LEDE Firmware optimization for wired deployments using BGP (Bird Daemon) and BMX for routing by enhancing and extending Bird Daemon's configuration and UI integration

THEALE, JUNE 2017

MAJOR: COMPUTER SCIENCE
MINOR: OPEN SOURCE SOFTWARE

AUTHOR:
ELOI CARBÓ SOLÉ

DIRECTOR:
JOAN MANUEL MARQUÈS PUIG
COMPUTER SCIENCE, MULTIMEDIA AND TELECOMUNICATIONS DEPARTMENT (DPCS-ICSO)

EXTERNAL CONSULTANT:
VÍCTOR ONCINS BIOSCA
ROUTEK SL

UNIVERSITAT OBERTA DE CATALUNYA
2017

Index

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Structure of the document

1.2 Motivation and description of the problem

This project aims to simplify and enhance management and monitoring capabilities of network administrators' using Bird Daemon software on top of an OpenWRT/LEDE-based Firmware. This project is a second iteration in the development of an existing configuration integration package already being used by OpenWRT/LEDE's community.

1.2.1 Motivation

Back in 2014, while working on my BSc. dissertation in the *Universitat Politècnica de Catalunya*, the department and, specifically, the investigation team I was working with, gave me the opportunity to participate in a GSoC[1] project under the umbrella of Freifunk, to design, develop and present a package that would help simplifying the configuration of Bird Daemon as a software able to share routes between BMX6 meshs and BGP infrastructure networks deployed in *frontier* nodes deployed in the Catalan community network Guifi.net.

That project was successful and the result was an integration package using OpenWRT's well-known UCI/LUCI configuration mechanism to set up Bird through a user-friendly Web UI even without deep knowledge of Bird's syntax. GSoC's time frame though was not enough to polish the package and add some secondary protocols and the package stopped getting maintenance from myself later that year. However, it has been an OSS project that has been on my *backlog* of things I want to keep improving and also been queried some times by Víctor Oncins as it is really helpful for network administrators

[1] Google Summer of Code (2014)

but it is not mature enough for complex production environments available in Guifi.net.

Therefore, I have been really lucky to have the opportunity to retake this package as my MSc. project while doing my MSc. and work together Víctor as this has meant that I have had direct feedback from administrators using the tool in production environments and to improve its most critical features. Moreover, Víctor has also published a report on GitHub [1] describing the main challenges found using the old version of the Package and a deep description of the environment.

1.2.2 Bird Daemon

Bird Daemon, from now onwards Bird, is an open source Internet Routing service (daemon) that allows network administrators to simplify route sharing configuration, management and monitoring by using Routing tables and a powerful filtering language[2].

1.2.3 OpenWRT/LEDE's configuration integration package

Bird-OpenWRT Package, from now onwards *the Package*, is an open source OpenWRT/LEDE-specific solution (*.ipk*) integrated by four separated packages (two for Bird IPv4 (*bird4-uci*) and IPv6 (*bird6-uci*) UCI integration and the other two for Web UI management (*luci-app-bird4* and *luci-app-bird6*) providing Bird Daemon a user-friendly configuration scheme (UCI) and a graphical interface in OpenWRT/LEDE-based routers.

1.2.4 Bird Daemon administration issues

As part of the GSoC project, the solution provided was not mature enough to fulfil all the requirements:

- Tight time-frame forcing to prioritise the key capabilities to implement.
- Some key protocols were not enabled in the final solution because they were not relevant for GSoC's scope (i.e. Pipe or Direct).
- Some secondary protocols were not enabled in the final solution because they were not relevant for GSoC's scope (i.e. OSPF or)
- Some basic processes require manual (terminal) changes
- No possible way to edit Filters or Functions files through Web UI.
- No Bird Daemon Status feedback (i.e. no way to know if bird is running or failed to start through Web UI).

[2]Bird Daemon: Link

- No possible way to see Bird Daemon's Log information through Web UI.
- Bird's API changed (from Bird 1.4.3 to 1.6.3) making bird crash using base Package configuration
- No possible way of monitoring Bird's current status (i.e. full information for BGP connections)

1.3 Scope of the project

This project's scope is to adopt as many of the mentioned enhancements that are clearly aligned with eradicating required manual changes in command line, improve the UX[3] and to align the packet with current Bird Daemon API in the given time frame of 3 months. As a result of a *backlog* prioritization, the following items were agreed (in priority order):

- Update the package to the latest Bird API.
- Update old version's disruptive issues (i.e. disabled Protocols).
- Status, Log, Filters and Functions Graphical integration.
- Theoretical viability investigation of uBus integration.

1.3.1 Deviations from the original plan and future work

While agreeing the original scope of the project, few extra ideas and tasks were planned but, as a matter of priorities and time constraints, some were dismissed or set as future work.

- Add secondary protocols: adopt more key features from Bird and increasing the range of administrators being able to take advantage of this Package.
- Integrate next generation of Web UI using LUCI2: HTML/JavaScript-based UI instead of LUA-based.
- Implementation of uBus integration according to the results of the investigation done in this project.
- Comparative set of tests between Quagga and Bird Daemon solutions.

[3]UX: User eXperience

Most of these extra tasks are already documented as part of the Package Documentation Reporitory[4] and open for discussion and Pull Requests[5] to add extra requirements.

