Gluon Documentation

Release 2014.3

Project Gluon

August 05, 2014

Contents

Gluon is a modular framework for creating OpenWrt-based firmwares for wireless mesh nodes. Several Freifunk communities in Germany use Gluon as the foundation of their Freifunk firmwares.

CHAPTER 1

User Documentation

1.1 Getting Started

To build Gluon, after checking out the repository change to the source root directory to perform the following commands:

```
git clone git://github.com/freifunk-gluon/site-ffhl.git site # Get the Freifunk Lübeck site repositor
make update                                                  # Get other repositories used by Gluon
make                                                         # Build Gluon
```

When calling make, the OpenWRT build environment is prepared/updated. To rebuild the images only, just use:

```
make images
```

The built images can be found in the directory *images*. Of these, the factory images are to be used when flashing from the original firmware a device came with, and sysupgrade is to upgrade from other versions of Gluon or any other OpenWRT-based system.

For the build reserve 6GB of disk space. The build requires packages for *subversion*, ncurses headers (*libncurses-dev*) and zlib headers (*libz-dev*).

There are two levels of *make clean*:

```
make clean
```

will ensure all packages are rebuilt; this is what you normally want to do after an update.

```
make dirclean
```

will clean the entire tree, so the toolchain will be rebuilt as well, which is not necessary in most cases, and will take a while.

So all in all, to update and rebuild a Gluon build tree, the following commands should be used:

```
git pull
(cd site && git pull)
make update
make clean
make
```

1.2 Site

The `site` consists of the files `site.conf` and `site.mk`. In the first community based values are defined, which both are processed during the build process and runtime. The last is directly included in the make process of Gluon.

1.2.1 Configuration

The `site.conf` is a lua dictionary with the following defined keys.

hostname_prefix A string which shall prefix the default hostname of a device.

site_name The name of your community.

site_code The code of your community. It is good practice to use the TLD of your community here.

prefix4 The IPv4 Subnet of your community mesh network in CIDR notation, e.g.

```
prefix4 = '10.111.111.0/18'
```

prefix6 The IPv6 subnet of your community mesh network, e.g.

```
prefix6 = 'fdca::ffee:babe:1::/64'
```

timezone The timezone of your community live in, e.g.

```
-- Europe/Berlin
timezone = 'CET-1CEST,M3.5.0,M10.5.0/3'
```

ntp_server List of NTP servers available in your community or used by your community, e.g.:

```
ntp_servers = {'1.ntp.services.ffeh','2.tnp.services.ffeh'}
```

opkg_repo [optional] Overwrite the default `opkg` repository server, e.g.:

```
opkg_repo = 'http://opkg.services.ffeh/attitude_adjustment/12.09/%S/packages'
```

The *%S* is a variable, which is replaced with the platform of an device during the build process.

regdom The wireless regulatory domain responsible for your area, e.g.:

```
regdom = 'DE'
```

wifi24 WLAN Configuration of your community in the 2.4Ghz radio. Consisting of `ssid` of your client network, the `channel` your community is using, `htmode`, the adhoc ssid `mesh_ssid` used between devices, the adhoc bssid `mesh_bssid` and the adhoc multicast rate `mesh_mcast_rate`. Combined in an dictionary, e.g.:

```
wifi24 = {
  ssid = 'http://kiel.freifunk.net/',
  channel = 11,
  htmode = 'HT40-',
  mesh_ssid = 'ff:ff:ff:ee:ba:be',
  mesh_bssid = 'ff:ff:ff:ee:ba:be',
  mesh_mcast_rate = 12000,
},
```

wifi5 Same as *wifi24* but for the 5Ghz radio.

next_node [package] Configuration of the local node feature of Gluon

```
next_node = {
  ip4 = '10.23.42.1',
  ip6 = 'fdca:ffee:babe:1::1',
  mac = 'ca:ff:ee:ba:be'
}
```

fastd_mesh_vpn Remote server setup for vpn.

```
fastd_mesh_vpn = {
  methods = {'salsa2012+gmac'},
  mtu = 1426,
  backbone = {
    limit = 2,
    peers = {
      ffki_rz = {
        key = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',
        remotes = {'ipv4 "vpn1.entenhausen.freifunk.net" port 10000'},
      },
    }
  }
}
```

mesh_on_wan [optional] Enables the mesh on the WAN port (`true` or `false`).

