

---

# **ARouteServer Documentation**

*Release latest*

**Pier Carlo Chiodi**

**Jun 23, 2018**



---

## Contents

---

<b>1</b>	<b>How it works</b>	<b>3</b>
<b>2</b>	<b>Features</b>	<b>5</b>
<b>3</b>	<b>Contents</b>	<b>7</b>
<b>4</b>	<b>Presentations</b>	<b>73</b>
<b>5</b>	<b>Who is using ARouteServer?</b>	<b>75</b>
<b>6</b>	<b>Status</b>	<b>77</b>
<b>7</b>	<b>Bug? Issues? Support requests?</b>	<b>79</b>
<b>8</b>	<b>Author</b>	<b>81</b>



A Python tool to automatically build (and test) feature-rich configurations for BGP route servers.



1. Two YAML files provide *general policies* and *clients configurations* options:

```
cfg:
  rs_as: 64496
  router_id: "192.0.2.2"
  add_path: True
  filtering:
    next_hop:
      policy: "same-as"
  blackhole_filtering:
    policy_ipv4: "rewrite-next-hop"
  ...
```

```
clients:
- asn: 64511
  ip:
  - "192.0.2.11"
  - "2001:db8:1:1::11"
  irrdb:
    as_sets:
      - "RIPE::AS-FOO"
  ...
```

2. ARouteServer acquires external information to enrich them: i.e. [bgpq3](#) for IRR data, [PeeringDB](#) for max-prefix limit and AS-SETs, ...
3. [Jinja2](#) built-in templates are used to render the final route server's configuration file.

Currently, **BIRD** (1.6.3 and 1.6.4) and **OpenBGPD** (OpenBSD 6.0 up to 6.3) are supported, with almost feature parity between them.

**Validation** and testing are performed using the built-in **live tests** framework: [Docker](#) instances are used to simulate several scenarios, and more custom scenarios can be built on the basis of the user's needs. More details on the [Live tests](#) section.





- **Path hiding** mitigation techniques (RFC7947 section 2.3.1).
- Basic filters (mostly enabled by default):
  - **NEXT\_HOP** enforcement (strict / same AS - RFC7948 section 4.8);
  - minimum and maximum IPv4/IPv6 **prefix length**;
  - maximum **AS\_PATH length**;
  - reject **invalid AS\_PATHs** (containing *private/invalid* ASNs);
  - reject AS\_PATHs containing **transit-free** ASNs;
  - reject **bogons**;
  - **max-prefix limit** based on global or client-specific values or on **PeeringDB** data.
- Prefixes and origin ASNs validation (also in *tag-only* mode):
  - **IRR-based filters** (RFC7948 section 4.6.2);
  - AS-SETs configured manually or fetched from PeeringDB;
  - support for **IRR sources** (RIPE::AS-FOO, RADB::AS-BAR);
  - **white lists** support;
  - extended dataset for filters generation:
    - \* **RPKI ROAs used as route objects**;
    - \* **Origin AS** from **ARIN Whois** database dump;
  - **RPKI-based filtering** (BGP Prefix Origin Validation).
- **Blackhole filtering** support:
  - optional **NEXT\_HOP rewriting**;
  - signalling via BGP Communities (**BLACKHOLE** and custom communities);
  - client-by-client control over propagation.

- **Graceful shutdown** support:
  - honor the **GRACEFUL\_SHUTDOWN** BGP community received from clients ([draft-ietf-grow-bgp-gshut-11](#));
  - allow to perform a graceful shutdown of the route server itself.
- Control and informative communities:
  - prefix/origin ASN present/not present in **IRRDBs data**;
  - do (not) announce to any / **peer** / on **RTT basis**;
  - **prepend** to any / **peer** / on **RTT basis**;
  - add **NO\_EXPORT** / **NO\_ADVERTISE** to any / **peer**;
  - custom informational BGP communities.
- Optional session features on a client-by-client basis:
  - prepend route server ASN ([RFC7947 section 2.2.2.1](#));
  - active sessions;
  - **GTSM** (Generalized TTL Security Mechanism - [RFC5082](#));
  - **ADD-PATH** capability ([RFC7911](#)).
- Automatic building of clients list:
  - [integration](#) with **IXP-Manager**;
  - [fetch lists](#) from **PeeringDB** records and **Euro-IX member list JSON** files.
- **IX-F Member Export JSON files creation**.
- Related tools:
  - [Invalid routes reporter](#), to log or report invalid routes and their reject reason.

A comprehensive list of features can be found within the comments of the distributed configuration file on [GitHub](#) or on the [documentation web page](#).

More feature are already planned: see the [Future work](#) section for more details.

## 3.1 Installation

### 3.1.1 Dependencies

Some components used by ARouteServer need Python dev header files and static libraries: some distributions have them already included, others may need to manually install them:

```
# Debian Jessie, Ubuntu Trusty
apt-get install python-dev # for Python 2
apt-get install python3-dev # for Python 3

# CentOS
yum -y install gcc python-devel
```

Please note that ARouteServer also needs `bgpq3` to build IRR-based filters: details on its installation can be found within the *External programs* section.

### 3.1.2 Install using pip (suggested)

If you plan to just use the program to build configurations or to run your own live tests scenarios, you can install it using `pip`.

Strongly suggested: setup a `Virtualenv`.

```
# on Debian/Ubuntu:
sudo apt-get install python-pip python-virtualenv

# on CentOS:
sudo yum install epel-release
sudo yum install python-pip python-virtualenv
```

(continues on next page)

(continued from previous page)

```
# setup a virtualenv
mkdir -p ~/.virtualenvs/arouteserver
virtualenv ~/.virtualenvs/arouteserver
source ~/.virtualenvs/arouteserver/bin/activate

# install the program
pip install arouteserver
```

More: [virtualenv installation and usage](#).

---

**Note:** If you receive the following error while installing the program (or its requirements): **error in setup command: 'install\_requires' must be a string or list of strings containing valid project/version requirement specifiers** then please upgrade the *setuptools* package that is used in your virtualenv: `pip install --upgrade setuptools`.

---

---

**Note:** In the case the pip installation process breaks with the **Failed building wheel for py-radix / fatal error: Python.h: No such file or directory** error, please verify that the dependencies are satisfied.

---

### 3.1.3 Install from GitHub

If you plan to run built-in *Live tests* on your own or to contribute to the project, clone the GitHub repository locally and install dependencies:

```
mkdir -p ~/src/arouteserver
cd ~/src/arouteserver

# use the URL of your fork here:
git clone https://github.com/USERNAME/arouteserver.git ./

export PYTHONPATH="`pwd`"
pip install -r requirements.txt
```

### 3.1.4 Setup and initialization

- Setup your system layout (confirmation will be asked before each action):

```
# if you used pip
arouteserver setup

# if you installed from GitHub
export PYTHONPATH="`pwd`"
./scripts/arouteserver setup
```

The program will ask you to create some directories (under `~/arouteserver` by default) and to copy some files there. These paths can be changed by editing the `arouteserver.yml` program configuration file or by using command line arguments. More information in the [configuration section](#).

- Define the route server configuration policies, using the `configure` command or manually by editing the `general.yml` file:

```
# if you used pip
arouteserver configure

# if you installed from GitHub
./scripts/arouteserver configure
```

The `configure` command asks some questions about the route server environment (ASN, router ID, local subnets) and then it builds a policy definition file based on best practices and suggestions which also includes a rich BGP communities list.

### 3.1.5 External programs

ARouteServer uses the following external programs:

- (mandatory) `bgpq3` is used to gather information from IRRDBs.

To install it:

```
mkdir /path/to/bgpq3/directory
cd /path/to/bgpq3/directory
git clone https://github.com/snar/bgpq3.git ./
# make and gcc packages required
./configure
make
make install
```

- (optional) `Docker` is used to perform *live validation* of configurations.

To install it, please refer to its [official guide](#).

- (optional) `KVM` is also used to perform *live tests* of OpenBGPD configurations on an OpenBSD virtual machine.

To install it:

```
apt-get install qemu-kvm virtinst
```

More details: <https://wiki.debian.org/KVM>

- (optional) `rtrlib` and `bird-rtrlib-cli`; ARouteServer can use these tools to load RPKI data into BIRD. More details in [ROAs sources](#).

To install them:

```
curl -o rtrlib.zip -L https://github.com/rtrlib/rtrlib/archive/v0.3.6.zip
unzip rtrlib.zip

cd rtrlib-0.3.6 && \
  cmake -D CMAKE_BUILD_TYPE=Release . && \
  make && \
  make install

curl -o bird-rtrlib-cli.zip -L https://github.com/rtrlib/bird-rtrlib-cli/archive/
↪v0.1.1.zip
unzip bird-rtrlib-cli.zip

cd bird-rtrlib-cli-0.1.1 && \
  cmake . && \
  make
```

More details: <https://github.com/rtrlib/rtrlib/wiki/Installation>

To configure bird-rtrlib-cli please refer to the [README](#).

### 3.1.6 Upgrading

Often upgrades bring new features and new options, sometimes they also introduce changes that might break backward compatibility with previous versions. It is advisable to always check the *CHANGELOG* to verify what's new: the `arouteserver show_config` command can also be used to verify if new configuration options are available and how they are set by default.

To upgrade the program, download the new version...

```
# if you cloned the repository from GitHub,  
# from within the local repository's directory:  
git pull origin master  
  
# if you installed it with pip:  
pip install --upgrade arouteserver
```

... then sync the local templates with those distributed in the new version:

```
arouteserver setup-templates
```

If local templates have been edited, make a backup of your files in order to merge your changes in the new ones later. To customize the configuration of the route server with your own options, please consider using *Site-specific custom configuration files* instead of editing the template files.

### 3.1.7 Development and pre-release versions

---

**Note:** Consider your needs carefully before using a version other than the current production versions. These are preview releases, and their use is not recommended in production settings.

---

The **dev** branch is used for the development of the project, while the **master** branch always contains the latest, (hopefully) stable production-ready code.

To install or to upgrade to the latest pre-release version use the [TestPyPI](#) instance of the Python Package Index (PyPI):

```
pip install --index-url https://test.pypi.org/simple/ --extra-index-url https://pypi.  
↳org/simple --pre arouteserver
```

### 3.1.8 Ansible role

An Ansible role to install and configure ARouteServer can be found on [Galaxy](#) or on [GitHub](#).

It is tested on Debian (Jessie, Stretch), Ubuntu (Trusty, Xenial) and CentOS 7.

## 3.2 Usage

The script can be executed via command-line:

```
# if cloned from GitHub, from the repository's root directory:
export PYTHONPATH="`pwd`"
./scripts/arouteserver bird --ip-ver 4 -o /etc/bird/bird4.conf

# if installed using pip:
arouteserver bird --ip-ver 4 -o /etc/bird/bird4.conf
```

It produces the route server configuration for BIRD and saves it on `/etc/bird/bird4.conf`. To build the configuration for OpenBGPD, the `bird` sub-command must be replaced with `openbgpd`.

The `--target-version` argument can be used to set the version of the target BGP daemon for which the configuration is generated: this allows to enable features that are supported only by more recent versions of BGP speakers and that, otherwise, would produce an error.

The script exits with 0 if everything is fine or with an exit code different than zero if something wrong occurs.

It can be scheduled at regular intervals to re-build the configuration (for example to add new clients or to update IRRDB information), test it and finally to deploy it in production:

```
# The following assumes that ARouteServer runs on the
# route server itself, that is a thing that you may want
# to avoid.
arouteserver bird --ip-ver 4 -o /etc/bird/bird4.new && \
  bird -p -c /etc/bird/bird4.new && \
  cp /etc/bird/bird4.new /etc/bird/bird4.conf && \
  birdctl configure
```

### 3.2.1 Route server graceful shutdown

Prior to a maintenance that requires the route server shutdown a graceful shutdown can be triggered by using the `--perform-graceful-shutdown` argument. This option allows to build a temporary configuration that includes an outbound policy which is applied to BGP sessions toward the clients and which adds the `GRACEFUL_SHUTDOWN` BGP community (65535:0) to all the routes that the route server announces to them.

Please note that the configuration built when using this argument should be used only **temporarily** before starting the maintenance; it should be **replaced** with the **production configuration** before the route server is reloaded.

### 3.2.2 Resources and MemoryError error messages

When building large configurations, for example those generated when huge AS-SETS need to be expanded, the program may crash with a `MemoryError` message or other memory related exceptions. In this case, raising `ulimits` for max locked memory (`-l`) and stack size (`-s`) has proven to be effective in solving the problem:

```
$ ulimit -l 2097152; ulimit -s 8192; arouteserver openbgpd ...
```

### 3.2.3 Library

ARouteServer can be used as a Python library too: see `LIBRARY` for more details.

