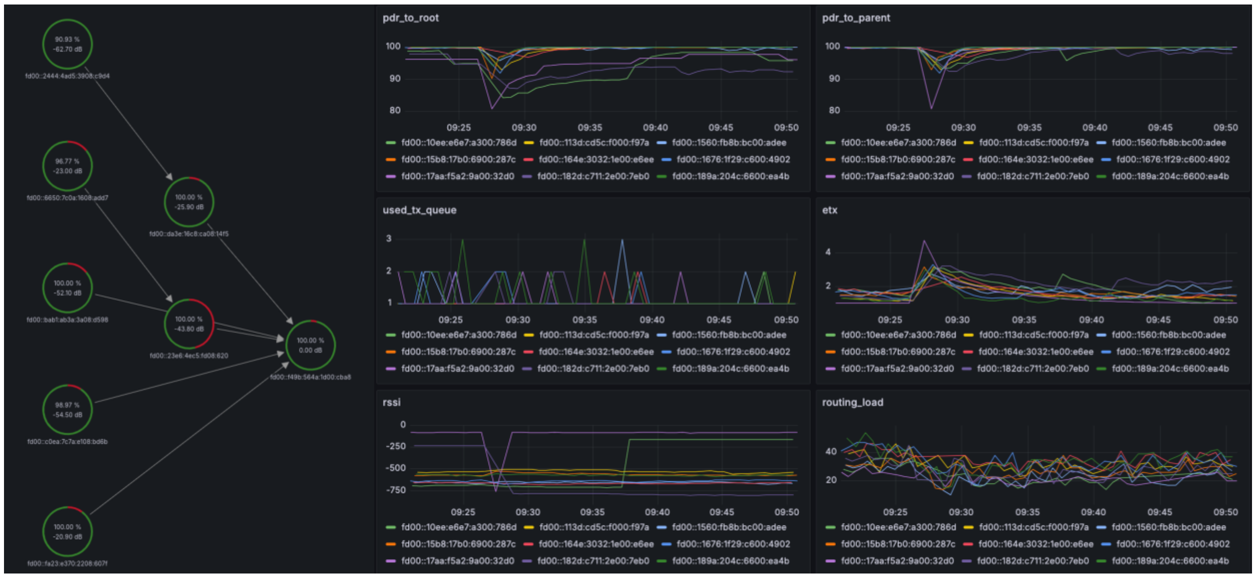
Mira version 2.10.0 Release Notes

Table of Contents

Features

Gateway Monitoring API and Dashboard

The Mira Border Gateway now comes with a built-in monitoring dashboard and API for easy visualization of key network performance metrics. It can be used in development to better understand the behavior of MiraMesh networks, as well as in the field to help installers validate a good installation.


pdr_to_root
pdr_to_parent
used_tx_queue
etx
rssi
routing_load
fd00::10ee:e6e7:a300:786d
fd00::113d:cd5c:f000:f97a
fd00::1560:fb8b:bc00:adee
fd00::15b8:17b0:6900:287c
fd00::164e:3032:1e00:e6ee
fd00::1676:1f29:c600:4902
fd00::17aa:f5a2:9a00:32d0
fd00::182d:c711:2e00:7eb0
fd00::189a:204c:6600:ea4b

The dashboard itself is implemented in [Grafana]() and contains a visual tree structure topology graph and time series graphs.

All data presented by Grafana is fetched through a REST API hosted by the Border Gateway application. This API is fully documented and stable, making the Grafana visualization optional. You can build your own visualization or just pass the data along to some other service, if desired. The API can supply the following monitoring metrics:

- Network topology
- Current network time
- Per node:
  - IPv6 address, net rate and role
  - PDR, ETX and RSSI to parent
  - Best 5 neighbors and their rating, PDR, ETX, RSSI and net rate
  - Routing load (explained below)
  - Used TX queue slots
  - MAC statistics

We regard the Grafana visualization as an example on how to use the REST API. While you can use it out of the box, it comes with a number of limitations that may make it unsuitable for your use case. The most apparent being the number of nodes in the node graph tree. Grafana handles 10-30 nodes ok, but the visualization and positioning of nodes becomes pretty messy with large amounts of nodes. The Grafana instance may also use a lot of RAM. Luckily, it's very easy to run the Grafana instance on a more capable computer and poll the Border Gateway API over a local network. The API is hosted over HTTP and not HTTPS.

New network metric: routing load

The **mira_diag** API has been extended with a new metric called "routing load". This is a metric of how many slots out of the available RX slots are consumed for a mesh node, expressed in percent. This takes both incoming and outgoing unicast traffic into account. However, it does not consider link-local multicast slots, since joining a multicast group may activate new slots.