autoupdater [package] Configuration for the autoupdater feature of Gluon.

```
autoupdater = {
  enabled = 1,
  branch = 'experimental',
  branches = {
    stable = {
      name = 'stable',
      mirrors = {
        'http://{fdca:ffee:babe:1::fec1}/firmware/stable/sysupgrade/',
        'http://{fdca:ffee:babe:1::fec2}/firmware/stable/sysupgrade/',
      },
      probability = 0.08,
      good_signatures = 2,
      pubkeys = {
        'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX', -- someguy
        'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX', -- someother
      }
    }
  }
}
```

simple_tc [package] Uplink traffic control

```
simple_tc = {
  mesh_vpn = {
    ifname = 'mesh-vpn',
    enabled = false,
    limit_egress = 200,
    limit_ingress = 3000,
  },
},
```

config_mode [package] Configuration Mode text blocks

legacy [package] Configuration for the legacy upgrade path. This is only required in communities upgrading from

Lübeck's LFF-0.3.x.

```
legacy = {
        version_files = {'/etc/.freifunk_version_keep', '/etc/.eff_version_keep'},
        old_files = {'/etc/config/config_mode', '/etc/config/ffeh', '/etc/config/freifunk'},
        config_mode_configs = {'config_mode', 'ffeh', 'freifunk'},
        fastd_configs = {'ffeh_mesh_vpn', 'mesh_vpn'},
        mesh_ifname = 'freifunk',
        tc_configs = {'ffki', 'freifunk'},
        wifi_names = {'wifi_freifunk', 'wifi_freifunk5', 'wifi_mesh', 'wifi_mesh5'},
}
```

1.2.2 Packages

The `site.mk` is a Makefile which should define constants involved in the build process of Gluon.

GLUON_SITE_PACKAGES Defines a list of packages which should installed additional to the `gluon_core` package.

GLUON_RELEASE The current release version Gluon should use.

GLUON_PRIORITY The default priority for the generated manifests (see the autoupdater documentation for more information).

1.2.3 Examples

An example configuration is maintained at https://github.com/freifunk-gluon/site-example.

1.3 Builds

For the build reserve 6GB of disk space. The building requires packages for `subversion`, ncurses headers (`libncurses-dev`) and zlib headers (`libz-dev`).

1.3.1 Building Gluon

To build Gluon, after checking out the repository change to the source root directory to perform the following commands:

```
git clone git://github.com/freifunk-gluon/site-ffhl.git site # Get the Freifunk Lübeck site repositor
make update                                                  # Get other repositories used by Gluon
make                                                         # Build Gluon
```

When calling `make`, the OpenWRT build environment is prepared and updated. To rebuild the images only, just use:

```
make images
```

The built images can be found in the directory `images`.

1.3.2 Cleaning up

There are three levels of `make clean`:

`make clean`

will only clean the Gluon-specific files;

`make cleanall`

will also call `make clean` on the OpenWRT tree, and

`make dirclean`

will do all this, and call `make dirclean` on the OpenWRT tree.

1.3.3 Environment variables

Gluon's build process can be controlled by various environment variables.

GLUON_SITEDIR Path to the site configuration. Defaults to `site/`.

GLUON_IMAGEDIR Path where images will be stored. Defaults to `images/`.

GLUON_BUILDDIR Working directory during build. Defaults to `build/`.

1.4 Frequently Asked Questions

CHAPTER 2

Features

2.1 Config Mode

When in Config Mode a node will neither participate in the mesh nor connect to the VPN using the WAN port. Instead, it'll offer a web interface on the LAN port to aid configuration of the node.

Whether a node is in Config Mode can be determined by a characteristic blinking sequence of the SYS LED:

2.1.1 Activating Config Mode

Config Mode is automatically entered at the first boot. You can re-enter Config Mode by pressing and holding the RESET/WPS button for about three seconds. The device should reboot (all LEDs will turn of briefly) and Config Mode will be available.

2.1.2 Port Configuration

In general, Config Mode will be offered on the LAN ports. However, there are two practical exceptions:

- Devices with just one network port will run Config Mode on that port.
- Devices with PoE on the WAN port will run Config Mode on the WAN port instead.

2.1.3 Accessing Config Mode

Config Mode can be accessed at http://192.168.1.1. The node will offer DHCP to clients. Should this fail, you may assign an IP from 192.168.1.0/24 to your computer manually.