### 3.2.4 Textual representation

To build an HTML textual representation of route server's options and policies, the `html` command can be used:

```
arouteserver html -o /var/www/html/rs_description.html
```

This command writes an HTML page that contains a brief textual representation of route server's policies. An example can be found [here](#).

### 3.2.5 Automatic `clients.yml` creation

#### Create `clients.yml` file from PeeringDB records

The `clients-from-peeringdb` command can be used to automatically create a `clients.yml` file on the basis of PeeringDB records. Given an IX LAN ID, it collects all the networks which are registered as route server clients on that LAN, then it builds the clients file accordingly.

If the IX LAN ID argument is not given, the script uses the [IX-F database](#) to show a list of IXPs and their PeeringDB ID; this can be used to easily search for the IXP PeeringDB ID.

```
$ arouteserver clients-from-peeringdb
Loading IX-F database... OK

Select the IXP for which the clients list must be built
Enter the text to search for (IXP name, country, city): LINX
   ID  IXP description
   18  GB, London, London Internet Exchange LON1 (LINX LON1)
   777 US, Ashburn, LINX NoVA (LINX NoVA)
   321  GB, London, London Internet Exchange LON2 (LINX LON2)

Enter the ID of the IXP you want to use to build the clients list: 18
```

#### Create `clients.yml` file from Euro-IX member list JSON file

The [Euro-IX member list JSON schema](#) defines a portable output format to export the list of members connected to an Internet Exchange. These files can be used to fetch the list of clients and their attributes (AS-SETs, max-prefix limits) and to use them to automatically build the `clients.yml` file used by ARouteServer to generate route server's configuration.

The `clients-from-euroix` command can be used for this purpose.

```
arouteserver clients-from-euroix --url <URL> <ixp_id> -o <output_file>
```

The JSON file may contain information about more than one IXP for every IX. For example, AMS-IX has 'AMS-IX', 'AMS-IX Caribbean', 'AMS-IX Hong Kong' and more. To filter only those clients which are connected to the IXP of interest an identifier (`ixp_id`) is needed. When executed without the `ixp_id` argument, the command prints the list of IXPs and VLANs reported in the JSON file; the ID can be found on this list:

```
$ arouteserver clients-from-euroix --url https://my.ams-ix.net/api/v1/members.json
IXP ID 1, short name 'AMS-IX'
- VLAN ID 502, name 'GRX', IPv4 prefix 193.105.101.0/25, IPv6 prefix 2001:7f8:86:1::/
↪64
- VLAN ID 504, name 'MDX', IPv4 prefix 195.60.82.128/26
- VLAN ID 600, name 'PI'
- VLAN ID 501, name 'ISP', IPv4 prefix 103.247.139.0/25, IPv6 prefix
↪2001:13c7:6004::/64
IXP ID 3, short name 'AMS-IX Caribbean'
- VLAN ID 600, name 'PI'
```

(continues on next page)



(continued from previous page)

```

- VLAN ID 501, name 'ISP', IPv4 prefix 103.247.139.0/25, IPv6 prefix_
↪2001:13c7:6004::/64
IXP ID 2, short name 'AMS-IX Hong Kong'
- VLAN ID 501, name 'ISP', IPv4 prefix 103.247.139.0/25, IPv6 prefix_
↪2001:13c7:6004::/64
...

```

Finally, the list of clients and their attributes can be fetched:

```

$ arouteserver clients-from-euroix --url https://my.ams-ix.net/api/v1/members.json 1 -
↪-vlan 502
clients:
- asn: 58453
  description: China Mobile International Limited
  ip: 193.105.101.100
- asn: 33849
  description: Comfone AG
  ip: 193.105.101.30
- asn: 8959
  description: Emirates Telecommunications Corporation (Etisalat) (GRX)
  ip: 193.105.101.22
- asn: 8959
  description: Emirates Telecommunications Corporation (Etisalat) (GRX)
  ip: 193.105.101.62
- asn: 12322
  description: Free SAS
  ip: 193.105.101.28
...

```

An example from the LONAP:

```

$ arouteserver clients-from-euroix --url https://portal.lonap.net/apiv1/member-list/
↪list 1
clients:
- asn: 42
  cfg:
    filtering:
      irrdb:
        as_sets:
          - AS-PCH
        max_prefix:
          limit_ipv4: 100
    description: Packet Clearing House AS42
    ip: 5.57.80.238
- asn: 42
  cfg:
    filtering:
      irrdb:
        as_sets:
          - AS-PCH
        max_prefix:
          limit_ipv6: 100
    description: Packet Clearing House AS42
    ip: 2001:7f8:17::2a:1
- asn: 714
  cfg:
    filtering:

```

(continues on next page)

(continued from previous page)

```
irrdb:
  as_sets:
    - AS-APPLE
  max_prefix:
    limit_ipv4: 1000
  description: Apple Europe Ltd
  ip: 5.57.81.57
...
```

To get a list of all the available options, run the `arouteserver clients-from-euroix --help` command.

## Integration with IXP-Manager

Since the popular [IXP-Manager](#) allows to [export the list of members in Euro-IX JSON format](#), the `arouteserver clients-from-euroix` command can also be used to integrate the two tools and to build ARouteServer's list of clients automatically:

```
#!/bin/bash

set -e

# Setup an API key on IXP-Manager and write it below.
# http://docs.ixpmanager.org/features/api/#creating-an-api-key
api_key="YOURAPIKEY"

# Adjust the URL below and point it to your IXP-Manager application.
url="http://www.example.com/api/v4/member-export/ixf/0.6?apikey=$api_key"

# This is the IXP ID you want to export members from.
# It must match with the 'ixp_id' field.
ixp_id=1

# Path of the output clients file that will be built.
clients_file=~/.ars/clients-from-ixpmanager.yml

# Build the clients file using info from IXP-Manager.
arouteserver clients-from-euroix \
  -o $clients_file \
  --url "$url" $ixp_id

# Build the route server configuration.
arouteserver bird \
  --clients $clients_file \
  --ip-ver 4 \
  -o /etc/bird/bird4.new

# Now test the new configuration and, finally,
# push it to the route server.
...
```

### 3.2.6 IX-F Member Export JSON file from `clients.yml`

The `ixf-member-export` command can be used to generate [IX-F Member Export JSON files](#) from the list of clients that are configured on the route server. Although the `clients.yml` file used by ARouteServer to build the

route server configuration contains only those clients that are supposed to connect to the route server itself, it's a quite common practice to preconfigure passive BGP sessions for all the IXP members there. When that's true the clients file contains a comprehensive representation of all the IXP participants.

Please note: the output file generated with this command contains only a subset of the attributes available in the IX-F JSON schema: ASN, IP addresses, max-prefix limits and AS macros. Only information that are hard-coded in the `clients.yml` file are exported: AS-SETS or max prefix limits that during the configuration building process are fetched from PeeringDB or other external data sources are not included in the output file.

```
$ arouteserver ixf-member-export --clients examples/rich/clients.yml "Test IXP"
{
  "version": "0.6",
  "timestamp": "2017-11-24T17:23:41Z",
  "ixp_list": [
    {
      "ixp_id": 0,
      "shortname": "Test IXP",
      "vlan": [
        {
          "id": 0
        }
      ]
    }
  ],
  "member_list": [
    {
      "asnum": 10745,
      "connection_list": [
        {
          "ixp_id": 0,
          "vlan_list": [
            {
              "vlan_id": 0,
              "ipv4": {
                "address": "192.0.2.22"
              }
            }
          ],
        }
      ],
    }
  ],
  [...]
}
```

### 3.2.7 Live tests, development and customization

#### Template context data

To dump the list of variables and data that can be used inside a template, the `template-context` command can be used:

```
arouteserver template-context
```

It produces a YAML document that contains the context variables and their values as they are passed to the template engine used to build configurations.

#### Initialize a custom live test scenario

To setup a new live test scenario:

```
arouteserver init-scenario ~/ars_scenarios/myscenario
```

More details on *How to build custom scenarios*.

## 3.3 Configuration

### 3.3.1 Program configuration

ARouteServer needs the following files to read its own configuration and to determine the policies to be implemented in the route server:

- `arouteserver.yml`: the main ARouteServer configuration file; it contains options and paths to other files (templates, cache directory, external tools...). By default, ARouteServer looks for this file in `~/arouteserver` and `/etc/arouteserver`. This path can be changed using the `--cfg` command line argument. See its default content on [GitHub](#).

The `logging_config_file` parameter here included can be used to *configure logging*.

For details regarding the `rtt_getter_path` option please see `RTT_GETTER`.

- `general.yml`: this is the most important configuration file, where the route server's options and policies are configured. By default, it is located in the same directory of the main configuration file; its path can be set with the `cfg_general` option in `arouteserver.yml`. See its default content on [GitHub](#).

An automatically generated *reStructuredText* version of the file with all its options and comments can be found in the `GENERAL` page.

- `clients.yml`: the list of route server's clients and their options and policies. By default, it is located in the same directory of the main configuration file; its path can be set with the `cfg_clients` option in `arouteserver.yml`. See its default content on [GitHub](#).
- `bogons.yml`: the list of bogon prefixes automatically discarded by the route server. By default, it is located in the same directory of the main configuration file; its path can be set with the `cfg_bogons` option in `arouteserver.yml`. See its default content on [GitHub](#).

The `arouteserver setup` command can be used to setup the environment where ARouteServer is executed and to install the aforementioned files in the proper places.

### 3.3.2 Route server's configuration

Route server's general configuration and policies are outlined in the `general.yml` file.

Configuration details and options can be found within the distributed `general` and `clients` configuration files on [GitHub](#) or in the `GENERAL` page.

Details about some particular topics are reported below.

- *YAML files inclusion and environment variables expansion*
- *Client-level options inheritance*
- *IRRDBs-based filtering*
  - *Use RPKI ROAs as if they were route objects*
  - *Use ARIN Whois database to accept routes*

- *White lists*
- *RPKI*
  - *ROAs sources*
  - *Origin validation*
- *BGP Communities*
  - *Custom BGP Communities*
- *Site-specific custom configuration files*
  - *BIRD hooks*
- *Reject policy and invalid routes tracking*
- *Caveats and limitations*

### YAML files inclusion and environment variables expansion

ARouteServer's YAML configuration files can contain a custom directive (`!include <filepath>`) that can be used to include other files. Moreover, environment variables (`${VAR_NAME}`) are expanded when the configuration files are loaded. This can be useful, for example, when the same configuration is shared by two route servers that differ only in their router ID.

Example with environment variables expansion:

#### general.yml

```
cfg:
  router_id: "${ROUTER_ID}"
  rs_as: 999
  passive: True
  gtsm: True
  filtering:
    [...]
```

Example with file inclusion:

#### general-rs1.yml

```
cfg:
  router_id: "192.0.2.1"
  !include general-shared.yml
```

#### general-rs2.yml

```
cfg:
  router_id: "192.0.2.2"
  !include general-shared.yml
```

#### general-shared.yml

```
#cfg:
# keep the indentation level of the line where
# the !include statement is placed
  rs_as: 999
  passive: True
```

(continues on next page)

(continued from previous page)

```
gtsm: True
filtering:
[...]
```

### Client-level options inheritance

Clients, which are configured in the `clients.yml` file, inherit most of their options from those provided in the `general.yml` file, unless their own configuration sets more specific values.

Options that are inherited by clients and that can be overwritten by their configuration are highlighted in the `general.yml` template file that is distributed with the project.

Example:

#### **general.yml**

```
cfg:
  rs_as: 999
  router_id: "192.0.2.2"
  passive: True
  gtsm: True
```

#### **clients.yml**

```
clients:
- asn: 11
  ip: "192.0.2.11"
- asn: 22
  ip: "192.0.2.22"
  passive: False
- asn: 33
  ip: "192.0.2.33"
  passive: False
  gtsm: False
```

In this scenario, the route server's configuration will look like this:

- a passive session with GTSM enabled toward AS11 client;
- an active session with GTSM enabled toward AS22 client;
- an active session with GTSM disabled toward AS33 client.

### IRRDBs-based filtering

The `filtering.irrdb` section of the configuration files allows to use IRRDBs information to filter or to tag routes entering the route server. Information are acquired using the external program `bgpq3`: installations details on [Installation](#) page.

One or more AS-SETs can be used to gather information about authorized origin ASNs and prefixes that a client can announce to the route server. AS-SETs can be set in the `clients.yml` file on a two levels basis:

- within the `asns` section, one or more AS-SETs can be given for each ASN of the clients configured in the rest of the file;
- for each client, one or more AS-SETs can be configured in the `cfg.filtering.irrdb` section.