Bird Daemon VS. Quagga deployments

There is a special reasoning behind not doing a comparative analysis of these two solutions. Of course, the timing constrains have strongly influenced the decision of dropping it from this project's scope, but there is also the big amount of evidence already collected for my GSoC project as well as some new evidence found either in some reputable sources as well as from Bird's own OSS Community proving that Bird Daemon has been far more stable, less resource eater and flexible (thanks to its Filter&Function scripting language) than other well-known enterprise level solutions. This evidence is available in the Appendix B.

1.3.2 Methodology and communication

This project starts with the premise that there is no need for a wide initial investigation phase as the Package used was designed and developed by myself. Nevertheless, there are three foreseen introductory tasks:

- Refresh the Package to the latest Bird Daemon version API.
- Investigate, understand and document the production environment.
- Update Documentation and prepare the repositories required (documentation, package and dissertation).

After this initial phase, the implementation tasks will be executed in a Kanban-like approach:

- Features will be executed following Backlog's priority order and one at a time.
- Each *feature/requirement* must be self-contained and the Package should be releasable at any time.
- There is no Board or framework to introduce the data (i.e. time spent or state of the tasks) as such as the overhead of doing it is not proportional to the number of tasks or value of the data that could be collected. However, during the first *Cycle* of the project (first two weeks), in order to illustrate how could this project look like using

[4] GitHub TODO List: Link.

[5] Pull Request: Changes pushed to a repository by an external party (i.e. a fork repository pushing changes to its parent.

Kanban, I did use an online OSS tool called *Taiga.io*[6]. See Appendix **??** in order to see some captures of the initial tasks created using the tool.

- There will be weekly/bi-weekly meetings with the Stakeholder in order to discuss progress, any blocker or issue and rearrange priorities if required.
- There will be a *demo* to the Stakeholder to show progress in a weekly/bi-weekly basis.

The communication, as already mentioned, will be done through regular meetings with the External Consultant (Stakeholder) using the Jitsi[7] conference service, which allows screen sharing and text communication while in conference, simplifying demoing and code reviews. Regular communication will be also done through Hangouts instant messaging service and by email to share progress, risks or blockers.

Gantt Diagram

Tasks' delivery forecast can be seen in Figures 1.1 and 1.2:

[6]Taiga.io: Online project management tool working either with Kanban or SCRUM Agile methodologies. This tool is widely used in OSS projects due to its power, simplicity and plugins (open API) and has also enterprise options.

[7]Jitsi: Open Source multiplatform VoIP conference service.

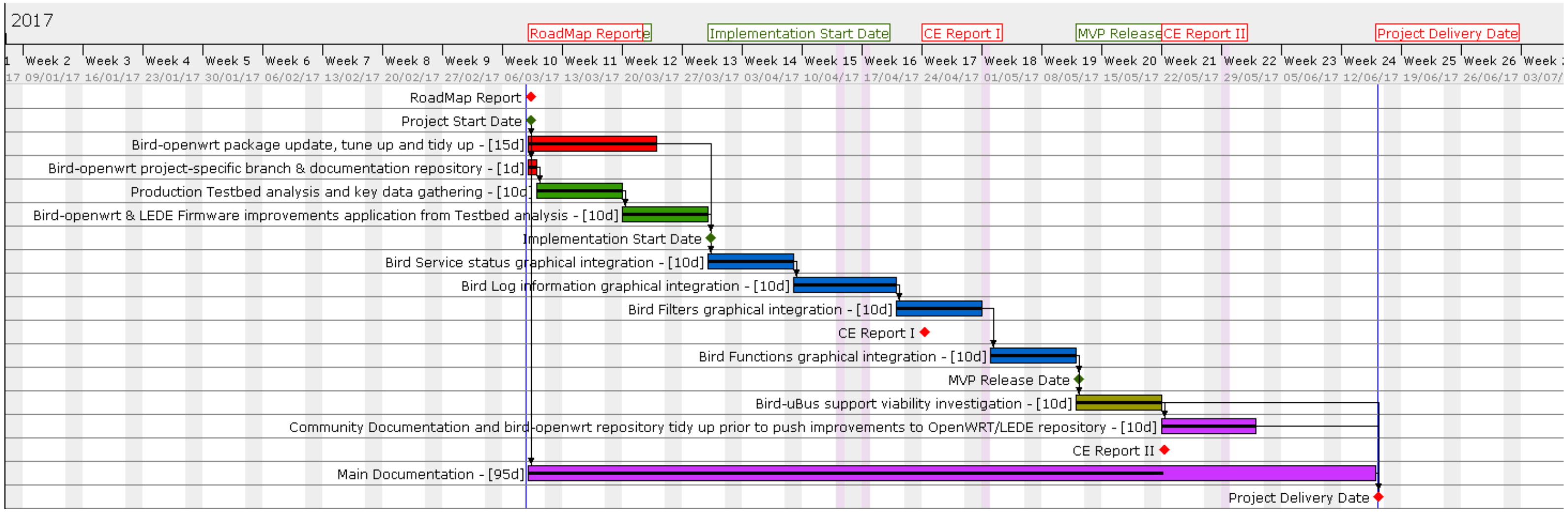

Figure 1.1: Tasks schedule

Key milestones:

- **Project's start date & RoadMap Report** (09/03/17): initial Package refresh and production environment investigation. Formal report of which are project's goals and when are they expected to be delivered.