2.2 Autoupdater

Gluon contains an automatic update system which can be configured in the site configuration.

2.2.1 Building Images

By default, the autoupdater is disabled (as it is usually not helpful to have unexpected updates during development), but it can be enabled by setting the variable GLUON_BRANCH when building to override the default branch set in the set in the site configuration.

A manifest file for the updater can be generated with *make manifest*. A signing script (using ecdsautils) can by found in the *contrib* directory. When creating the manifest, `GLUON_PRIORITY` can be set on the command line, or it can be taken from the `site.mk`.

The priority defines the maximum number of days that may pass between releasing an update and installation of the images. The update probability with start at 0 after the release time mentioned in the manifest and then slowly rise to 1 after the number of days given by the priority has passed.

The priority may be an integer or a decimal fraction.

A fully automated nightly build could use the following commands:

```
git pull
(cd site && git pull)
make update
make clean
make -j5 GLUON_BRANCH=experimental
make manifest GLUON_BRANCH=experimental
contrib/sign.sh $SECRETKEY images/sysupgrade/experimental.manifest

rm -rf /where/to/put/this/experimental
cp -r images /where/to/put/this/experimental
ln -s experimental.manifest /where/to/put/this/experimental/sysupgrade/manifest
```

2.2.2 Infrastructure

We suggest to have following directory tree accessible via http:

```
firmware/
        stable/
                sysupgrade/
                factory/
        snapshot/
                sysupgrade/
                factory/
        experimental/
                sysupgrade/
                factory/
```

The last level is generated by the Gluon build process. Do not forget to create symlinks from `manifest` to `<branch>.manifest` in the sysupgrade directories to allow upgrades from Gluon versions before 2014.3.

The server should be available via IPv6.

2.2.3 Command Line

These commands can be used on a node.

```
# Update with some probability
autoupdate

# Force update check, even when the updater is disabled
autoupdater -f
```

2.3 Mesh on WAN

It's possible to enable the mesh on the WAN port like this:

:: uci set network.mesh_wan.auto=1 uci commit

It may also be disabled again by running:

:: uci set network.mesh_wan.auto=0 uci commit

2.3.1 site.conf

The optional option `mesh_on_wan` may be set to `true` (`false` is the default) to enable meshing on the WAN port without further configuration.

2.4 Announcing Node Information

Gluon is capable of announcing information about each node to the mesh and to neighbouring nodes. This allows nodes to learn each others hostname, IP addresses, location, software versions and various other information.

2.4.1 Format of collected data

Information to be announced is currently split into two categories:

nodeinfo In this category (mostly) static information is collected. If something is unlikely to change without human intervention it should be put here.

statistics This category holds fast changing data, like traffic counters, uptime, system load or the selected gateway.

Both categories will have a `node_id` key be default. It should be used to match data from *statistics* to *nodeinfo*.

2.4.2 Accessing Node Information

There are two packages responsible for distribution of the information. For one, information is distributed across the mesh using alfred. Information between neighbouring nodes is exchanged using *gluon-announced*.

alfred (mesh bound)

The package `gluon-alfred` is required for this to work.

Using alfred both categories are distributed within the mesh. In order to retrieve the data you'll need both a local alfred daemon and alfred-json installed. Please note that at least one alfred daemon is required to run as *master*.

nodeinfo is distributed as alfred datatype *158*, while *statistics* uses *159*. Both are compressed using GZip (alfred-json can handle the decompression).

In order to retrieve statistics data you could run:

```
# alfred-json -z -r 159
{
  "f8:d1:11:7e:97:dc": {
    "processes": {
```

```
      "total": 55,
      "running": 2
    },
    "idletime": 30632.290000000001,
    "uptime": 33200.07,
    "memory": {
      "free": 1660,
      "cached": 8268,
      "total": 29212,
      "buffers": 2236
    },
    "node_id": "f8d1117e97dc",
    "loadavg": 0.01
  },
  "90:f6:52:3e:b9:50": {
    "processes": {
      "total": 58,
      "running": 2
    },
    "idletime": 28047.470000000001,
    "uptime": 33307.849999999999,
    "memory": {
      "free": 2364,
      "cached": 7168,
      "total": 29212,
      "buffers": 1952
    },
    "node_id": "90f6523eb950",
    "loadavg": 0.34000000000000002
  }
}
```


You can find more information about alfred in its README.

gluon-announced

gluon-announced allows querying neighbouring nodes for their *nodeinfo*. It is a daemon listening on the multicast address `ff02::2:1001` on UDP port 1001 on the bare mesh interfaces. There is no client yet (but it's being developed), but you can query the information using tools like `socat`:

```
# socat - UDP6-DATAGRAM:[ff02::2:1001%wlan0-1]:1001
nodeinfo
```

This output is not compressed, but that is likely to change in the future. If you intent to use `gluon-announced`, please contact *tcatm* in Gluon's IRC channel.