To gather information from the IRRDBs, at first the script uses the AS-SETs provided in the client-level configuration; if no AS-SETs are provided there, it looks to the ASN configuration. If no AS-SETs are found in both the client and the ASN configuration, if the `cfg.filtering.irrdb.peering_db` option is set to True the AS-SET from PeeringDB is used (“IRR Record” field). The ASN’s `autnum` object will be used in any case.

Example:

#### clients.yml

```
asns:
  AS22:
    as_sets:
      - "AS-AS22MAIN"
  AS33:
    as_sets:
      - "AS-AS33GLOBAL"
clients:
  - asn: 11
    ip: "192.0.2.11"
    cfg:
      filtering:
        irrdb:
          as_sets:
            - "AS-AS11NETS"
  - asn: 22
    ip: "192.0.2.22"
  - asn: 33
    ip: "192.0.2.33"
    cfg:
      filtering:
        irrdb:
          as_sets:
            - "AS-AS33CUSTOMERS"
  - asn: 44
    ip: "192.0.2.44"
```

With this configuration, the following values will be used to run the `bgpq3` program:

- **AS-AS11NETS** will be used for 192.0.2.11 (it’s configured at client-level for that client);
- **AS-AS22MAIN** for the 192.0.2.22 client (it’s inherited from the `asns`-level configuration of AS22, client’s AS);
- **AS-AS33CUSTOMERS** for the 192.0.2.33 client (the `asns`-level configuration is ignored because a more specific one is given at client-level);
- **AS44** for the 192.0.2.44 client, because no AS-SETs are given at any level. In this case, if the `cfg.filtering.irrdb.peering_db` was set to True, the AS-SET from PeeringDB would be used.

Optionally, the source that must be used to expand the AS macro can be prepended, followed by two colon characters: **RIPE::AS-FOO, RADB::AS64496:AS-FOO**.

### Use RPKI ROAs as if they were route objects

If the `filtering.irrdb.use_rpki_roas_as_route_objects` option is enabled, RPKI ROAs are used as if they were route objects to validate routes whose origin ASN is already authorized by a client’s AS-SET but whose prefix is not. A lookup into the ROA table is made on the basis of the route origin ASN and, if a covering ROA is found, the route is validated. In this case, if the `filtering.irrdb.tag_as_set` general option is True the `prefix_validated_via_rpki_roas` informative community is added to the route.

Please refer to *ROAs sources* in order to configure the source that should be used to gather RPKI ROAs.

### Use ARIN Whois database to accept routes

Similarly to the previous option, `filtering.irrdb.use_arin_bulk_whois_data` allows to support IRR-based filters with additional data. Records from the ARIN Whois database are used to accept those routes whose origin ASN is authorized by the client's AS-SET but whose prefix has not a registered route object. In this case, a lookup into the ARIN Whois database is made on the basis of the origin ASN and if a covering entry is found the route is accepted.

The ARIN Whois database can be obtained by signing an [agreement with ARIN](#). It must be then converted into the appropriate JSON format that ARouteServer expects to find; the `arin-whois-bulk-parser` script can be used for this purpose.

A parsed version of the database dump is offered by NLNOG at the following URL: <http://irrexplorer.nlnog.net/static/dumps/arin-whois-originas.json.bz2>

Further details can be found in [this message](#) appeared on the NANOG mailing list.

### White lists

In addition to prefixes and ASNs gathered as said above, white lists can be configured at client level to manually enter prefixes and origin ASNs that will be treated as if they were included within clients' AS-SET.

If the `filtering.irrdb.tag_as_set` general option is also set to True, routes that fail the basic IRR filters but that are accepted solely because they match a white list entry are tagged with the `prefix_not_present_in_as_set` and `origin_not_present_in_as_set` informational communities.

Example:

```
clients:
- asn: 11
  ip: "192.0.2.11"
  cfg:
    filtering:
      irrdb:
        as_sets:
          - "AS-AS11NETS"
        white_list_route:
          - prefix: "203.0.113.0"
            length: 24
            asn: 65534
```

This configuration allows to authorize routes for 203.0.113.0/24{24-32} with origin ASN 65534 received from the client.

### RPKI

#### ROAs sources

A couple of methods can be used to acquire RPKI data (ROAs):

- (BIRD and OpenBGPD) the builtin method based on [RIPE RPKI Validator cache](#) export file: the URL of a local and trusted instance of RPKI Validator should be provided to ensure that a cryptographically validated dataset is used. By default, the URL of the public instance is used.



- (BIRD only) external tools from the `rtrlib` suite: `rtrlib` and `bird-rtrlib-cli`. One or more trusted local validating caches should be used to get and validate RPKI data before pushing them to BIRD. An overview is provided on the [rtrlib GitHub wiki](#), where also an [usage guide](#) can be found.

The configuration of ROAs source can be done within the `rpki_roas` section of the `general.yml` file.

## Origin validation

RPKI-based validation of routes can be configured using the `general.filtering.rpki_bgp_origin_validation` section. RFC8097 BGP extended communities are used to mark routes on the basis of their validity state. Depending on the `reject_invalid` configuration, INVALID routes can be rejected before entering the route server or accepted for further processing by external tools or functions provided within `.local.files`. INVALID routes are not propagated to clients.

## BGP Communities

BGP communities can be used for many features in the configurations built using ARouteServer: blackhole filtering, AS\_PATH prepending, announcement control, various informative purposes (valid origin ASN, valid prefix, ...) and more. All these communities are referenced by *name* (or *tag*) in the configuration files and their real values are reported only once, in the `communities` section of the `general.yml` file. For each community, values can be set for any of the three *formats*: standard (RFC1997), extended (RFC4360/RFC5668) and large (RFC8092).

## Custom BGP Communities

Custom, locally significant BGP communities can also be used for informational purposes, for example to keep track of the geographical origin of a route or the nature of the relation with the announcing route server client.

Custom communities are declared once in the `general.yml` configuration file and then are referenced by clients definitions in the `clients.yml` file.

Example:

### `general.yml`

```
cfg:
  rs_as: 6777
  router_id: "80.249.208.255"
custom_communities:
  colo_digitalrealty_ams01:
    std: "65501:1"
    lrg: "6777:65501:1"
  colo_equinix_am3:
    std: "65501:2"
    lrg: "6777:65501:2"
  colo_evoswitch:
    std: "65501:3"
    lrg: "6777:65501:3"
  member_type_peering:
    std: "65502:1"
    lrg: "6777:65502:1"
  member_type_probono:
    std: "65502:2"
    lrg: "6777:65502:2"
```

### `clients.yml`

```
clients:
- asn: 112
  ip: "192.0.2.112"
  cfg:
    attach_custom_communities:
      - "colo_digitalrealty_ams01"
      - "member_type_probono"
- asn: 22
  ip: "192.0.2.22"
  passive: False
  cfg:
    attach_custom_communities:
      - "colo_equinix_am3"
      - "member_type_peering"
- asn: 33
  ip: "192.0.2.33"
  cfg:
    attach_custom_communities:
      - "colo_evoswitch"
      - "member_type_peering"
```

### Site-specific custom configuration files

Local configuration files can be used to load static site-specific snippets of configuration into the BGP daemon, bypassing the dynamic ARouteServer configuration building mechanisms. These files can be used to configure, for example, neighborhood with peers which are not route server members or that require custom settings.

Local files inclusion can be enabled by a command line argument, `--use-local-files`: there are some fixed points in the configuration files generated by ARouteServer where local files can be included:

- BIRD:

```
BIRDConfigBuilder.LOCAL_FILES_IDS = ['header', 'header4', 'header6', 'footer', 'footer4', 'footer6']
```

- OpenBGPD:

```
OpenBGPDConfigBuilder.LOCAL_FILES_IDS = ['header', 'pre-irrd', 'post-irrd', 'pre-client', 'post-client']
```

One or more of these labels must be used as the argument's value in order to enable the relative inclusion points. For each enabled label, an *include* statement is added to the generated configuration in the point identified by the label itself. To modify the base directory, the `--local-files-dir` command line option can be used.

These files must be present on the host running the route server.

- Example, BIRD, file name "footer4.local" in "/etc/bird" directory:

```
protocol bgp RouteCollector {
  local as 999;
  neighbor 192.0.2.99 as 65535;
  rs client;
  secondary;

  import none;
  export all;
}
```

- Example, OpenBGPD, header and post-clients:

```

$ arouteserver openbgpd --use-local-files header post-clients
include "/etc/bgpd/header.local"

AS 999
router-id 192.0.2.2

[...]

group "clients" {
    neighbor 192.0.2.11 {
        [...]
    }
}

include "/etc/bgpd/post-clients.local"

[...]

```

In the example above, the `header` and `post-clients` inclusion points are enabled and allow to insert two `include` statements into the generated configuration: one at the start of the file and one between clients declaration and filters.

- Example, OpenBGPD, client and footer:

```

$ arouteserver openbgpd --use-local-files client footer --local-files-dir /etc/
AS 999
router-id 192.0.2.2

[...]

group "clients" {
    neighbor 192.0.2.11 {
        include "/etc/client.local"
        [...]
    }
    neighbor 192.0.2.22 {
        include "/etc/client.local"
        [...]
    }
}

[...]

include "/etc/footer.local"

```

The example above uses the `client` label, that is used to add an `include` statement into every neighbor configuration. Also, the base directory is set to `/etc/`.

## BIRD hooks

In BIRD, hook functions can also be used to tweak the configuration generated by ARouteServer. Hooks are enabled by the `--use-hooks` command line argument, that accepts one or more of the following hook IDs:

```
BIRDConfigBuilder.HOOKS = ['pre_receive_from_client', 'post_receive_from_client', 'pre
```

Functions with name `hook_<HOOK_ID>` must then be implemented within *.local* configuration files, in turn included using the `--use-local-files` command line argument.

Example:

```
$ arouteserver bird --ip-ver 4 --use-local-files header --use-hooks pre_
↪receive_from_client
router id 192.0.2.2;
define rs_as = 999;

log "/var/log/bird.log" all;
log syslog all;
debug protocols all;

protocol device {};

table master sorted;

include "/etc/bird/header.local";

[...]

filter receive_from_AS3333_1 {
    if !(source = RTS_BGP ) then
        reject "source != RTS_BGP - REJECTING ", net;

    if !hook_pre_receive_from_client(3333, 192.0.2.11, "AS3333_1") then
        reject "hook_pre_receive_from_client returned false -
↪REJECTING ", net;

    scrub_communities_in();

[...]
```

Details about hook functions can be found in the `BIRD_HOOKS` page.

An example (including functions' prototypes) is provided within the “examples/bird\_hooks” directory (also on [GitHub](#)).

### Reject policy and invalid routes tracking

Invalid routes, that is those routes that failed the validation process, can be simply discarded as they enter the route server (default behaviour) or, optionally, they can be kept for troubleshooting purposes, analysis or statistic reporting.

The `reject_policy` configuration option can be set to `tag` in order to have invalid routes tagged with a user-configurable BGP Community (`reject_reason`) whose purpose is to keep track of the reason for which they are considered to be invalid. These routes are also set with a low local-pref value (1) and tagged with a control BGP Community that prevents them from being exported to clients. If configured, the `rejected_route_announced_by_community` is used to track the ASN of the client that announced the invalid route to the route server.

The goal of this feature is to allow the deployment of route collectors that can be used to further process invalid routes announced by clients. These route collectors can be configured using *site-specific .local files*. The `InvalidRoutesReporter` is an example of this kind of route collector.

The reason that brought the server to reject the route is identified using a numeric value in the last part of the BGP Community; the list of reject reasons follow:

ID	Reason
0	Special meaning: the route must be treated as rejected. *
1	Invalid AS_PATH length
2	Prefix is bogon
3	Prefix is in global blacklist
4	Invalid AFI
5	Invalid NEXT_HOP
6	Invalid left-most ASN
7	Invalid ASN in AS_PATH
8	Transit-free ASN in AS_PATH
9	Origin ASN not in IRRDB AS-SETs
10	IPv6 prefix not in global unicast space
11	Prefix is in client blacklist
12	Prefix not in IRRDB AS-SETs
13	Invalid prefix length
14	RPKI INVALID route
65535	Unknown

\* This is not really a reject reason code, it only means that the route must be treated as rejected and must not be propagated to clients.

### Caveats and limitations

Not all features offered by ARouteServer are supported by both BIRD and OpenBGPD. The following list of limitations is based on the currently supported versions of BIRD (1.6.3 and 1.6.4) and OpenBGPD (OpenBSD 6.0 up to 6.3).