- **Project's implementation start date** (30/03/17): beginning of features' implementation.

- **Continuous Evaluation Report I** (22/04/17): formal report to present Project's progress, pending work, any issue or blocker and updated timeline.

- **MVP Release** (12/05/2017): forecast delivery date of the final version of the Package. No extra changes planned unless the investigation task requires them.
- **CE Report II** (22/05/17): optional progress report prior to Project's delivery.
- **Project Delivery Date** (12/06/17): final date to deliver the dissertation, slides+recording and any extra archive required.

Duration	Name	Begin date	End date
0	RoadMap Report	09/03/17	09/03/17
0	Project Start Date	09/03/17	09/03/17
15	Bird-openwrt package update, tune up and tidy up - [15d]	09/03/17	23/03/17
1	Bird-openwrt project-specific branch & documentation repository - [1d]	09/03/17	09/03/17
10	Production Testbed analysis and key data gathering - [10d]	10/03/17	19/03/17
10	Bird-openwrt & LEDE Firmware improvements application from Testbed analysis - [10d]	20/03/17	29/03/17
0	Implementation Start Date	30/03/17	30/03/17
10	Bird Service status graphical integration - [10d]	30/03/17	08/04/17
10	Bird Log information graphical integration - [10d]	09/04/17	20/04/17
10	Bird Filters graphical integration - [10d]	21/04/17	30/04/17
0	CE Report I	24/04/17	24/04/17
10	Bird Functions graphical integration - [10d]	02/05/17	11/05/17
0	MVP Release Date	12/05/17	12/05/17
10	Bird-uBus support viability investigation - [10d]	12/05/17	21/05/17
10	Community Documentation and bird-openwrt repository tidy up prior to push improvements to OpenWRT/LEDE repository - [10d]	22/05/17	01/06/17
0	CE Report II	22/05/17	22/05/17
95	Main Documentation - [95d]	09/03/17	15/06/17
0	Project Delivery Date	16/06/17	16/06/17

Figure 1.2: Schedule details

1.4 Background information

1.4.1 Guifi.net

Guifi.net is a community network working for and by its own users (self-organised) giving an affordable alternative for anyone willing to connect to the Internet. This network's principles are freedom, open design, administration and management and neutrality. This network was born in Catalonia as a wireless network but it has spread all over the world and with about 33.124[8] active nodes (as 26/05/17) using roof antennas and optical fiber deployments.

This network is connected to the Catalan Internet Exchange Point (CATNIX[9]), has its own Government Foundation[10] to promote and protect network's principles defined in an operational and behavioural common regulation (Comuns - XOLN[11]) and it is open for any company that adheres to the XOLN/FONNC principles, to professionally operate and advertise itself as a Guifi.net Internet or services provider.

Finally, although Guifi.net main routing protocol is BGP for infrastructure and OSPF for internal routing, there are several isles[12][13] operating as Mesh Networks using BMX6 dynamic routing protocol.

1.4.2 OpenWRT/LEDE Project

OpenWRT, and its Fork LEDE-Project[14], are Open Source Linux-based firmwares primarily focused on commodity routers, but aiming to work in any Linux-based system. This firmware supports a wide variety of manufacturer's hardware and also a wide range of software, services and routing protocols to enhance, secure and efficiently operate as a standalone router and service provider.

1.4.3 Infrastructure vs Mesh Network Routing Protocols

Routing protocols' job is to receive a route and, according to its attributes and the information stored in the system, to redirect this route to the next step towards its destination or to drop it. However,

- **Infrastructure**: commonly used in structural networks. Stable, robust and scalable. Their main handicap is to suffer of big overheads

[8] Guifi.net live statistics: Link
[9] CATNIX: Link
[10] Fundació Guifi.net: Link
[11] XOLN/FONN: Compact for a Free, Open & Neutral Network
[12] Guifi.net Mesh Networks: Link
[13] qMp: Most commonly used firmware in Guifi.net for mesh networks. This OpenWRT fork aims to simplify and automate mesh deployments.
[14] LEDE-Project: Linux Embedded Development Environment.

on topology changes (i.e. low/non fault tolerance) and have big convergence times in large-scale networks.

- **Mesh Networks**: oppositely to classic dynamic networks, mesh networks' strength is to be able to converge almost instantly after any topology event. These networks work in a cooperative manner in order to achieve a fully connected network (point-to-multipoint) where all the nodes share network's knowledge in order to optimise routes and nodes floods the network in order to keep the network topology knowledge up to date.

BGP

BGP is a dynamic infrastructure IP routing protocol designed for large-scale internet topologies (EGP[15]). Its routing algorithm relies on the best path according to route's attributes.

BMX6

Batman-eXperimental6 [2] is a fork of the Mesh protocol BATMAN[16]. This is a mesh networking routing protocol is compatible with most of linux-like systems but only operates with IPv6 networks. This routing protocol uses a table-driven[17] distance-vector approach[18].

1.5 State of the art

[15]EGP: Exterior Gateway Protocol. This includes all the protocols that routes between Autonomous systems.

[16]B.A.T.M.A.N: Better Approach To Mobile Adhoc Networking.