2.4.3 Adding a fact

To add a fact just add a file to either `/lib/gluon/announce/nodeinfo.d/` or `/lib/gluon/announce/statistics.d/`.

The file must contain a lua script and its name will become the key for the resulting JSON object. A simple script adding a `hostname` field might look like this:

```
return uci:get_first('system', 'system', 'hostname')
```

The directory structure will be converted to a JSON object, i.e. you may create subdirectories. So, if the directories look like this

```
.
-- hardware
|   -- model
-- hostname
-- network
|   -- mac
-- node_id
-- software
    -- firmware
```

the resulting JSON would become:

```
# /lib/gluon/announce/announce.lua nodeinfo
{
   "hardware" : {
      "model" : "TP-Link TL-MR3420 v1"
   },
   "hostname" : "mr3420-test",
   "network" : {
      "mac" : "90:f6:52:82:06:02"
   },
   "node_id" : "90f652820602",
   "software" : {
      "firmware" : {
         "base" : "gluon-v2014.2-32-ge831099",
         "release" : "0.4.1+0-exp20140720"
      }
   }
}
```

CHAPTER 3

Developer Documentation

3.1 Development Basics

Gluon's source is kept in git repositories at GitHub.

3.1.1 Bug Tracker

The main repo does have issues enabled.

3.1.2 IRC

Gluon's developers frequent #gluon on hackint. You're welcome to join us!

3.1.3 Working with repositories

To update the repositories used by Gluon, just adjust the commit IDs in *modules* and rerun

```
make update
```

make update also applies the patches that can be found in the directories found in *patches*; the resulting branch will be called *patched*, while the commit specified in *modules* can be refered to by the branch *base*.

```
make unpatch
```

sets the repositories to the *base* branch,

```
make patch
```

re-applies the patches by resetting the *patched* branch to *base* and calling *git am* for the patch files. Calling *make* or a similar command after calling *make unpatch* is generally not a good idea.

After new patches have been commited on top of the patched branch (or existing commits since the base commit have been edited or removed), the patch directories can be regenerated using

```
make update-patches
```

If applying a patch fails because you have changed the base commit, the repository will be reset to the old *patched* branch and you can try rebasing it onto the new *base* branch yourself and after that call *make update-patches* to fix the problem.

Always call *make update-patches* after making changes to a module repository as *make update* will overwrite your commits, making *git reflog* the only way to recover them!

CHAPTER 4

Supported Devices

- TP-Link
  - TL-WR740N (v1, v3, v4)
  - TL-WR741N/ND (v1, v2, v4)
  - TL-WR841N/ND (v3, v5, v7, v8, v9)
  - TL-WR842N/ND (v1, v2)
  - TL-WR941N/ND (v2, v3, v4)
  - TL-WR1043N/ND (v1)
  - TL-WDR3500 (v1)
  - TL-WDR3600 (v1)
  - TL-WDR4300 (v1)
  - TL-WA901N/ND (v2)
  - TL-MR3020 (v1)
  - TL-MR3040 (v1)
  - TL-MR3220 (v1)
  - TL-MR3420 (v1, v2)
- Ubiquiti
  - Bullet M2
  - Nanostation M2
  - Picostation M2
  - UniFi AP
  - UniFi AP Outdoor
- D-Link
  - DIR-615 (E1)
  - DIR-825 (B1)
- Linksys
  - WRT160NL

CHAPTER 5

Releases

5.1 Gluon 2014.3

5.1.1 New hardware support

- Linksys WRT160NL

5.1.2 New features

New autoupdater

The autoupdater has been rewritten.

Two new fields have been added to the manifest:

DATE Specifies the time and date the update was released. `make manifest` will take care of setting it to the correct value.

PRIORITY Specifies the maximum number of days until the update should be attempted (thus lower numbers mean the priority is higher). It must be set either in `site.mk` or on the `make manifest` command line.