- OpenBGPD
  - Currently, **path hiding** mitigation is not implemented for OpenBGPD configurations. Only single-RIB configurations are generated.
  - **ADD-PATH** is not supported by OpenBGPD.
  - For max-prefix filtering, only the `shutdown` and the `restart` actions are supported by OpenBGPD. Restart is configured with a 15 minutes timer.
  - [An issue](#) is preventing next-hop rewriting for **IPv6 blackhole filtering** policies on OpenBGPD/OpenBSD 6.0.
  - **Large communities** are not supported by OpenBGPD 6.0: features that are configured to be offered via large communities only are ignored and not included into the generated OpenBGPD configuration.
  - OpenBGPD does not offer a way to delete **extended communities** using wildcard (`rt xxx:*`): peer-ASN-specific extended communities (such as `prepend_once_to_peer`, `do_not_announce_to_peer`) are not scrubbed from routes that leave OpenBGPD route servers and so they are propagated to the route server clients.
  - **Graceful shutdown** is supported only on OpenBGPD 6.2 or later.
  - The Site of Origin Extended BGP communities in the range `65535:*` are reserved for internal reasons.

Depending on the features that are enabled in the `general.yml` and `clients.yml` files, compatibility issues may arise; in this case, ARouteServer logs one or more errors, which can be then acknowledged and ignored using the `--ignore-issues` command line option:

```

$ arouteserver openbgpd
ARouteServer 2017-03-23 21:39:45,955 ERROR Compatibility issue ID 'path_hiding'. The
↳'path_hiding'
general configuration parameter is set to True, but the configuration generated by
↳ARouteServer for
OpenBGPD does not support path-hiding mitigation techniques.
ARouteServer 2017-03-23 21:39:45,955 ERROR One or more compatibility issues have been
↳found.

Please check the errors reported above for more details.
To ignore those errors, use the '--ignore-issues' command line argument and list the
↳IDs of the
issues you want to ignore.
$ arouteserver openbgpd --ignore-issues path_hiding
AS 999
router-id 192.0.2.2

fib-update no
log updates
...

```

## 3.4 Logging

ARouteServer's logging is based on the Python logging facility and can be configured using the `fileConfig()` format.

The `logging_config_file` parameter which can be found in the main program configuration file (`arouteserver.yml`) must be set with the path of the INI file that contains the logging configuration statements. By default, it is set to point to the `log.ini` file that ships with the program and that contains a basic configuration which simply prints log messages to `stderr`.

The `log.ini` shipped with the program contains some commented sections that can be used as guidance for setting up other logging methods: files, syslog (both local and remote via UDP), email, Slack.

### 3.4.1 Logging levels

- INFO: quite verbose, informational messages are logged using this level to inform the user about the progress of the building process.
- WARN: minor issues which do not prevent the configuration file from being built but that should be analyzed by a route server operator are logged using this level. This includes, for example, peers with missing AS-SETs when IRR-based filters are enabled, or empty AS-SETs, or new release notices.
- ERROR: this level is used to log messages related to issues that prevent the configuration file from being built.

For example, `>=INFO` messages could be logged to `stderr`, `>=WARN` messages via a `buffered email` and `>=ERROR` via email and Slack (using a third-party component like `slack-log-handler`).

Output of an example execution is reported below:

```

WARNING The 'filtering.global_black_list_pref' option is missing or empty. It is
↳strongly suggested to provide at least the list of local IPv4/IPv6 networks here.
WARNING The 'filtering.rrdb.tag_as_set' option is set but no BGP communities are
↳provided to tag prefixes.
INFO Started processing configuration for /home/pierky/arouteserver/templates/bird/
↳main.j2

```

(continues on next page)

(continued from previous page)

```
INFO Enricher 'IRRdb origin ASNs' started
WARNING No AS-SETs provided for the 'AS3333_1' client. Only AS3333 will be expanded.
WARNING No AS-SETs provided for the 'AS10745_1' client. Only AS10745 will be expanded.
INFO Enricher 'IRRdb origin ASNs' completed successfully after 0 seconds
INFO Enricher 'IRRdb prefixes' started
INFO Enricher 'IRRdb prefixes' completed successfully after 0 seconds
INFO Enricher 'PeeringDB max-prefix' started
INFO Enricher 'PeeringDB max-prefix' completed successfully after 0 seconds
INFO Configuration processing completed after 0 seconds.
INFO Started template rendering for /home/pierky/arouteserver/templates/bird/main.j2
INFO Template rendering completed after 1 seconds.
```

## 3.5 Examples of configurations

### 3.5.1 Default

Configurations built using the default `general.yml` and `clients.yml` files distributed with the project.

<https://github.com/pierky/arouteserver/blob/master/examples/default>

See the textual representation of this configuration.

### 3.5.2 Feature-rich example

Configurations built using the files provided in the `examples/rich` directory.

- GTSM and ADD-PATH are enabled by default on the route server.
- Next-hop filtering allows clients to set NEXT\_HOP of any client in the same AS.
- Local networks are filtered, and also transit-free ASNs, invalid paths and prefixes/origin ASNs which are not authorized by clients' AS-SETs.
- Dataset used for prefix validation extended using ARIN Whois DB dump and RPKI ROAs.
- RPKI-based Origin Validation is enabled; INVALID routes are rejected.
- A max-prefix limit is enforced on the basis of PeeringDB information.
- Blackhole filtering is implemented with a rewrite-next-hop policy and can be triggered with BGP communities BLACKHOLE, 65534:0 and 999:666:0.
- Control communities allow selective announcement control and prepending, also on the basis of peers RTT.
- Graceful BGP session shutdown is enabled.
- Client timers are configured using the custom, site-specific `.local` file.
- Informational custom BGP communities are used to tag routes from European or American clients.

Please note: for the sake of readability of the configuration files built in this example the set of RPKI ROAs is artificially limited to just a bunch of them.

<https://github.com/pierky/arouteserver/blob/master/examples/rich>

See the textual representation of this configuration.

### 3.5.3 BIRD hooks example

The BIRD configurations provided in this example have been generated enabling **BIRD hooks**:

```
$ arouteserver bird --ip-ver 4 --use-local-files header --use-hooks pre_receive_from_
↳client post_receive_from_client [...]
```

The above list of hooks passed to the `bird` command has been truncated for the sake of readability; the complete list used in this example is provided below.

The command line argument `--use-local-files` enables the header inclusion point, in order to add the `include "/etc/bird/header.local";` configuration statement to the BIRD configuration generated by ARouteServer.

```
define rs_as = 999;

log "/var/log/bird.log" all;
log syslog all;
debug protocols all;

protocol device {};

table master sorted;

include "/etc/bird/header.local";
...
```

This file must be present on the route server where BIRD is executed and must contain the custom functions used to implement the hooks. See the `header.local` file for the functions declaration.

List of hooks used in this example:

- `pre_receive_from_client`
- `post_receive_from_client`
- `pre_announce_to_client`
- `post_announce_to_client`
- `scrub_communities_in`
- `scrub_communities_out`
- `apply_blackhole_filtering_policy`
- `route_can_be_announced_to`
- `announce_rpki_invalid_to_client`

[https://github.com/pierky/arouteserver/blob/master/examples/bird\\_hooks](https://github.com/pierky/arouteserver/blob/master/examples/bird_hooks)

### 3.5.4 Clients from Euro-IX member list JSON file

Some clients files automatically built from Euro-IX member list JSON files are reported here.

<https://github.com/pierky/arouteserver/blob/master/examples/clients-from-euroix>



### 3.5.5 configure command output

The `configure` command can be used to quickly generate policy definition files (*general.yml*) which are based on suggested settings and best practices.

A list of BGP communities is also automatically built.

```

$ arouteserver configure --output examples/auto-config/README.rst

BGP daemon
=====

Depending on the BGP daemon used for the route server some features may not be
available.

Details here:
https://arouteserver.readthedocs.io/en/latest/CONFIG.html#caveats-and-
limitations

Which BGP daemon will be used? [bird/openbgpd] bird
Router server's ASN
=====

What's the ASN of the route server? 64496
Route server's BGP router-id
=====

Please enter the route server BGP router-id: 192.0.2.1
List of local networks
=====

A list of local IPv4/IPv6 networks must be provided here: routes announced by
route server clients for these prefixes will be filtered out.

Please enter a comma-separated list of local networks: 192.0.2.0/24,2001:db8::/32

Route server policy definition file generated successfully!
=====

The content of the general configuration file will now be written to examples
/auto-config/bird-general.yml

Some notes:

- Accepted prefix lengths are 8-24 for IPv4 and 12-48 for IPv6.
- Routes with 'transit-free networks' ASNs in the middle of AS_PATH are
rejected.
- IRR-based filters are enabled; prefixes that are more specific of those
registered are accepted.
- PeeringDB is used to fetch AS-SETS for those clients that are not explicitly
configured.
- RPKI ROAs are used as if they were route objects to further enrich IRR data.
- ARIN Whois database dump is fetched from NLNOG to further enrich IRR data.
- PeeringDB is used to fetch networks prefix count.
- Routes tagged with the GRACEFUL_SHUTDOWN well-known community (65535:0) are
processed accordingly to draft-ietf-grow-bgp-gshut.

```

The textual description (HTML) generated on the basis of the *general.yml* files produced by this command is also

reported here.

<https://github.com/pierky/arouteserver/blob/master/examples/auto-config>

[bird-general.yml.html](#) - See the textual representation of this configuration.

[openbgpd62-general.yml.html](#) - See the textual representation of this configuration.

### 3.5.6 IX-F Member Export files

The files reported within this directory were generated using the `ixf-member-export` command.

<https://github.com/pierky/arouteserver/blob/master/examples/ixf-member-export>

## 3.6 Tools

### 3.6.1 Invalid routes reporter

This script is intended to be used as an ExaBGP process to elaborate and report/log invalid routes received by route servers that have been previously configured using the “tag” reject policy option of ARouteServer.

For more information: <https://invalidroutesreporter.readthedocs.io>

## 3.7 Live tests

Live tests are used to validate configurations built by ARouteServer and to test compliance between expected and real results.

A mix of Python unittest and Docker (and KVM too for OpenBGPD tests) allows to create scenarios where some instances of BGP speakers (the clients) connect to a route server whose configuration has been generated using this tool.

Some built-in tests are included within the project and have been used during the development of the tool; new *custom scenarios* can be easily built by users and IXP managers to test their own policies.

Example: in a configuration where blackhole filtering is enabled, an instance of a route server client (AS1) is used to announce some tagged prefixes (203.0.113.1/32) and the instances representing other clients (AS2, AS3) are queried to ensure they receive those prefixes with the expected blackhole NEXT\_HOP (192.0.2.66).

```
def test_071_blackholed_prefixes_as_seen_by_enabled_clients(self):
    for inst in (self.AS2, self.AS3):
        self.receive_route(inst, "203.0.113.1/32", self.rs,
                           next_hop="192.0.2.66",
                           std_comms=["65535:666"], lrg_comms=[])
```

Travis CI log file contains the latest built-in live tests results. Since (AFAIK) OpenBGPD can't be run on Travis CI platform, the full live tests results including those run on OpenBGPD can be found on [this file](#).

### 3.7.1 Setting up the environment to run live tests

1. To run live tests, Docker must be present on the system. Some info about its installation can be found on the [External programs](#) installation section.

- In order to have instances of the route server and its clients to connect each other, a common network must be used. Live tests are expected to be run on a Docker bridge network with name `arouteserver` and subnet `192.0.2.0/24`:`2001:db8:1:1::/64`. The following command can be used to create this network:

```
docker network create --ipv6 --subnet=192.0.2.0/24 --subnet=2001:db8:1:1::/
↪64 arouteserver
```

- Route server client instances used in live tests are based on BIRD 1.6.4, as well as the BIRD-based version of the route server used in built-in live tests; the `pierky/bird:1.6.4` image is expected to be found on the local Docker repository. Build the Docker image (or pull it from [Dockerhub](#)):

```
# build the image using the Dockerfile
# from https://github.com/pierky/dockerfiles
mkdir ~/dockerfiles
cd ~/dockerfiles
curl -o Dockerfile.bird -L https://raw.githubusercontent.com/pierky/dockerfiles/
↪master/bird/1.6.4/Dockerfile
docker build -t pierky/bird:1.6.4 -f Dockerfile.bird .

# or pull it from Dockerhub
docker pull pierky/bird:1.6.4
```

If there is no plan to run tests on the OpenBGPD-based version of the route server, no further settings are needed. To run tests on the OpenBGPD-based version too, the following steps must be done as well.