[17]Table-driven: Compose a routing table with all the source-destination entries.

[18]Distance-vector: Best path (cost of going) from source to destination.

Chapter 2

Network Architecture

As shown in the figure 2.1, our targeted network is a mixed section requiring the use of Exterior Gateway Protocols (IGP) and Internal Gateway Protocols (EGP) in order be able to share routes routes between the two BGP ends (named *E* and *F*) going through a BMX6 Mesh Network (from report [1] - in Catalan).

As shown in the figure:

- **Infrastructure Super Node 1** (ISN1): BGP Supernode connected to the BGP network (Guifi.net, section 1) via wireless and to the MXN1 Router through Ethernet.
- **Mesh eXchange Node 1** (MXN1): LEDE/OpenWRT router connected via Ethernet to the ISN1 and to the antenna (or an Ethernet port) providing access to the Mesh Network. This frontier node provides BGP to BMX6 route-sharing capabilities using Bird Daemon.
- **Mesh Network**: A number of Nodes connected using BMX6 forming an isle between BGP nodes.
- **Mesh eXchange Node 2** (MXN2): LEDE/OpenWRT router connected via Ethernet to the ISN2 and to the antenna (or an Ethernet port) providing access to the Mesh Network. This frontier node provides BGP to BMX6 route-sharing capabilities using Bird Daemon.
- **Infrastructure Super Node 2** (ISN1): BGP Supernode connected to the BGP network (Guifi.net, section 2) via wireless and to the MXN2 Router through Ethernet.

2.1 Routing requirements

Routing requirements to successfully ensure that all routes are shared between both BGP ends are:

- Routes must be shared/announced between ISN1 (**E**) and ISN2 (**F**).
- Mesh Network's Routes (BMX6 - **C**&**D**) must be shared/announced to ISN1 and ISN2 (**A**&**B**). Therefore, shared/announced to Guifi.net network.
- ISN1 and ISN2 Routes (BGP - **A**&**B**) must be shared/announced to the Mesh Network (**C**&**D**).
- MXN1/2 must configure Bird to use a custom Routing Table that will be shared with BMX6.
- MXN1/2 must configure BMX6 to use the *Table* plugin in order to redirect its routes from Kernel's Table to a custom one.
- MXN1/2 must configure Bird to set them both as BGP Peers to stablish an iBGP session between them (AS2).
- MXN1 must configure Bird to set ISN1 as BGP Peer AS1
- MXN2 must configure Bird to set ISN1 as BGP Peer AS3

2.1.1 Caveats

There is an important caveat with this network distribution:

Current version of BMX6 is not able to handle the number of routes that this Guifi.net BGP section is sharing (2.500+). Therefore, BMX6 starts aggregating routes, which eventually shut-downs the service and leaves the node overloaded as it is not able to achieve it.

In order to avoid this disruptive issue, Bird Daemon filter scripting capabilities available in MXN1 and MXN2 allow to reduce the geographical scope of the routes imported and exported to/from the Barcelonès[1] Zone.

[1]Barcelonès: Network Zone including Badalona, Barcelona, Hospitalet del Llobregat, Sant Adrià del Besos and Santa Coloma de Gramanet

