```ini
[mqtt]
host = 127.0.01
port = 1883
user = your-mqtt-user-if-any
password = your-mqtt-password-if-any
```

Glue it together with Python

Required packages

```bash
apt install python3 python3-setuptools python3-pip
python3-wheel python3-gammu python3-dev
pip3 install paho-mqtt
```

Service script

/usr/local/bin/mqtt2sms.py

```python
#!/usr/bin/env python3

import paho.mqtt.client as mqtt
import json
import configparser
import gammu
import os
import time
import logging
import sys

class SMSGateway(object):
    def __init__(self):
        self.sm = gammu.StateMachine()
        self.sm.ReadConfig(Filename='/etc/gammurc')
        self.sm.Init()

    def send(self, text, number):
        message = {
            'Text': '%s' % text,
            'SMSC': {'Location': 1},
            'Number': '%s' % number,
        }

        try:
```

```
            self.sm.SendSMS(message)
            print("sending SMS to %s with text %s" % (number,
text), file=sys.stdout)
            return True
        except Exception as e:
            print("SMS sending failed: %s" % e,
file=sys.stderr)
            return False

class MQTTSMSListener(mqtt.Client):
    def on_message(self, mqttc, obj, msg):
        try:
            data = json.loads(msg.payload.decode("utf-8"))
            message = data.get('message', None)
            if not message:
                print('no message body to send',
file=sys.stderr)
                return False

            number = data.get('number', None)
            if not number:
                print('no number to send to', file=sys.stderr)
                return False
            self.sms.send(message, number)
        except Exception as e:
            print('failed to decode JSON, reason: %s, string:
%s' % (e, msg.payload), file=sys.stderr)

    def run(self):
        self.sms = SMSGateway()
        mqttconf = configparser.ConfigParser()
        mqttconf.read('/etc/mqtt.ini')
        self.username_pw_set(
            mqttconf.get('mqtt', 'user'),
            mqttconf.get('mqtt', 'password')
        )

        self.connect(
            mqttconf.get('mqtt', 'host'),
            mqttconf.getint('mqtt', 'port'),
            60
```

```
        )
        self.subscribe("sms")

        rc = 0
        while rc == 0:
            rc = self.loop()
        return rc


mqttc = MQTTSMSListener(clean_session=True)
rc = mqttc.run()
```

systemd unit file

If you have systemd, you can save this as a simple unit file and run the service with it:

`/lib/systemd/system/mqtt2sms.service`

```
[Unit]
Description=start Python MQTT 2 SMS gateway
After=network.target

[Service]
Type=simple
ExecStartPre=/usr/bin/gammu -c /etc/gammurc setautonetworklogin
ExecStartPre=/usr/bin/gammu -c /etc/gammurc setpower ON
ExecStart=/usr/local/bin/mqtt2sms.py

[Install]
WantedBy=multi-user.target
```

Once done, do:

```bash
systemctl daemon-reload
systemctl enable mqtt2sms
systemctl start mqtt2sms
```

And you're good to go. Anything that goes into the topic `sms`, JSON encoded and has a `message` and a `number` field will be forwarded as SMS.

Alerting for the local server conditions

These are a few checks that are running on my local home server to poke me if something goes wrong.

Required packages

ThinkPads have a few special packages: these can handle all the extra hardware ThinkPad have.

```bash
apt install acpi-call-dkms tp-smapi-dkms

for mod in "tp_smapi" "thinkpad_ec" "thinkpad_acpi"; do
    modprobe "$mod"
    echo "$mod" >> /etc/modules
done
```

A simple Bash wrapper for sending messages to MQTT

`/usr/local/bin/alert`

```bash
#!/usr/bin/env bash

message="$1"
number="${2-[your default phone number comes here]}"

function mqttconf {
    grep -i "$1" /etc/mqtt.ini | awk '{print $3}'
}

mosquitto_pub -h "$(mqttconf host)" -p "$(mqttconf port)" -u "$
(mqttconf user)" -P "$(mqttconf password)" -t "sms" -m
"{\"message\": \"$message\", \"number\": \"$number\"}"
```

ThinkPad specific checks to send alerts

`/usr/local/bin/alerts`

bash

```
#!/usr/bin/env bash

function original_to_previous {
    original_path="$1"
    echo "/tmp/__$(basename "$original_path")"
}

function get_previous_status {
    path="$1"
    previous_path="$(original_to_previous $path)"

    if [ ! -f "$previous_path" ]; then
        touch "$previous_path"
    fi

    cat "$previous_path"
}

function set_previous_status {
    path="$1"
    previous_path="$(original_to_previous $path)"
    status="$2"

    echo "$status" > "$previous_path"
}

# ac status
path="/sys/devices/platform/smapi/ac_connected"
curr="$(cat $path)"
prev="$(get_previous_status $path)"
#echo "AC: $curr, previous: $prev"

if [ "$curr" != "$prev" ]; then
    if [ "$curr" != "1" ]; then
        /usr/local/bin/alert "WARNING: $(hostname -f) running
on battery"
    else
        /usr/local/bin/alert "OK: $(hostname -f) running on AC"
    fi
    set_previous_status "$path" "$curr"
fi
```

```
# battery level
path="/sys/devices/platform/smapi/BAT0/remaining_percent"
curr="$(cat $path)"
prev="$(get_previous_status $path)"
#echo "Battery level: $curr, previous: $prev"

if [ $curr -lt 30 ]; then
    if [ "$curr" -lt "$prev" ]; then
        /usr/local/bin/alert "WARNING: $(hostname -f) low
battery at $curr %"
    fi
fi
set_previous_status "$path" "$curr"

# internet connectivity
path="internet_connectivity"
curr="$(nc -z google.com 443 && echo "ok" || echo "failed")"
prev="$(get_previous_status $path)"
#echo "Internet connectivity: $curr, previous: $prev"

if [ "$curr" != "$prev" ]; then
    if [ "$curr" == "failed" ]; then
        /usr/local/bin/alert "WARNING: $(hostname -f) can't
connect to google.com"
    else
        /usr/local/bin/alert "OK: $(hostname -f) can connect
to google.com"
    fi
    set_previous_status "$path" "$curr"
fi
```

Run it as a cron job

`/etc/cron.d/alerts`

```
* * * * * root /usr/local/bin/alerts
```

Enjoy your text messages!

Links

1. http://linrunner.de/en/tlp/docs/tlp-linux-advanced-power-management.html
2. https://www.giffgaff.com/orders/affiliate/petermolnar2

Created by Peter Molnar <mail@petermolnar.net>, published at 2017-08-24 19:00 UTC, last modified at 2021-05-11 11:49 UTC , to canonical URL https://petermolnar.net/article/home-sms-gateway-with-mqtt-gammu-thinkpad/ , licensed under CC-BY-4.0 .