### OpenBGPD live-tests environment

- To run an instance of OpenBGPD, KVM is needed. Some info about its installation can be found on the [External programs](#) installation section.
- Setup and install a KVM virtual-machine running one of the supported versions of OpenBSD. This VM will be started and stopped many times during tests: don't use a production VM.
  - By default, the VM name must be `arouteserver_openbgpd60` or `arouteserver_openbgpd61` or `arouteserver_openbgpd62`; this can be changed by setting the `VIRSH_DOMAINNAME` environment variable before running the tests.
  - The VM must be connected to the same Docker network created above: the commands `ip link show` and `ifconfig` can be used to determine the local network name needed when creating the VM:

```
$ ifconfig
br-2d2956ce4b64 Link encap:Ethernet HWaddr 02:42:57:82:bc:91
inet addr:192.0.2.1 Bcast:0.0.0.0 Mask:255.255.255.0
inet6 addr: fe80::42:57ff:fe82:bc91/64 Scope:Link
inet6 addr: 2001:db8:1:1::1/64 Scope:Global
inet6 addr: fe80::1/64 Scope:Link
UP BROADCAST MULTICAST MTU:1500 Metric:1
...
```

- In order to run built-in live test scenarios, the VM must be reachable at `192.0.2.2/24` and `2001:db8:1:1::2/64`.

On the following example, the virtual disk will be stored in `~/vms`, the VM will be reachable by connecting to any IP address of the host via VNC, the installation disk image is expected to be found in the `install60.iso` file and the network name used is `br-2d2956ce4b64`:

```
sudo virsh pool-define-as --name vms_pool --type dir --target ~/vms
sudo virsh pool-start vms_pool
sudo virt-install \
  -n arouteserver_openbgpd60 \
  -r 512 \
  --vcpus=1 \
  --os-variant=openbsd4 \
  --accelerate \
  -v -c install60.iso \
  -w bridge:br-2d2956ce4b64 \
  --graphics vnc,listen=0.0.0.0 \
  --disk path=~/.vms/arouteserver_openbgpd.qcow2,size=5,format=qcow2
```

Finally, add the current user to the libvirtd group to allow management of the VM:

```
sudo adduser `id -un` libvirtd
```

3. To interact with this VM, the live tests framework will use SSH; by default, the connection will be established using the root username and the local key file `~/.ssh/arouteserver`, so the VM must be configured to accept SSH connections using SSH keys:

```
mkdir /root/.ssh
cat << EOF > .ssh/authorized_keys
ssh-rsa [public_key_here] arouteserver
EOF
```

The `StrictHostKeyChecking` option is disabled via command line argument in order to allow to connect to multiple different VMs with the same IP address.

The SSH username and key file path can be changed by setting the `SSH_USERNAME` and `SSH_KEY_PATH` environment variables before running the tests.

Be sure that the `bgpd` daemon will startup automatically at boot and that the `bgpctl` tool can be executed correctly on the OpenBSD VM:

```
echo "bgpd_flags=" >> /etc/rc.conf.local
chmod 0555 /var/www/bin/bgpctl
```

### 3.7.2 How to run built-in live tests

To run built-in live tests, the full repository must be cloned locally and the environment must be configured as reported above.

To test both the BIRD- and OpenBGPD-based route servers, run the Python unittest using `nose`:

```
# from within the repository's root
nosetests -vs tests/live_tests/
```

### 3.7.3 How it works

Each directory in `tests/live_tests/scenarios` represents a scenario: the route server configuration is stored in the usual `general.yml` and `clients.yml` files, while other BGP speaker instances (route server clients and their peers) are configured through the `ASxxx.j2` files. These files are Jinja2 templates and are expanded by the Python code at runtime. Containers' configuration files are saved in the local `var` directory and are used to mount the BGP speaker configuration file (currently, `/etc/bird/bird.conf` for BIRD and `/etc/bgpd.conf` for

OpenBGPD). The unittest code sets up a Docker network (with name `arouteserver`) used to attach instances and finally brings instances up. Regular Python unittest tests are then performed and can be used to match expectations to real results.

Details about the code behind the live tests can be found in the `LIVETESTS_CODEDOC` section.

### 3.7.4 Built-in scenarios

Some notes about the built-in scenarios that are provided with the program follow.

#### BGP communities

Communities:

fmt	do_not_announce_to_peer	announce_to_peer	do_not_announce_to_any
std	0:peer_as	999:peer_as	0:999
ext	rt:0:peer_as	rt:999:peer_as	rt:0:999
lrg	999:0:peer_as	999:999:peer_as	999:0:999

AS2

- announced prefixes:

Prefix ID	Comms	Prefix	Expected result
AS2_only_to_AS1	0:999, 999:1	2.0.1.0/24	only AS1 receives the prefix
AS2_only_to_AS1	rt:0:999, rt:999:1	2.0.2.0/24	only AS1 receives the prefix
AS2_only_to_AS1	1999:0:999, 999:999:1	2.0.3.0/24	only AS1 receives the prefix
AS2_only_to_AS131073	0:999, rt:999:131073	2.0.4.0/24	only AS131073 receives the prefix
AS2_only_to_AS131073	1999:0:999, 999:999:131073	2.0.5.0/24	only AS131073 receives the prefix
AS2_bad_cust_comm	65501:65501	2.0.6.0/24	AS1 and AS131073 receive the prefix without the cust community (scrubbed by the rs)

AS1

- configured to have its routes tagged with `cust_comm1` (65501:65501, 999:65501:65501, rt:65501:65501)
- announced prefixes:

Prefix ID	Comms	Prefix	Expected result
AS1_good1		1.0.1.0/24	AS2 and AS131073 receive the prefix, tagged with the custom community

AS131073

#### Default configuration

A simple scenario to verify that the default `general.yml` and `clients.yml` files distributed with the program lead to a working configuration.

The files used here are links to those provided within the `config.d` directory.

## Global scenario

Built to group as many tests as possible in a single scenario.

- **AS1:**

AS-SETs:

- AS-AS1 (AS1, 1.0.0.0/8, 128.0.0.0/7)
- AS-AS1\_CUSTOMERS (AS101, AS103, 101.0.0.0/16, 103.0.0.0/16, AS104)
- white list: 11.1.0.0/16, ASN 1011
- white list routes: exact 11.3.0.0/16 AS1011, 11.4.0.0/16 or more spec w/o origin AS

Enabled to perform graceful BGP session shutdown.

clients:

- AS1\_1 (192.0.2.11, RTT 0.1 ms)
  - \* next-hop-self configured in AS1\_1.conf
  - \* next\_hop.policy: strict (inherited from general config)

Originated prefixes:

Prefix ID	Prefix	AS_PATH	Expected result
AS1_good1	1.0.1.0/24		pass
AS1_good2	1.0.2.0/24		pass
bogon1	10.0.0.0/24		fail prefix_is_bogon
local1	192.0.2.0/24		fail prefix_is_in_global_blacklist
pref_len1	128.0.0.0/7		fail prefix_len_is_valid
peer_as1	128.0.0.0/8	[2, 1]	fail bgp_path.first != peer_as
invalid_asn1	128.0.0.0/9	[1, 65536 1]	fail as_path_contains_invalid_asn
aspath_len1	128.0.0.0/10	[1, 2x6]	fail bgp_path.len > 6
AS1_whitel_1	11.1.1.0/24	[1, 1011]	accepted, cause in white list
AS1_whitel_2	11.1.2.0/24	[1, 1000]	rejected, bad ASN even if prefix in white list
AS1_whitel_3	11.2.1.0/24	[1, 1011]	rejected, bad prefix even if ASN in white list
AS1_whitel_4	11.3.0.0/16	[1, 1011]	accepted because in white_list_route
AS1_whitel_5	11.4.1.0/24	[1, 1000]	accepted because in white_list_route
AS1_whitel_6	11.3.1.0/24	[1, 1011]	rejected, more specific of prefix allowed by white list route

- AS1\_2 (192.0.2.12, RTT 5 ms)
  - \* NO next-hop-self in AS1\_2.conf (next-hop of AS101 used for AS101\_good == 101.0.1.0/24)
  - \* next\_hop.policy: same-as (from clients config)
  - \* not enabled to receive blackhole requests

Originated prefixes:

Prefix ID	Prefix	Feature	Expected result
AS1_good1	1.0.1.0/24		
AS1_good2	1.0.2.0/24		
AS1_good3	1.0.3.0/24	next_hop=AS1_1	win next_hop_is_valid_for_AS1_2 (same-as)

- **AS2:**

AS-SETs:

- AS-AS2 (AS2, 2.0.0.0/16)
- AS-AS2\_CUSTOMERS (AS101, AS103, 101.0.0.0/16, 103.0.0.0/16)

Not enabled to perform graceful BGP session shutdown.

clients:

- AS2 (192.0.2.21, RTT 17.3 ms)
  - \* next-hop-self configured in AS2.conf
  - \* next\_hop.policy: authorized\_addresses (from clients config)
  - \* next\_hop.authorized\_addresses\_list: - 192.0.2.21 and 2001:db8:1:1::21, its own IP addresses - 192.0.2.22 and 2001:db8:1:1::22, IP addresses not configured as route server client

Originated prefixes:

Prefix ID	Pre-fix	Feature	Expected result
AS2_good1	2.0.1.0/24		
AS2_good2	2.0.2.0/24		
AS2_blackhole1	2.0.3.1/24	announced with BLACKHOLE 65535:666 comm	propagated with only 65535:666 to AS1_1 and AS3 (AS1_2 has “announce_to_client” = False) and next-hop 192.0.2.66; NO_EXPORT also added
AS2_blackhole2	2.0.3.2/24	announced with local 65534:0 comm	as above
AS2_blackhole3	2.0.3.3/24	announced with local 65534:0:0 comm	as above
AS2_nonclient1	2.0.4.0/24	announce with an authorized next-hop	received by other clients
AS2_nonclient2	2.0.5.0/24	announce with an unknown next-hop	not received by other clients

- **AS3:**

AS-SETs: none

clients:

- AS3 (192.0.2.31, RTT 123.8)
  - \* no enforcing of origin in AS-SET
  - \* no enforcing of prefix in AS-SET
  - \* ADD-PATH enabled
  - \* passive client-side (no passive on the route server)

Originated prefixes:

Prefix ID	Prefix	Communities	Expected result
AS3_blacklist1	3.0.1.0/24		fail prefix_is_in_AS3_1_blacklist
AS3_cc_AS1only	3.0.2.0/24	0:999, 65501:1	seen on AS1_1/2 only
AS3_cc_not_AS1	3.0.3.0/24	0:1	seen on AS2 only
AS3_cc_none	3.0.4.0/24	0:999	not seen
AS3_prepend1any	3.0.5.0/24	65521:65521	AS_PATH 3, 3
AS3_prepend2any	3.0.6.0/24	65522:65522	AS_PATH 3, 3, 3
AS3_prepend3any	3.0.7.0/24	65523:65523	AS_PATH 3, 3, 3, 3
AS3_prepend1_AS1	3.0.8.0/24	65521:1	AS_PATH 3, 3 on AS1 clients
AS3_prepend2_AS2	3.0.9.0/24	65522:2	AS_PATH 3, 3, 3 on AS2 clients
AS3_prep3AS1_1any	3.0.10.0/24	65523:1 65521:65521	AS_PATH 3, 3, 3, 3 on AS1 clients, 3, 3 on AS2 clients
AS3_noexport_any	3.0.11.0/24	65507:999	received by all with NO_EXPORT
AS3_noexport_AS1	3.0.12.0/24	65509:1 65523:2	(prepend x3 to AS2) received by AS1 with NO_EXPORT
AS3_rfc1997_noexport	3.0.13.0/24	NO_EXPORT	received by all with NO_EXPORT
AS3_transitfree_2	3.0.14.0/24		AS_PATH 3, 174, 33: rejected even if 3 is in the transit-free ASN list
Default_route	0.0.0.0/0		rejected by rs

• **AS4:**

AS-SETs: none

clients:

– AS4 (192.0.2.41, RTT 600)

- \* no enforcing of origin in AS-SET
- \* no enforcing of prefix in AS-SET
- \* RTT thresholds configured on rs: 5, 10, 15, 20, 30, 50, 100, 200, 500
- \* other peers RTTs: - AS1\_1: 0.1 - AS1\_2: 5 - AS2: 17.3 - AS3: 123.8

Originated prefixes:



Pre-fix ID	Pre-fix	Communi-ties	Goal	Who receives it
AS4_rtt410.1.0/24	1999	64532:15	Do not announce to any + announce to peers with RTT <= 15 ms	AS1_1, AS1_2
AS4_rtt420.2.0/24	1999	64532:5	Do not announce to any + announce to peers with RTT <= 5 ms	AS1_1, AS1_2
AS4_rtt430.3.0/24	531:15	64531:15	Do not announce to peers with RTT > 15 ms	AS1_1, AS1_2
AS4_rtt440.4.0/24	531:5	64531:5	Do not announce to peers with RTT > 5 ms	AS1_1, AS1_2
AS4_rtt450.5.0/24	531:5	65501:3	Do not announce to peers with RTT > 5 ms but announce to AS3	AS1_1, AS1_2, AS3
AS4_rtt460.6.0/24	530:5	64531:100	Do not announce to peers with RTT <= 5 and Do not announce to peers with RTT > 100	AS2
AS4_rtt470.7.1/32	535:666	64531:20	BLACKHOLE request, do not announce to peers with RTT > 20	AS1_1, AS2 (AS1_2 not enabled to receive black-hole requests)
AS4_rtt480.8.0/24	539:100	64538:10	Prepend 3x to > 100 ms, 2x to > 10 ms	AS1_1, AS1_2, AS2 2x, AS3 3x
AS4_rtt490.9.0/24	536:5	64535:20 65521:65521	Prepend 3x to <= 5 ms, 2x to <= 20, 1x to any	AS1_1 & AS1_2 3x, AS2 2x, AS3 1x
AS4_rtt410.10.0/24	537:10	rt:64538:20	Prepend 1x to > 10 ms, 2x to > 20 ms	AS1_1 & AS1_2 no prep, AS2 1x, AS3 2x

- **AS101:**

clients:

- Not a route server client, it only peers with AS1\_1, AS1\_2 and AS2 on 192.0.2.101.
- RPKI ROAs:

ID	Prefix	Max	ASN
1	101.0.8.0/24		101
2	101.0.9.0/24		102
3	101.0.128.0/20	23	101
4	101.2.0.0/17		101
5	101.2.128.0/17	24	101
6	101.3.0.0/16	24	105

Originated prefixes:

Prefix ID	Prefix	AS_PATH	Expected result
AS101_good1	101.0.1.0/24		fail next_hop_is_valid_for_AS1_2 (for the prefix announced by AS101 to AS1_2)
AS101_no_rset	101.1.0.0/24		fail prefix_is_in_AS1_1_r_set and prefix_is_in_AS2_1_r_set
AS102_no_asset	102.0.1.0/24	[101 102]	fail origin_as_in_AS1_1_as_set and origin_as_in_AS2_1_as_set
AS101_bad_std	101.0.2.0/24		add 65530:0, scrubbed by rs
AS101_bad_lrg	101.0.3.0/24		add 999:65530:0, scrubbed by rs
AS101_other_s	101.0.4.0/24		add 888:0, NOT scrubbed by rs
AS101_other_l	101.0.5.0/24		add 888:0:0, NOT scrubbed by rs
AS101_bad_good	101.0.6.0/24		add 65530:1,999:65530:1,777:0,777:0:0, 65530 are scrubbed by rs, 777:** are kept
AS101_transitfree	101.0.7.0/24	[101 174]	fail as_path_contains_transit_free_asn
AS101_roa_valid	101.0.8.0/24		roa check ok (roa n. 1), tagged with 64512:1 / 999:64512:1
AS101_roa_invalid	101.0.9.0/24		roa check fail (roa n. 2, bad origin ASN), rejected
AS101_roa_badlen	101.0.128.0/24		roa check fail (roa n. 3, bad length), rejected
AS101_roa_blackhole	101.0.128.1/32		65535:666, pass because blackhole filtering request
AS101_roa_routeobj	101.2.0.0/17		accepted because roa_as_route_objects, add 65530:2
AS101_roa_routeobj	101.2.1.0/24		fail, roa_as_route_objects but prefix is more specific than ROA
AS101_roa_routeobj	101.3.128.0/24		accepted because roa_as_route_objects, add 65530:2
AS101_roa_routeobj	101.4.0.0/24	[101 105]	fail, roa_as_route_objects but origin ASN not allowed by AS-SETs
AS101_no_ipv6_global	2001::/32		fail IPv6 global unicast space check
AS103_gshut_1	103.0.1.0/24	to AS1: [101 103] to AS2: [101*2 103]	AS1 (best) performs gshut of this route; AS3 and AS4 receive the route via AS2 (sub-optimal path)
AS103_gshut_2	103.0.2.0/24	to AS1: [101*2 103] to AS2: [101 103]	AS2 (best) tries gshut of this route but it's not enabled; AS3 and AS4 receive the route via AS2
AS104_arin_1	104.0.1.0/24	[101 104]	Accepted from AS1 via ARIN Whois DB dump; rejected by others

### Route server graceful shutdown scenario

AS1 and AS2 are route server clients.

Route server performs a graceful shutdown and propagates the routes after adding the GRACEFUL\_SHUTDOWN BGP community to them.

### Max-prefix limits

General policy:

- limit: 4
- peering DB: True (increment: 1 / 20%)
- action: block

AS1:

- no peering DB

- no specific limits
- expected limit: 4

AS2 (client with `peering_db.increment` set to 0/0):

- peering DB (3)
- no specific limits
- expected limit: 3

AS3:

- specific limit: 2
- expected limit: 2

AS4:

- peering DB (4)
- no specific limits
- expected limit: 6 (given by  $(4 + 1) * 1.20$ )

### Path hiding mitigation technique

AS1, AS2, AS3 and AS4 are clients of the route server. AS4 has ADD-PATH rx on.

Only one prefix is used, *AS101\_pref\_ok1*, announced by AS101 to AS1 and AS2:

- AS101 -> AS1, AS\_PATH = [101]
- AS101 -> AS2, AS\_PATH = [101 101 101 101]

The route server has the path toward AS1 as the preferred one.

AS1 announces this prefix to the rs after having added the *do not announce to AS3* and *do not announce to AS4* BGP communities.

- When mitigation is on, AS3 and AS4 receive the prefix via the sub-optimal path toward AS2.
- When mitigation is off, AS3 does not receive the prefix at all, AS4 receives it because of ADD-PATH capability.

### Rich configuration example

A simple scenario to verify that the `general.yml` and `clients.yml` files distributed within the rich configuration example (`examples/rich`) lead to a working configuration.

The files used here are links to those provided within the `examples/rich` directory.

### RPKI INVALID routes tagging

Mostly to test hooks and include files in a scenario where a custom configuration allows to propagate RPKI INVALID routes to some selected clients and to tag them with locally significant BGP communities.

Hooks used:

- `announce_rpki_invalid_to_client`, implemented in the `header[4|6]` include files and used to discriminate which clients should receive INVALIDs;

- `post_announce_to_client`, implemented in the header include file and used to convert RFC8097 extended communities into locally significant ones.
- RPKI ROAs:

ID	Prefix	Max	ASN
1	2.0.8.0/24		101
2	2.0.9.0/24		102
3	2.0.128.0/20	23	101
4	3.0.8.0/24		103
5	3.0.9.0/24		102
6	3.0.128.0/20	23	103

ID	Prefix	Max	ASN
1	3002:0:8::/48		101
2	3002:0:9::/48		102
3	3002:0:8000::/33	34	101
4	3003:0:8::/48		103
5	3003:0:9::/48		102
6	3003:0:8000::/33	34	103

- Locally significant communities:

Validity state	BGP community
VALID	64512:1
INVALID	64512:2
UNKNOWN	64512:3

- AS1, receives only

Configured to receive INVALID routes using the hook `announce_rpki_invalid_to_client`, implemented in the local header [4 | 6] file.

- AS2:

Configured with `reject_invalid` False.

Announced prefixes:

Prefix ID	Prefix	AS_PATH	Expected result and BGP community received by AS1
AS2_valid1	2.0.8.0/24, 3002:0:8::/48	2 101	roa check ok, 64512:1 on AS1 and AS4
AS2_valid2	2.0.128.0/21, 3002:0:8000::/34	2 101	roa check ok, 64512:1 on AS1 and AS4
AS2_invalid1	2.0.9.0/24, 3002:0:9::/48	2	roa check fail (roa n. 2, bad origin ASN), 64512:2 on AS1 only
AS2_badlen	2.0.128.0/24, 3002:0:8000::/35	2 101	roa check fail (roa n. 3, bad length), 64512:2 on AS1 only
AS2_unknown1	2.2.0.0/16, 3002:3002::/32	2	roa check unknown, 64512:3 on AS1 and AS4

- AS3:

Configured with `reject_invalid` True.

Announced prefixes:

Prefix ID	Prefix	AS_PATH	Expected result and BGP community received by AS1
AS3_valid1	3.0.8.0/24, 3003:0:8::/48	3 103	roa check ok, 64512:1 on AS1 and AS4
AS3_valid2	3.0.128.0/21, 3003:0:8000::/34	3 103	roa check ok, 64512:1 on AS1 and AS4
AS3_invalid1	3.0.9.0/24, 3003:0:9::/48	3	roa check fail (roa n. 2, bad origin ASN), rejected
AS3_badlen	3.0.128.0/24, 3003:0:8000::/35	3 103	roa check fail (roa n. 3, bad length), rejected
AS3_unknown1	3.2.0.0/16 3003:3003::/32	2	roa check unknown, 64512:3 on AS1 and AS4

- AS4, receives only with no particular configuration.

### Tag prefixes/origin ASNs present/not-present in IRRDB

Built to test the `irrd.db.tag_as_set` option.

Two sub-scenarios exist for this test:

1. AS-SETs are populated with origin ASNs and prefixes reported below.
2. AS-SETs are empty.

Communities:

OK / Not OK	Comm
prefix OK	64512
prefix NOT OK	64513
origin OK	64514
origin NOT OK	64515
RPKI ROA OK	64516
ARIN Whois OK	64518
route wht list	64517

RPKI ROAs:

prefix	ASN
2.4.0.0/16	AS2
2.5.0.0/16	AS2
2.7.0.0/16	AS2
2.0.4.0/24	AS2
3.1.0.0/16	AS3
3.3.0.0/16	AS3
6.0.1.0/24	AS6

ARIN Whois DB entries:

prefix	ASN
2.6.0.0/16	AS2
2.7.0.0/16	AS2
3.2.0.0/16	AS3
3.3.0.0/16	AS3
2.0.5.0/24	AS3
6.0.1.0/24	AS6

## AS2

- allowed objects:
  - prefix: 2.0.0.0/16
  - origin: [2]
- configuration:
  - enforcing: no
  - tagging: yes
- white lists:
  - prefixes: 2.2.0.0/16
  - asns: 21

AS2 announces:

id	prefix	AS_PATH	prefix ok?	origin ok?	expected result 1	expected result 2
AS2_pref_ok_origin_01	1.0/24	2	yes	yes	64512 64514	64513 64515
AS2_pref_ko_origin_01	1.0/24	2	no	yes	64513 64514	64513 64515
AS3_pref_ok_origin_01	2.0/24	2 3	yes	no	64512 64515	64513 64515
AS3_pref_ko_origin_01	1.0/24	2 3	no	no	64513 64515	64513 64515
AS2_pref_wl_origin_01	2.1.0/24	2	yes (WL)	yes	64512 64514	64512 64515
AS2_pref_wl_origin_02	2.2.0/24	2 3	yes (WL)	no	64512 64515	the same
AS2_pref_wl_origin_03	2.3.0/24	2 21	yes (WL)	yes (WL)	64512 64514	the same
AS2_pref_ko_origin_03	2.1.0/24	2 21	no	yes (WL)	64513 64514	the same
AS2_pref_ok_origin_03	2.3.0/24	2 21	yes	yes (WL)	64512 64514	64513 64514
AS2_roa2	2.5.0.0/16	2	no	yes	64513 64514 64516	64513 64515
AS2_arin1	2.6.0.0/16	2	no	yes	64513 64514 64518	64513 64515
AS2_roa3_arin2	2.7.0.0/16	2	no	yes	64513 64514 64516 64518	64513 64515
AS2_ok_ok_roa3	2.0.4.0/24	2	yes	yes	64512 64514 64516	64513 64515
AS2_ok_ok_arin3	2.0.5.0/24	2	yes	yes	64512 64514 64518	64513 64515

### AS3

Not a route server client here, used just to track RPKI ROAs and ARIN Whois DB entries:

### AS4

- allowed objects:
  - prefix: 4.0.0.0/16
  - origin: 4
- configuration:
  - enforcing: origin only
  - tagging: yes
- white lists:
  - prefixes: 4.2.0.0/16
  - asns: 41
  - routes:
    - \* exact 4.4.0.0/16, AS 44
    - \* 4.5.0.0/16, AS 43
    - \* 4.6.0.0/16, no origin AS