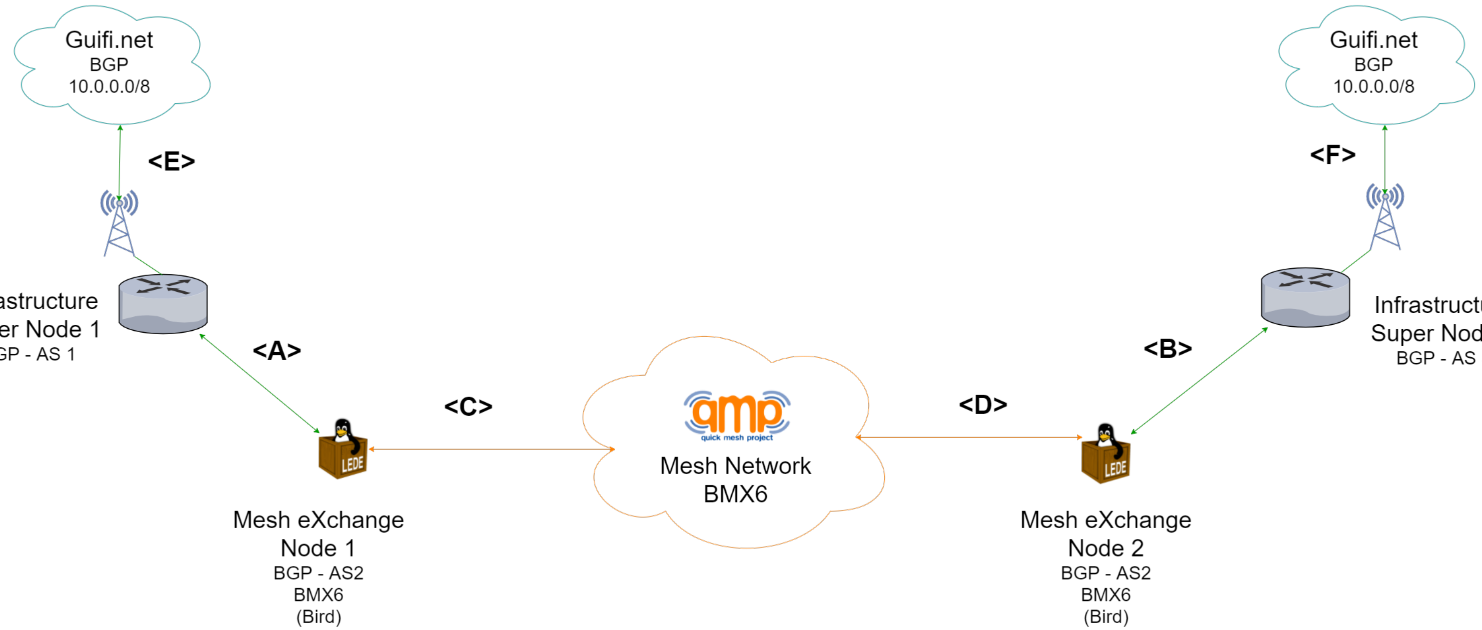

Figure 2.1: Production Network targeted in this project

2.2 Real environment testing

Although it has not been possible to test Package's improvements due to the impact that a change like this could incur in a live network involving several critical *Super Nodes* and all the routes in the Mesh Network, it is foreseen to be updated soon in a planned and backed up manner.

2.2.1 Development environment testing

The *Universitat Oberta de Catalunya* and Víctor have provided me a number of Virtual Machines plus a number of network resources in order to simulate our target network but without the risk of damaging the production network or flooding unwanted routes to Guifi.net.

- VPN access to the Guifi Network using UOC's resources.
- 4 Virtual Machines using LEDE17.01 Firmware.
- Virtual Bridge to connect the VMs simulating a Mesh Network.
- Network way through two different network sections.
  - Connection using UOC's Super Node
  - Virtual Machine 4 connects through UPF[2]'s internal network to find path out to a near Guifi network.