Updates will be attempted at night, between 04:00 and 5:00, with a specific probability. When less than `PRIORITY` days have passed (calculated using `DATE` and the current time), the probability will proportional to the time passed. I.e. the update probability will start at 0 and slowly increase to 1 until `PRIORITY` days have passed. From then, the probability will be fixed at 1.

Note: For the new update logic to work, a valid NTP server reachable over the mesh (using IPv6) must be configured in `site.mk`. If the autoupdater is unable to determine the correct time, it will fall back to a behavior similar to the old implementation (i.e. hourly update attempts).

Seperation of announced data

The data announced by alfred has been split into two data types:

- *nodeinfo* (type 158) contains all static information about a node
- *statistics* (type 159) contains all dynamic information about a node

Both types also contain a new field `node_id` which contains an arbitrary unique ID (currently the primary MAC address, sans colons) which can be used to match the *nodeinfo* with *statistics* information.

gluon-announced

A new daemon has been added in a new package `gluon-announced`. This daemon can be used for querying the *nodeinfo* data of a node via link-local multicast on the ad-hoc interfaces.

At the moment, this daemon is not used, but we recommend including it in `site.mk` nevertheless as we plan to implement a new status page showing some information about neighbor nodes in the next version of Gluon.

VPN over IPv6

It is now possible to use fastd in IPv6 WAN networks. This still needs testing, but it should work well.

Please note that the MTU of 1426 used by many communities for VPN over IPv4 is too big for IPv6 as the IPv6 header is 20 bytes longer (fastd over IPv4 has an overhead of 66 bytes, fastd over IPv6 has an overhead of 86 bytes).

More modular Config Mode

The package `gluon-config-mode` has been split into multiple packages to simplify the development of extensions. The low-level logic (handling of the button, starting the services for the config mode) has been moved into a new package `gluon-setup-mode`, while `gluon-config-mode` only contains the frontend now.

Extended Expert Mode

The Expert Mode now has a nice info page. In addition, the new package `gluon-luci-portconfig` has been added which allows simple configuration of batman-adv on the WAN interface.

Site validators

The content of the `site.conf` is now validated when the images are built to make it less likely to accidentially build broken images.

gluon-firewall

The package `gluon-firewall` has been removed. Its features are now part of the packages `gluon-core` and `gluon-mesh-batman-adv`.

gluon-ath9k-workaround

This package installs a cron job which tries to recognize ath9k hangs and restart the WLAN while recording some information. It is very rudimentary and we can't really recommend using it on "production" nodes.

5.1.3 Bugfixes

Improved ath9k stability

Multiple bugs in the WLAN driver ath9k have been fixed upstream. This should greatly improve the WLAN stability.

odhcp6c 50 day bug

An important update for odhcp6c fixes a bug which caused Gluon nodes to lose their IPv6 addresses on br-client after an uptime of 50 days, making the nodes unable perform automated updates (besides other issues).

IPv6 preference

Commands like `wget` now prefer IPv6 for domains with both AAAA and A records, allowing to use such domains for the autoupdater URLs and as NTP servers in `site.conf`.

5.1.4 Site changes

- `site.conf`
  - The `probability` fields for the autoupdater branches can be dropped as they aren't used anymore
  - The type of the `enabled` options of the `gluon-simple-tc` configuration has been changed to boolean, so `true` and `false` must be used instead of 1 and 0 now
- `site.mk`
  - Obsolete packages:
    * `gluon-firewall`
  - Recommended new packages:
    * `gluon-announced`
    * `gluon-luci-portconfig`
  - GLUON_PRIORITY must be set in `site.mk` or on the `make manifest` commandline. Use `GLUON_PRIORITY ?= 0` in `site.mk` to allow overriding from the commandline.

5.1.5 Internals

Some internal changes not mentioned before which are interesting for developers:

- Many more shell scripts have been converted to Lua
- `gluon-mesh-vpn-fastd` now uses the new package `gluon-wan-dnsmasq`, which provides a secondary DNS server on port 54 that is only reachable from *localhost* and uses the DNS servers on the WAN interface for everything. This allowed us to remove some ugly hacks which were making the DNS servers used depend on the domain being resolved.

  For IPv6, the default route is now controlled via packet marks, so the secondary DNS server and fastd set the packet mark so they use the default route provided on the WAN interface instead of the mesh.

CHAPTER 6

License

See LICENCE

CHAPTER 7

Indices and tables

- *genindex*
- *search*