AS4 announces:

id	prefix	AS_PATH	prefix ok?	origin ok?	expected result 1	expected result 2
AS4_pref_ok_origin	ok.D.1.0/24	4	yes	yes	64512 64514	rejected
AS4_pref_ko_origin	ok.L.0.0/24	4	no	yes	64513 64514	rejected
AS3_pref_ok_origin	ko.D.2.0/24	4 3	yes	no	rejected	rejected
AS3_pref_ko_origin	ko.D.1.0/24	4 3	no	no	rejected	rejected
AS4_pref_wl_origin	ok.2.1.0/24	4	yes (WL)	yes	64512 64514	rejected
AS4_pref_wl_origin	ko.2.2.0/24	4 3	yes (WL)	no	rejected	rejected
AS4_pref_wl_origin	wl.2.3.0/24	4 41	yes (WL)	yes (WL)	64512 64514	the same
AS4_pref_ko_origin	wl.3.1.0/24	4 41	no	yes (WL)	64513 64514	the same
AS4_pref_ok_origin	wl.0.3.0/24	4 41	yes	yes (WL)	64512 64514	64513 64514
AS4_routewl_1	4.4.0.0/16	4 44	r WL	r WL	64513 64515 64517	the same
AS4_routewl_2	4.4.1.0/24	4 44	r WL KO	r WL	rejected	rejected
AS4_routewl_3	4.5.1.0/24	4 43	r WL	r WL	64513 64515 64517	the same
AS4_routewl_4	4.5.2.0/24	4 45	r WL	r WL KO	rejected	rejected
AS4_routewl_5	4.6.1.0/24	4 45	r WL	r WL	64513 64515 64517	the same

## AS5

- allowed objects (AS-SET from PeeringDB):
  - prefix: 5.0.0.0/16
  - origin: 5

configuration:

- enforcing: prefix only
- tagging: yes
- white lists:
  - prefixes: 5.2.0.0/16
  - asns: 51

AS5 announces:



id	prefix	AS_PATH	prefix ok?	origin ok?	expected result 1	expected results 2
AS5_pref_ok_origin	6.0.0.0/24	5	yes	yes	64512 64514	rejected
AS5_pref_ko_origin	6.0.0.0/24	5	no	yes	rejected	rejected
AS3_pref_ok_origin	3.2.0.0/24	5 3	yes	no	64512 64515	rejected
AS3_pref_ko_origin	3.2.0.0/24	5 3	no	no	rejected	rejected
AS5_pref_wl_origin	6.2.1.0/24	5	yes (WL)	yes	64512 64514	64512 64515
AS5_pref_wl_origin	6.2.2.0/24	5 3	yes (WL)	no	64512 64515	the same
AS5_pref_wl_origin	6.2.3.0/24	5 51	yes (WL)	yes (WL)	64512 64514	the same
AS5_pref_ko_origin	6.3.1.0/24	5 51	no	yes (WL)	rejected	rejected
AS5_pref_ok_origin	6.0.3.0/24	5 51	yes	yes (WL)	64512 64514	rejected

## AS6

- allowed objects:
  - prefix: 6.0.0.0/16
  - origin: 6, 3

configuration:

- enforcing: both origin ASN and prefix
- tagging: yes
- white lists:
  - routes:
    - \* 3.2.0.0/16+, AS3 (1)

AS6 announces:

id	prefix	AS_PATH	prefix ok?	origin ok?	expected result 1	expected results 2
AS2_roa1	2.4.0.0/16	6 2	no	no	rejected	rejected
AS3_roa2	3.1.0.0/16	6 3	ROA	yes	64513 64514 64516	rejected
AS3_arin1	3.2.1.0/24	6 3	ARIN (1)	yes	64513 64514 64518	64513 64515 64517
AS3_roa3_arin2	3.3.0.0/16	6 3	no	yes	64513 64514 64516 64518	rejected
AS6_ok_ok_roa6_arin	6.0.1.0/24	6	yes	yes	64512 64514 64516 64518	rejected

1) The route white list is used to verify that: - in scenario 1, 3.2.1.0/24 AS3 is accepted and tagged with the ARIN db community, and not because of the white list entry; - in scenario 2, 3.2.1.0/24 AS3 is accepted anyway, but solely because of the route white list

## Reject policy: *tag*

This scenario uses the same base layout of the *global* one, with the addition of an *Invalid routes collector* that receives only the routes that have been classified as *invalid* by the route server.

All the test cases used here inherit from the `LiveScenario_TagRejectPolicy` class, that dynamically changes the `general.yml` content to reflect the use of the `tag reject_policy`: the BGP community used to mark the rejected routes and the reject reasons is `65520:x`.

BIRD and OpenBGPD are configured using *local* files to setup the sessions with the route collector and to properly announce only the invalid routes that have been previously marked with the `reject_reason` BGP community.

### 3.7.5 How to build custom scenarios

A live test scenario skeleton is provided in the `pierky/arouteserver/tests/live_tests/skeleton` directory.

It seems to be a complex thing but actually most of the work is already done in the underlying Python classes and prepared in the skeleton.

To configure the route server and its clients, please consider that the Docker network used by the framework is on `192.0.2.0/24` and `2001:db8:1:1::/64` subnets.

1. Initialize the new scenario into a new directory:

- using the `init-scenario` command:

```
arouteserver init-scenario ~/ars_scenarios/myscenario
```

- manually, by cloning the provided skeleton directory:

```
mkdir -p ~/ars_scenarios/myscenario
cp pierky/arouteserver/tests/live_tests/skeleton/* ~/ars_scenarios/myscenario
```

2. Document the scenario, for example in the `README.rst` file: write down which BGP speakers are involved, how they are configured, which prefixes they announce and what the expected result should be with regards of the route server's configuration and its policies.
3. Put the `general.yml`, `clients.yml` and `bogons.yml` configuration files you want to test in the new directory.
4. Configure your scenario and write your test functions in the `base.py` file.
  - Declare the BGP speakers you want to use in the `_setup_rs_instance()` and `_setup_instances()` methods of the base class.

```
classmethod SkeletonScenario._setup_instances()
```

Declare the BGP speaker instances that are used in this scenario.

The `cls.INSTANCES` attribute is a list of all the instances that are used in this scenario. It is used to render local Jinja2 templates and to transform them into real BGP speaker configuration files.

The `cls.RS_INSTANCE_CLASS` and `cls.CLIENT_INSTANCE_CLASS` attributes are set by the derived classes (`test_XXX.py`) and represent the route server class and the other BGP speakers class respectively.

– The first argument is the instance name.

– The second argument is the IP address that is used to run the instance. Here, the `cls.DATA` dictionary is used to lookup the real IP address to use, which is configured in the derived classes (`test_XXX.py`).

- The third argument is a list of files that are mounted from the local host (where Docker is running) to the container (the BGP speaker). The list is made of pairs in the form (local\_file, container\_file). The `cls.build_rs_cfg` and `cls.build_other_cfg` helper functions allow to render Jinja2 templates and to obtain the path of the local output files.

For the route server, the configuration is built using ARouteServer’s library on the basis of the options given in the YAML files.

For the other BGP speakers, the configuration must be provided in the Jinja2 files within the scenario directory.

Example:

```
@classmethod
def _setup_instances(cls):
    cls.INSTANCES = [
        cls._setup_rs_instance(),

        cls.CLIENT_INSTANCE_CLASS(
            "AS1",
            cls.DATA["AS1_IPAddress"],
            [
                (
                    cls.build_other_cfg("AS1.j2"),
                    "/etc/bird/bird.conf"
                )
            ]
        ),
        ...
    ]
```

- To ease writing the test functions, set instances names in the `set_instance_variables()` method.

`SkeletonScenario.set_instance_variables()`

Simply set local attributes for an easier usage later

The argument of `self._get_instance_by_name()` must be one of the instance names used in `_setup_instances()`.

Example:

```
def set_instance_variables(self):
    self.AS1 = self._get_instance_by_name("AS1")
    self.AS2 = self._get_instance_by_name("AS2")
    self.rs = self._get_instance_by_name("rs")
```

- Write test functions to verify that scenario’s expectations are met.

Some helper functions can be used:

- `LiveScenario.session_is_up(inst_a, inst_b)`  
Test if a BGP session between the two instances is up.

If a BGP session between the two instances is not up, the `TestCase.fail()` method is called and the test fails.

#### Parameters

- \* **inst\_a** – the `BGPSpeakerInstance` instance where the BGP session is looked for.
- \* **inst\_b** – the `BGPSpeakerInstance` instance that `inst_a` is expected to peer with.

Example:

```
def test_020_sessions_up(self):
    """{}: sessions are up"""
    self.session_is_up(self.rs, self.AS1)
    self.session_is_up(self.rs, self.AS2)
```

- `LiveScenario.receive_route` (*inst*, *prefix*, *other\_inst=None*, *as\_path=None*, *next\_hop=None*, *std\_comms=None*, *lrg\_comms=None*, *ext\_comms=None*, *local\_pref=None*, *filtered=None*, *only\_best=None*, *reject\_reason=None*)

Test if the BGP speaker receives the expected route(s).

If no routes matching the given criteria are found, the `TestCase.fail()` method is called and the test fails.

#### Parameters

- \* **inst** – the `BGPSpeakerInstance` instance where the routes are searched on.
- \* **prefix** (*str*) – the IPv4/IPv6 prefix of the routes to search for.
- \* **other\_inst** – if given, only routes received from this `BGPSpeakerInstance` instance are considered.
- \* **as\_path** (*str*) – if given, only routes with this `AS_PATH` are considered.
- \* **next\_hop** – can be a string or a `BGPSpeakerInstance` instance; if given, only routes that have a `NEXT_HOP` address matching this one are considered.
- \* **lrg\_comms**, **ext\_comms** (*std\_comms*,) – if given, only routes that carry these BGP communities are considered. Use an empty list (`[]`) to consider only routes with no BGP comms.
- \* **local\_pref** (*int*) – if given, only routes with local-pref equal to this value are considered.
- \* **filtered** (*bool*) – if given, only routes that have been (not) filtered are considered.
- \* **only\_best** (*bool*) – if given, only best routes are considered.
- \* **reject\_reason** (*int*) – valid only if *filtered* is `True`: if given the route must be reject with this reason code. It can be also a set of codes: in this case, the route must be rejected with one of those codes.

The list of valid codes is reported in docs/CONFIG.rst or at <https://arouteserver.readthedocs.io/en/latest/CONFIG.html#reject-policy>

Example:

```
def test_030_rs_receives_AS2_prefix(self):
    """{}: rs receives AS2 prefix"""
    self.receive_route(self.rs, self.DATA["AS2_prefix1"],
                      other_inst=self.AS2, as_path="2")
```

- `LiveScenario.log_contains` (*inst*, *msg*, *instances={}*)  
Test if the BGP speaker's log contains the expected message.

This only works for BGP speaker instances that support message logging: currently only BIRD.

If no log entries are found, the `TestCase.fail()` method is called and the test fails.

#### Parameters

- \* **inst** – the `BGPSpeakerInstance` instance where the expected message is searched on.
- \* **msg** (*str*) – the text that is expected to be found within BGP speaker's log.
- \* **instances** (*dict*) – a dictionary of pairs “<macro>: <BGPSpeakerInstance>” used to expand macros on the *msg* argument. Macros are expanded using the BGP speaker's specific client ID or protocol name.

### Example

Given *self.rs* the instance of the route server, and *self.AS1* the instance of one of its clients, the following code expands the “{AS1}” macro using the BGP speaker specific name for the instance *self.AS1* and then looks for it within the route server’s log:

```
self.log_contains(self.rs, "{AS1} bad ASN", {"AS1": self.AS1})
```

On BIRD, “{AS1}” will be expanded using the “protocol name” that BIRD uses to identify the BGP session with AS1.

Example:

```
def test_030_rs_rejects_bogon(self):
    """{}: rs rejects bogon prefix"""
    self.log_contains(self.rs,
                      "prefix is bogon - REJECTING {}".format(
                          self.DATA["AS2_bogon1"]))
    self.receive_route(self.rs, self.DATA["AS2_bogon1"],
                      other_inst=self.AS2, as_path="2",
                      filtered=True)
    # AS1 should not receive the bogon prefix from the route server
    with six.assertRaisesRegex(self, AssertionError, "Routes not found
↵"):
        self.receive_route(self.AS1, self.DATA["AS2_bogon1"])
```

5. Edit IP version specific and BGP speaker specific classes within the `test_XXX.py` files and set the prefix ID / real IP addresses mapping schema.

**class** `pierky.arouteserver.tests.live_tests.skeleton.test_bird4.SkeletonScenario_BIRDIPv4` BGP speaker specific and IP version specific derived class.

This class inherits all the test functions from the base class. Here, only IP version specific attributes are set, such as the prefix IDs / real IP prefixes mapping schema.

The prefix IDs reported within the `DATA` dictionary must be used in the parent class’ test functions to reference the real IP addresses/prefixes used in the scenario. Also the other BGP speakers’ configuration templates must use these IDs. For an example please see the “AS2.j2” file.

The `SHORT_DESCR` attribute can be set with a brief description of this scenario.