[2]Universitat Pompeu Fabra

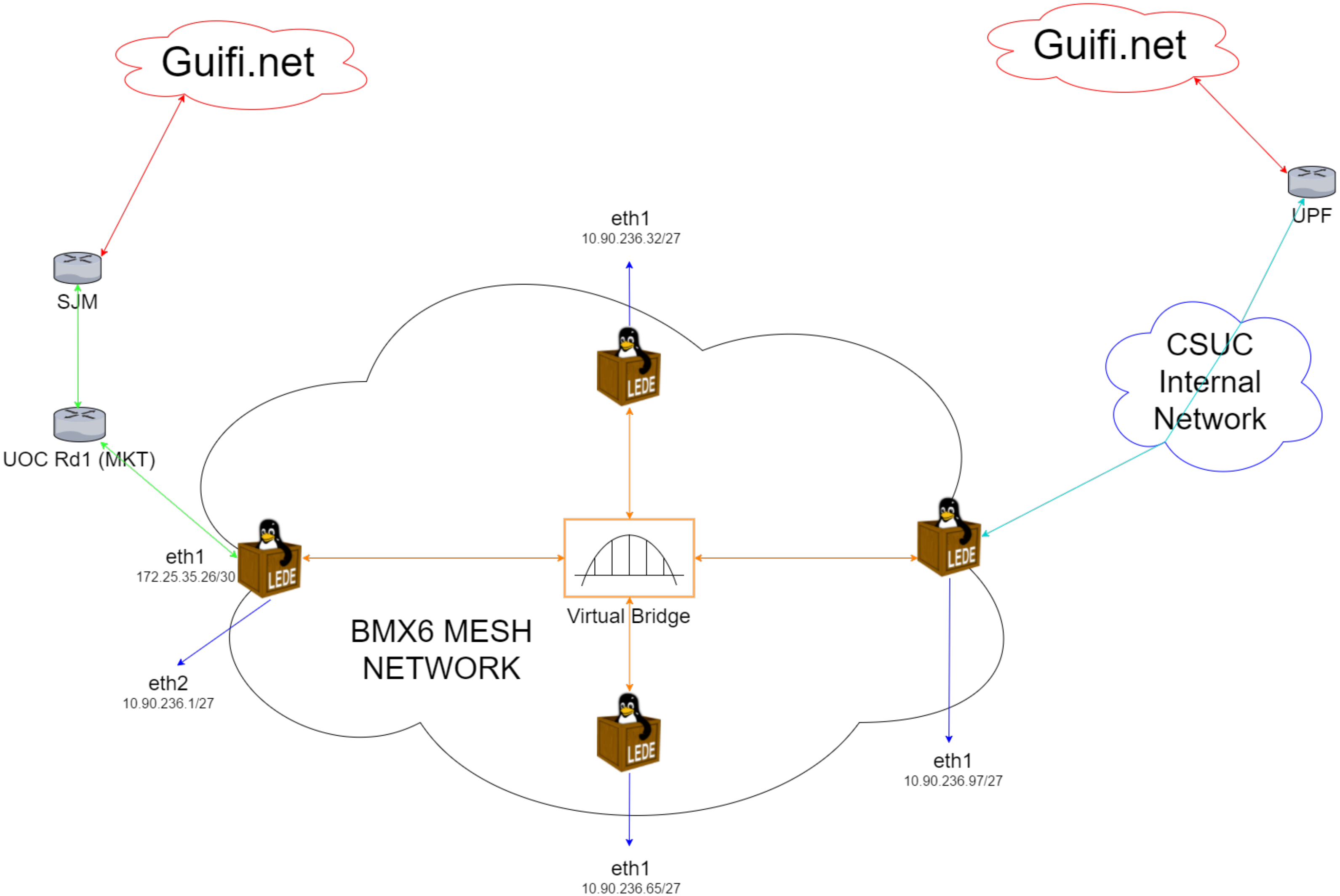

Figure 2.2: Development Network simulating production's environment

Chapter 3

Package improvements implementation

3.1 Administration requirements

One of the administrator's main tasks while managing networks is, if necessary, to facilitate the coexistence of different Routing protocols in the same network section. Hence the requirement of rich tools as Quagga or Bird to act as facilitators of this route transference between them.

Administrators require:

- A full-featured tool with an easy and intuitive UI to manage and monitor protocol health and data efficiently and avoiding any handmade/-custom edit, reducing configuration's complexity.
- Use LEDE/OpenWRT-based firmwares widely used in the target network.
- A Routing protocols management tool that, at least, supports BGP Static Routing Protocol and it is able to share routes with BMX6 Dynamic Routing Protocol in a manageable way.
- Use Bird Daemon instead of Quagga to make us of its proven efficiency, low resource consumption and powerful filter capabilities, which is critical to some of the widely used commodity hardware in Guifi.net.
- Use and improve Bird Daemon's configuration integration package (bird-uci and luci-app-bird) available in the official Routing Repository of LEDE/OpenWRT.
- Avoid project-specific customisations in the integration package that would not benefit all the community. If required, add those custom enhancements in a development branch.

- Update Package's documentation and create new topics to cover Web UI interface and any manual process not covered by package's improvements.
- Update Bird integration package in order to be compliant to the latest API (v1.6.3 when this document was written).
- Enhance Web UI to support user-friendly configuration and visualisation of the following:
  - Bird Daemon service status
  - Bird Daemon events information (Logs)
  - Filters and Functions editing using an embedded HTML text editor
  - Update old Web UI pages to
- Do theoretical viability investigation to use uBus Daemon as a mechanism to communicate with Bird Daemon and get health information and current-status information for handled protocol using JSON messages.

3.2 Changes in the code

3.2.1 Apply code standards

3.2.2 init.d script and service management

3.2.3 UCI Configuration improvements

3.2.4 LUCI UI improvements

Status Page

Log Page

Filters & Functions Page

BGP and Classic protocols Pages

3.2.5 Align documentation and upgrade to Markdown

3.3 Package Testing

Testing an integration/translation Package, and this one specifically, is a rather complex task to evaluate as Bird configuration files are modular and desired settings can be achieved in different ways. Even more, although a *it works/it does not work* policy could be accepted, it does not mean that there are not other possible implementations that could work in a better way. For example, filters and functions can be either written in the *.conf* file or included using **%include** mechanism, being the second one a better approach as it enhances code readability as well as it avoids bloating the configuration file unnecessarily.

With this introduction in mind, the following sections will explain how this package has been tested following Bird's configuration base requirements and service behaviour and some *future work* ideas to achieve automatic and unit tests.

3.3.1 Configuration Translation Tests (future work)

To perform configuration integrity tests in current package, it is required to repeat the execution of `/etc/bird{4|6}/init.d/bird4 restart` in order to trigger the UCI-bird.conf translation from a target UCI file. The code to do this translation has been refactored in an functional manner to allow future unit tests or, at least, make it easier to integrate in an automated test framework or process. For example, an automated CI/CD build process could build an update of the package, push it into a test node, execute the translation process and compare it against the previous (or a stable) version as well as check its correctness by querying Bird's status.

Reviewing v0.2 against v0.3

Testing the outputs from the old and new packages, and taking into account that there are some manual changes in the old one, the following example is configured as follows:

- Router IDs follow node's IP Address
- Kernel, Device and Static Protocols have been set by default
- A Static Route has been added (identical)
- BGP Template and Instance have been configured following v0.2 scheme with matching settings to avoid Bird failures
- BGP Instance AS and Neighbours are dummy values
- A BGP Filter called "all_ok" (accept all routes) has been added using each version's process.

In the new package, we have instantaneous configuration correctness feedback as we can check Bird's status in the Status Page. In the old package, after executing `/etc/bird{4|6}/init.d/bird4 start`, Bird will fail and it is required to move the Filter "all_ok" to the top of the document. Bird will start correctly after this modification.

After checking that both daemons are running, we can then perform a *diff* between the configuration files and look for any noticeable difference

```
3,9d2
<     #Filter filter1:
<     filter all_ok
<     {
<         accept "all ok";
<     }
<
13c6
<     router id 192.168.1.200;
---
>     router id 192.168.1.100;
17a11,17
>     #Functions Section:
>     #End of Functions --
>
>     #Filters Section:
>     include "/etc/bird4/filters/filter1";
>     #End of Filters --
>
19c19
<     protocol kernel {
---
>     protocol kernel kernel1 {
46c45
<     source address 192.168.1.200;
---
>     source address 192.168.1.100;
57c57
<     neighbor 192.168.1.201 as 1002;
---
>     neighbor 192.168.1.101 as 1002;
```

Listing 3.1: Battlemesh experiment code

As shown in this *diff* snippet, almost all the translated configuration is identical apart from:

- Different Router IDs and BGP neighbours (expected)
- Kernel Protocol definition (minor change in the API)
- BGP Filter definition (major change in the API)

3.3.2 Bird Daemon Errors

Bird Daemon provides an error exit code together with different text outputs in order to highlight errors in the configuration. Although most of the times it can be easily spotted using Bird's feedback, there are also instances where the Daemon's documentation may be required to fix them.