Routing load is a good metric to gauge how much headroom a mesh or root node has for routing other nodes traffic. It can be used to identify network overload before it happens, as well as finding bottlenecks in heavily loaded nodes. Keep in mind that MiraMesh does not use a TDMA schema. Instead, sending is pseudo-random, making a value of 100% basically unattainable. In our experience, an average value of 50% or above is the practical limit at which heavy collisions start to occur.

Network time from Border Gateway

Getting the network time from the Border Gateway is now possible through the monitoring API. There are functions to get the current network time in ticks, and functions to translate between network time and UTC time. The UTC time is based on the current time of the host operating system.

Warning: Keep in mind that gateway restarts may change the current network time **drastically**, since the network time is randomized on startup of a gateway when it's not re-joining an existing network. This can cause problems when scheduling things to happen in the future. Try to keep the translation between network time and UTC as fresh as possible. If you're scheduling something very critical, it might be worth keeping track of gateway restarts.

Signals – a new way to wake processes

Signals is a mix between events and polls, used to wake up one or more sleeping processes. Processes can register that they should be woken by one or more signals and other processes can wake up all processes waiting on a signal. The processes are then woken with a poll-event and all the signals they were waiting for are forgotten.

This was introduced because it was easy to write code like this by mistake:
PROCESS_WAIT_UNTIL(mira_net_get_state() == MIRA_NET_STATE_JOINED);

PROCESS_WAIT_UNTIL would put the process to sleep, but since nothing was explicitly waking up the process, it could take a long time before the expression was evaluated again. Now **mira_net_get_state()** has been changed to register a signal and when the state is changed all processes waiting for that signal will be polled.

Some functions saved the current process to know which process had called them. This was later used to wake said process when the state had changed. **mira_config_write()** was such a function. This usually worked fine, but when **mira_config_is_working()** was being waited on before **mira_config_write()** was called, the process would not be woken until some broadcast event woke it.

These functions have been changed to now use signals to wake up processes: **mira_net_get_state(), mira_config_is_working(), mira_fota_is_working()** and **mira_flash_is_working().**

Read more about signals in the documentation:
https://docs.lumenrad.io/miraos/latest/api/miraos/kernel/events.html#signals

Network state change events

Previously the network state could only be checked from a simple get function. This means that monitoring the net state requires a process continuously checking this function on an interval. This adds latency to react to changes, current consumption of waking up often or both. In 2.10 it is possible to register a callback, through `mira_net_register_net_state_cb()`, to be called on network state changes, enabling the application to be event driven instead.

Option to override set frontend configuration

With the introduction of the **mira_net_set_active_frontend_cfg()** it is now possible to reconfigure the frontend configuration from the application. Note, the API must only be called before **mira_net_init().**

Radio validation tool

Mira now comes with a firmware designed for validating a radio design, the "radio validation tool". It contains functions to continuously transmit, transmit duty-cycled and frequency hopping. It can control output power and potential radio frontend settings in detail and set up a connection between two devices for range measurements. It's primarily designed to make regulatory certification measurements easier but can be used in the lab for performance characterization as well.

The firmware is controlled with a serial interface over either UART or USB. All commands are described when writing the help command "help". You'll find the radio validation tool in the libmira archive under the folder "firmwares".

Updates and improvements

CRC functions exposed

MiraMesh uses CRC32 for validation check of firmware transfers. Sometimes it's useful to re-use or replicate this CRC calculation in the application. To make sure that implementations of the CRC32 algorithm are identical in such cases, MiraMesh's CRC32 functions are now exposed in an API called **mira_crc.**

Build system now supports pyOCD

The build system of MiraOS and MiraMesh required either nrfjprog or jlink software suite to flash devices. Now the open source openOCD-based python tool – pyOCD (v.0.36 or higher) – can be used instead. This introduces support for more types of programmers and debuggers, most notably the very inexpensive raspberry pi debug probe.

Watchdog for Border Gateway

We have seen rare instances of the Border Gateway application hanging indefinitely. We have not been able to reproduce the issue. As a 'catch-all' resolution to this we have introduced a watchdog that terminates the application if it hangs. It is recommended to run the Border Gateway application in a systemd service configured for auto-restart.

Border Gateway built for arm64 linux

Previously, the pre-compiled Border Gateway applications were limited to x86 64bit Linux and arm32 (armhf) Linux. Now a build for arm64 Linux is added in the archive.

"Factory config" replaces license area and tooling

The license flash area (also referred to as to certificate) has been used for more things than just storing the Mira license, such as frontend configuration. This has been confusing. The area and associated tools are now renamed to "factory config", as the area should store things flashed in a production (factory) once and never modified again. Aliases to the old naming exists so code should not have to be modified, although using the new naming is encouraged.