Bird Daemon Error examples

Most common errors that an administrator may need to resolve are:

- A configured field has incorrect syntax. Bird will give you hints about what is wrong most of the times: wrong IP address format `bird: /tmp/bird4.conf, line 7: Invalid IPv4 address 1921.68.1.1`. But some *rare* times the message is less helpful and you may need to check the contents of the file and understand the error.

  As an example of this: `bird4: Failed - bird: /tmp/bird4.conf, line 65: syntax error`. We need to check the bird4.conf file and see that in line 65:

```
64:    protocol bgp BGPExample {
65:        import Filter NonExistingFilter;
66:    }
```

Listing 3.2: Bird4.conf contents

  We will need to find out that the shown filter used in the **import** field of BGP Protocol, does not exist.

- Non-compatible configuration. The other set of common errors is non-compatible fields in a Protocol.

  As an example of this: `bird: /tmp/bird4.conf, line 76: Only internal neighbor can be RR client`. We need to remove the Route Reflector Client setting from the BGP Instance to fix this behaviour.

- Missing filter or function If you include a filter name in any of the Protocols or if any of your filters use a non-existing function, Bird will fail to start showing an error as follows: `bird: /tmp/bird4.conf, line 71: No such filter`.

- Syntax errors in a filter or function. This error follows the same approach as the first bullet: `bird: /etc/bird4/filters/filter-20170507-0951, line 4: syntax error`. You are required to go to command line and fix the problem checking the configuration and filter or function files.

- Filter calling to non-existing functions. If your filter executes a command that is not defined by Bird's syntax, it will handle it as a function. If that function does not exist in any of the handled files, it will show this error: `bird: /tmp/bird4.conf, You can't call something which is not a function. Really.`

- Filters not accepting/rejecting routes. Bird Daemon filters must return an *accept* or *reject* policy per route received. If any of your filters does not return any policy per route, it will be silently ignored and substituted with an "accept".

  As an example of this issue:

```
filter doNothing
{
    print "HelloWorld";
}
```

Listing 3.3: Filter printing message

Bird Daemon will succeed starting up but, if we check the log information in the Log Page, this error message will be shown:

```
<ERR> Filter doNothing did not return accept nor reject. Make
   up your mind
<INFO> HelloWorld
```

Listing 3.4: Filter printing message

3.3.3 Real Scenario: VM with simple BGP configuration connected to Guifi.net

As part of the acceptance tests, a VM was set up by a sysadmin in the *Universitat Oberta de Catalunya* to act as a pre-production machine. This VM is connected to a *Mikrotik* Router acting as Gateway to *Guifi.net* but this scenario does **not** connect or communicate through any Mesh Network using BMX6, so it is an end point.

The configuration of this system is almost identical, component-wise, to the ones available in Guifi.net. However, this system will only route itself (1 route) and import any.

Bird UCI configuration set through the WEB UI and its translation into Bird4 configuration can be reviewed in appendix A.

This VM is communicating to Guifi.net through a Mikrotik which is already doing some filtering but, in any case, it is still able to import 3000+ Routes and export itself:

```
root@LEDE-eloi:~# birdcl4 show protocols all
[...]
BGPImportALL BGP      master   up     2017-05-10  Established
  Preference:     100
  Input filter:   ebgp_in
```

```
  Output filter:  ebgp_out
  Import limit:   3000 [HIT]
    Action:       warn
  Routes:         2999 imported, 1 exported, 2999 preferred
  Route change stats:     received   rejected   filtered
     ignored   accepted
    Import updates:        1208383          0          0
              88    1208295
    Import withdraws:       337268          0        ---
             300     336968
    Export updates:        1208298    1208295          2
             ---          1
    Export withdraws:       336968        ---        ---
             ---          0
  BGP state:          Established
    Neighbor address: 172.25.35.25
    Neighbor AS:      59361
    Neighbor ID:      10.90.224.65
    Neighbor caps:    refresh AS4
    Session:          external AS4
    Source address:   172.25.35.26
    Route limit:      2999/3000
    Hold timer:       160/180
    Keepalive timer:  29/60
```

Listing 3.5: UCI Configuration

Using Bird Lightweight Remote Control (**birdcl4**) we can verify Bird's BGP instance. As key information:

- BGP Instance: BGPImportALL
- Filters applied: *ebgp_in* and *bgp_out*
- We are connected to our neighbour 10.90.224.65 with Autonomous System ID 59361
- The number of routes received fluctuates but the data shown presents 2999 routes imported.
- We do not know when, but the import Limit reached (HIT) and that generated warnings. From our Package's Log Page: `2017-05-21 22:09:13 <WARN> Protocol BGPImportALL hits route import limit (3000), action: warn`
- We are exporting 1 Route.

As a health check, we can query Bird of its last reconfiguration, reboot time or status using `bircl4 status`:

```
root@LEDE-eloi:~# birdcl4 show status
BIRD 1.6.3 ready.
```

```
BIRD 1.6.3
Router ID is 10.139.173.161
Current server time is 2017-05-22 00:20:23
Last reboot on 2017-05-10 19:31:09
Last reconfiguration on 2017-05-10 19:31:09
Daemon is up and running
```

Listing 3.6: UCI Configuration

3.3.4 Full Network Virtual Environment

Appendices

Appendix A

Bird Daemon's Configuration using v0.3 Package - UOC's VM in Guifi.net

A.1 UCI Configuration

```
config bird 'bird'
        option use_UCI_config '1'
        option UCI_config_file '/tmp/bird4.conf'
        option UCI_config_File '/tmp/bird4.conf'

config global 'global'
        option log_file '/tmp/bird4.log'
        option router_id '10.139.173.161'
        option log 'all'

config table
        option name 'aux'

config kernel 'kernel1'
        option import 'all'
        option export 'all'
        option scan_time '10'
        option learn '1'
        option disabled '0'

config device 'device1'
        option scan_time '10'
        option disabled '0'

config bgp_template 'BGP_COMMON'
        option receive_limit_action 'warn'
        option local_as '92099'
        option igp_table 'bgpTable'
        option export_limit_action 'warn'
```

```
        option import_limit_action 'warn'
        option next_hop_self '0'
        option next_hop_keep '0'
        option rr_client '0'

config table
        option name 'bgpTable'

config bgp 'BGPImportALL'
        option receive_limit_action 'warn'
        option template 'BGP_COMMON'
        option neighbor_as '59361'
        option neighbor_address '172.25.35.25'
        option export_limit_action 'warn'
        option import_limit_action 'warn'
        option import_limit '3000'
        option import 'filter ebgp_in'
        option export 'filter ebgp_out'
        option next_hop_self '0'

config kernel 'Kernel_BGP'
        option disabled '0'
        option table 'bgpTable'
        option kernel_table '251'
        option scan_time '10'
        option learn '1'
        option import 'all'
        option export 'all'

config pipe 'pipe1'
        option disabled '0'
        option peer_table 'bgpTable'
        option table 'aux'
        option import 'all'
        option export 'all'
        option mode 'transparent'

config direct 'direct1'
        option disabled '0'
        option interface '"br-lan","br-wan", "br-mgmt"'

config static 'static1'
        option disabled '0'
        option table 'aux'
```

Listing A.1: UCI Configuration

A.2 Bird Configuration

```
#Bird4 configuration using UCI:

log "/tmp/bird4.log" all;
```

```
#Router ID
router id 10.139.173.161;

#Secondary tables
table aux;
table bgpTable;

#Functions Section:
include "/etc/bird4/functions/function-20170507-1038";
#End of Functions --

#Filters Section:
include "/etc/bird4/filters/filter-20170507-0951";
#End of Filters --

#kernel1 configuration:
protocol kernel kernel1 {
#   disabled;
    learn;
    persist;
    scan time 10;
    import all;
    export all;
}

#Kernel_BGP configuration:
protocol kernel Kernel_BGP {
#   disabled;
    table bgpTable;
    kernel table 251;
    learn;
    persist;
    scan time 10;
    import all;
    export all;
}

#static1 configration:
protocol static {
    table aux;
}

#device1 configuration:
protocol device {
#   disabled;
    scan time 10;
}

#direct1 configuration:
protocol direct {
#   disabled;
    interface "br-lan","br-wan", "br-mgmt";
}
```

```
#pipe1 configuration:
protocol pipe pipe1 {
#   disabled;
    table aux;
    peer table bgpTable;
    mode transparent;
    import all;
    export all;
}

#BGP_COMMON template:
template bgp BGP_COMMON {
    local as 92099;
#    next hop self;
#    next hop keep;
    igp table bgpTable;
#    rr client;
}

#BGPImportALL configuration:
protocol bgp BGPImportALL from BGP_COMMON {
    import filter ebgp_in;
    export filter ebgp_out;
#    rr client;
    import limit 3000 action warn;
    neighbor 172.25.35.25 as 59361;
}


=======================================
BGP Filters and Functions:

root@LEDE-eloi:~# cat /etc/bird4/filters/filter-20170507-0951
filter ebgp_in {

        krt_prefsrc = 10.139.173.161;

        if match_guifi_prefix() then accept;
        reject;
}

filter ebgp_out {

        if match_guifi_prefix() then accept;
        reject;


root@LEDE-eloi:~# cat
   /etc/bird4/functions/function-20170507-1038
function match_guifi_prefix()
{
        return net ~ [ 10.0.0.0/8{9,32} ];
}
```

Listing A.2: Bird4.conf Configuration

Appendix B

Bird Daemon presence in Worldwide IXPs and other institutions

Bibliography

[1] "Integració entre BMX6 i BGP en dispositius basats en LEDE." `https://github.com/guifi-exo/doc/blob/master/knowledge/bmx6-bgp-lede.md`.

[2] A. Neumann, E. López, and L. Navarro, "An evaluation of bmx6 for community wireless networks.," in *WiMob*, pp. 651–658, IEEE Computer Society, 2012.