The mira_license.py still exists, but is limited to license related operations only.

Configurable UART pins via factory config

The factory config area has been extended to contain configuration for UART pins. These can be read in application code and used when running the UART init function.

Pre-compiled firmwares use this feature to make them more hardware agnostic:

- Border Gateway Radio on a Stick (RoaS) firmware
- Radio validation tool

This removes the requirement for the hardware to implement the exact UART pins that we chose in the firmware. If the factory config does not contain information about UART pins, the previous values are used as default.

Breaking change: mira_fota_set_driver() takes struct as argument

In order to be able to extend the API without doing breaking changes, ironically, a braking change had to be made. `mira_fota_set_driver()` now takes a struct as an argument, making it possible to add fields in the struct without breaking application code.

Bugfixes

Workaround for nrf anomaly 102 on 832-rev1

nRF anomaly 102 workaround implemented. This workaround reduces receiver sensitivity by 3 dB, but only on 832-rev1 hardware.

Setting config areas to zero does not make MiraOS crash

Using linker scripts with the config areas set to have length 0 cause MiraOS to crash on boot. In 2.10.0 it is allowed to set them to have 0 length and the device will boot as normal, but the `mira_config` API will return errors when called.

Possible to have custom module config without frontend config

When registering a custom module config with `mira_sys_module_config_register()` the frontend config field had to have valid values, even if a frontend module wasn't used. Setting it to NULL would make the stack read uninitialized memory and interpret that as a frontend config, causing unidentified behavior. Now it's possible to set the frontend field to NULL, which will make the stack ignore the field and operate like no frontend is present.

Link-local multicast now works on Border Gateway

The Border Gateway did not relay messages sent to link-local addresses between the host and radio, even if the addresses were bound to in the configuration. Now the link-local messages get forwarded properly. An example, "multicast", has been added where the Border Gateway is used to listen to link-local multicasts sent by mesh nodes.

Known issues

Allocating too many events lacks error handling

`process_alloc_event()` returns a `process_event_t` (one unsigned byte) of increasing values, starting from `PROCESS_EVENT_MAX` (0x8a). Events above `PROCESS_EVENT_MAX` are user space, while events below are internal to the stack. The potential problem arises when more than 177 (0xFF - 0x8A) events are allocated. This does not produce an error. Instead, the value will wrap around and return an event already used for internal purposes. This causes undefined behavior.

Allocating this many events is not a valid use case, and if it were to happen most likely the result of a bug. However, to write safer code it might be worth to check the returned event to limit the damage of such a bug.

RSSI sometimes reports invalid value

Both RSSI to parent and RSSI to neighbor can sometimes have the value **INT16MIN** (-32768). This value gets returned when a real value is missing, for example when the network is started but no parent or neighbor is present yet. However, this value can sometimes get returned during normal operation. This is an erroneous value and should be ignored.

USB radio stick bootloader doesn't enter DFU mode after update

When starting the gateway with an older radio stick using USB it may need to update the USB DFU bootloader. The update will finish, but it will not enter the DFU mode. It needs to be in DFU mode to be able to also load the gateway radio firmware, so this hangs the gateway host program. A workaround is to manually set the USB radio stick into DFU mode by running the script reset_to_dfu.py, located in the tools folder.

Using gcc option -B results in infinite loop

When compiling an application with libmira.a using gcc option -B, compilation gets stuck in an infinite loop, compiling the application over and over again.

Higher current consumption when running with 2.6.2 root

During compatibility testing it was observed that 2.10.0 nodes draw slightly more current when having a 2.6.2 root node. In the test case an increase of 1-2 uA was observed for devices normally drawing around 18 uA. It is therefore recommended to run the same Mira version on all devices in the network except for in an upgrade scenario.

mira_flash_write() hangs if writing to page 0

Instead of throwing an error, the API hangs indefinitely if one tries to write to the invalid block 0.

Lower output power than desired on MWA N3

Setting the output power to a value between 21 and 52 cBm on the MWA_N3 will cause the output power to be set to −147 cBm. Use either values below 21 or above 52.

2.8.X based nodes calculate PDR incorrectly to nodes with older versions

A 2.8.x based node will calculate estimated PDR incorrectly on links to nodes running 2.7.x and below. This will cause the `mira_diag_get_estimated_pdr()` to be incorrect and affect routing, which can cause network instability and degrade performance. It is therefore recommended to run the same Mira version on all devices in the network except for in an upgrade scenario.

This bug was resolved in 2.9.0, which makes 2.7.2 and below and 2.9.0 and above ok to mix.