Example:

```
class SkeletonScenario_BIRDIPv4(SkeletonScenario):

    # Leave this to True in order to allow nose to use this class
    # to run tests.
    __test__ = True

    SHORT_DESCR = "Live test, BIRD, skeleton, IPv4"
    CONFIG_BUILDER_CLASS = BIRDConfigBuilder
    RS_INSTANCE_CLASS = BIRDInstanceIPv4
    CLIENT_INSTANCE_CLASS = BIRDInstanceIPv4
    IP_VER = 4

    DATA = {
        "rs_IPAddress": "99.0.2.2",
        "AS1_IPAddress": "99.0.2.11",
        "AS2_IPAddress": "99.0.2.22",
```

(continues on next page)

(continued from previous page)

```
"AS2_prefix1":          "2.0.1.0/24",
"AS2_bogon1":          "192.168.2.0/24"
}
```

6. Edit (or add) the template files that, once rendered, will produce the configuration files for the other BGP speakers (route server clients) that are involved in the scenario (the skeleton includes two template files, AS1.j2 and AS2.j2).

Example:

```
router id 192.0.2.22;

# This is the path where Python classes look for
# to search BIRD's log files.
log "/var/log/bird.log" all;
log syslog all;
debug protocols all;

protocol device { }

# Prefixes announced by this BGP speaker to the route server.
#
# The Jinja2 'data' variable refers to the class 'DATA' attribute.
#
# IP prefixes are not configured directly here, only a reference
# to their ID is given in order to maintain a single configuration
# file that can be used for both the IPv4 and the IPv6 versions
# of the scenario.
protocol static own_prefixes {
    route {{ data.AS2_prefix1 }} reject;
    route {{ data.AS2_bogon1 }} reject;
}

protocol bgp the_rs {
    local as 2;
    neighbor {{ data.rs_IPAddress }} as 999;
    import all;
    export all;
    connect delay time 1;
    connect retry time 1;
}
```

7. Run the tests using nose:

```
nosetests -vs ~/ars_scenarios/myscenario
```

Details about the code behind the live tests can be found in the LIVETESTS\_CODEDOC section.

## Debugging live tests scenarios

To debug custom scenarios some utilities are provided:

- the REUSE\_INSTANCES environment variable can be set when executing nose to avoid Docker instances to be torn down at the end of a run. When this environment variable is set, BGP speaker instances are started only the first time tests are executed, then are left up and running to allow debugging. When tests are executed again,

the BGP speakers' configuration is rebuilt and reloaded. **Be careful:** this mode can be used only when running tests of the same scenario, otherwise Bad Things (tm) may happen.

Example:

```
REUSE_INSTANCES=1 nosetests -vs tests/live_tests/scenarios/global/test_bird4.py
```

- once the BGP speaker instances are up (using the REUSE\_INSTANCES environment variable seen above), they can be queried using standard Docker commands:

```
$ # list all the running Docker instances
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
↳ STATUS          PORTS              NAMES
142f88379428       pierky/bird:1.6.3  "bird -c /etc/bird...  18 minutes ago
↳ Up 18 minutes    179/tcp            ars_AS101
26a9ec58dcf1       pierky/bird:1.6.3  "bird -c /etc/bird...  18 minutes ago
↳ Up 18 minutes    179/tcp            ars_AS2

$ # run 'birdcl show route' on ars_AS101
$ docker exec -it 142f88379428 birdcl show route
```

Some utilities are provided within the `/utils` directory to ease these tasks:

```
# execute the 'show route' command on the route server BIRD Docker instance
./utils/birdcl rs show route

# print the log of the route server
./utils/run rs cat /var/log/bird.log
```

The first argument (“rs” in the examples above) is the name of the instance as set in the `_setup_instances()` method.

- the BUILD\_ONLY environment variable can be set to skip all the tests and only build the involved BGP speakers' configurations. Docker instances are not started in this mode.

Example:

```
BUILD_ONLY=1 nosetests -vs tests/live_tests/scenarios/global/test_bird4.py
```

## 3.8 Testing realistic scenarios

Some *realistic* scenarios have been tested using ARouteServer by feeding it with lists of clients pulled from actual IXPs' members lists.

- **Euro-IX JSON member list** files exposed by the IXPs reported below have been used to automatically generate **ARouteServer clients.yml** file. AS-SETS and max-prefix limits from PeeringDB have been taken into account when they were not available in the Euro-IX JSON file.

List of IXPs used to run these tests:

- AMS-IX, VLAN ID 501 (“ISP”)
- BCIX, VLAN ID 1 (“BCIX Peering LAN”)
- BIX, VLAN ID 1 (“IPv4 Peering LAN”) and 2 (“IPv6 Peering LAN”)
- GR-IX

- INEX, VLAN ID 2 (“Peering VLAN #1”)
- LONAP, VLAN ID 1 (“LONAP Peering LAN #1”)
- SIX, VLAN ID 2 (“MTU 1500”)
- STHIX - Stockholm
- SwissIX

The files produced by the `clients-from-euroix` command can be found within the `tests/real/clients` directory (see it on [GitHub](#)).

- A **rich ARouteServer setup** has been used to build BIRD and OpenBGPD configurations, in order to enable as many features as possible: filters based on “same AS” NEXT\_HOP, invalid AS\_PATHs, transit-free ASNs, IRRDB information, and also max-prefix limits and BGP communities for blackhole filtering, selective announcement, prepending...

The [full description](#) of the configuration can be found in the `general.html` file, automatically generated from the `general.yml` file.

- For each IXP, configurations for BIRD and OpenBGPD (both 6.0 and 6.2) have been finally built using ARouteServer and loaded into an instance of their respective daemon, to verify that no errors occurred:
  - for BIRD, a Docker container has been used;
  - for OpenBGPD 6.0, a virtual server has been instantiated on [Vultr](#) in order to have enough resources to process the configuration;
  - for OpenBGPD 6.2, only a few configurations were tested because of lack of resources on my machines :-/

**Results** can be found within the `tests/real/last` file ([here on GitHub](#)). They are also reported below.

```
Real configs: AMS-IX, BIRD, IPv4, build ... ok
Real configs: AMS-IX, BIRD, IPv6, build ... ok

-----

Ran 2 tests in 801.256s

OK
Building config for bird, IPv4: 664 seconds
Building config for bird, IPv6: 137 seconds

=====

Real configs: AMS-IX, OpenBGPD 6.0, build ... ok

-----

Ran 1 test in 625.545s

OK
Building config for openbgpd 6.0, IPv4 & IPv6: 626 seconds

=====

Real configs: AMS-IX, OpenBGPD 6.2, build ... ok

-----

Ran 1 test in 642.808s

OK
```

(continues on next page)



(continued from previous page)

```
Building config for openbgpd 6.2, IPv4 & IPv6: 643 seconds

=====
Real configs: AMS-IX, BIRD, IPv4, load ... ok
Real configs: AMS-IX, BIRD, IPv6, load ... ok

-----

Ran 2 tests in 23.333s

OK
Loading config for bird, IPv4: 5 seconds
Loading config for bird, IPv6: 3 seconds

=====
Real configs: AMS-IX, OpenBGPD 6.0, load ... ok

-----

Ran 1 test in 2474.119s

OK
Loading config for openbgpd 6.0, IPv4 & IPv6: 2469 seconds

=====
Real configs: AMS-IX, OpenBGPD 6.2, load ... SKIP: Lack of resources

-----

Ran 1 test in 0.004s

OK (SKIP=1)

=====
Real configs: BCIX, BIRD, IPv4, build ... ok
Real configs: BCIX, BIRD, IPv6, build ... ok

-----

Ran 2 tests in 252.759s

OK
Building config for bird, IPv4: 206 seconds
Building config for bird, IPv6: 46 seconds

=====
Real configs: BCIX, OpenBGPD 6.0, build ... ok

-----

Ran 1 test in 195.231s

OK
Building config for openbgpd 6.0, IPv4 & IPv6: 196 seconds

=====
```

(continues on next page)

(continued from previous page)

```
Real configs: BCIX, OpenBGPD 6.2, build ... ok

-----

Ran 1 test in 192.919s

OK
Building config for openbgpd 6.2, IPv4 & IPv6: 193 seconds

=====

Real configs: BCIX, BIRD, IPv4, load ... ok
Real configs: BCIX, BIRD, IPv6, load ... ok

-----

Ran 2 tests in 18.258s

OK
Loading config for bird, IPv4: 3 seconds
Loading config for bird, IPv6: 3 seconds

=====

Real configs: BCIX, OpenBGPD 6.0, load ... ok

-----

Ran 1 test in 275.215s

OK
Loading config for openbgpd 6.0, IPv4 & IPv6: 271 seconds

=====

Real configs: BCIX, OpenBGPD 6.2, load ... SKIP: Lack of resources

-----

Ran 1 test in 0.004s

OK (SKIP=1)

=====

Real configs: BIX_IPv4, BIRD, IPv4, build ... ok
Real configs: BIX_IPv4, BIRD, IPv6, build ... SKIP: IPv4 only
Real configs: BIX_IPv6, BIRD, IPv4, build ... SKIP: IPv6 only
Real configs: BIX_IPv6, BIRD, IPv6, build ... ok

-----

Ran 4 tests in 142.915s

OK (SKIP=2)
Building config for bird, IPv4: 111 seconds

=====

Building config for bird, IPv6: 32 seconds
```

(continues on next page)

(continued from previous page)

```
=====
Real configs: BIX_IPv4, OpenBGPD 6.0, build ... ok
Real configs: BIX_IPv6, OpenBGPD 6.0, build ... ok

-----

Ran 2 tests in 201.500s

OK
Building config for openbgpd 6.0, IPv4 & IPv6: 103 seconds

=====

Building config for openbgpd 6.0, IPv4 & IPv6: 98 seconds

-----

Real configs: BIX_IPv4, OpenBGPD 6.2, build ... ok
Real configs: BIX_IPv6, OpenBGPD 6.2, build ... ok

-----

Ran 2 tests in 205.375s

OK
Building config for openbgpd 6.2, IPv4 & IPv6: 106 seconds

-----

Building config for openbgpd 6.2, IPv4 & IPv6: 99 seconds

=====

Real configs: BIX_IPv4, BIRD, IPv4, load ... ok
Real configs: BIX_IPv4, BIRD, IPv6, load ... SKIP: IPv4 only
Real configs: BIX_IPv6, BIRD, IPv4, load ... SKIP: IPv6 only
Real configs: BIX_IPv6, BIRD, IPv6, load ... ok

-----

Ran 4 tests in 18.206s

OK (SKIP=2)
Loading config for bird, IPv4: 3 seconds

-----

Loading config for bird, IPv6: 3 seconds

=====

Real configs: BIX_IPv4, OpenBGPD 6.0, load ... ok
Real configs: BIX_IPv6, OpenBGPD 6.0, load ... ok

-----

Ran 2 tests in 128.880s

OK
Loading config for openbgpd 6.0, IPv4 & IPv6: 59 seconds
```

(continues on next page)

(continued from previous page)

```
=====  
Loading config for openbgpd 6.0, IPv4 & IPv6: 60 seconds  
  
=====  
Real configs: BIX_IPv4, OpenBGPD 6.2, load ... SKIP: Lack of resources  
Real configs: BIX_IPv6, OpenBGPD 6.2, load ... SKIP: Lack of resources  
  
-----  
Ran 2 tests in 0.008s  
  
OK (SKIP=2)  
  
=====  
  
=====  
Real configs: GR-IX, BIRD, IPv4, build ... ok  
Real configs: GR-IX, BIRD, IPv6, build ... ok  
  
-----  
Ran 2 tests in 50.569s  
  
OK  
Building config for bird, IPv4: 33 seconds  
Building config for bird, IPv6: 18 seconds  
  
=====  
Real configs: GR-IX, OpenBGPD 6.0, build ... ok  
  
-----  
Ran 1 test in 61.728s  
  
OK  
Building config for openbgpd 6.0, IPv4 & IPv6: 62 seconds  
  
=====  
Real configs: GR-IX, OpenBGPD 6.2, build ... ok  
  
-----  
Ran 1 test in 61.141s  
  
OK  
Building config for openbgpd 6.2, IPv4 & IPv6: 61 seconds  
  
=====  
Real configs: GR-IX, BIRD, IPv4, load ... ok  
Real configs: GR-IX, BIRD, IPv6, load ... ok  
  
-----  
Ran 2 tests in 18.477s
```

(continues on next page)

(continued from previous page)

```
OK
Loading config for bird, IPv4: 4 seconds
Loading config for bird, IPv6: 3 seconds

=====
Real configs: GR-IX, OpenBGPD 6.0, load ... ok

-----

Ran 1 test in 40.555s

OK
Loading config for openbgpd 6.0, IPv4 & IPv6: 35 seconds

=====
Real configs: GR-IX, OpenBGPD 6.2, load ... ok

-----

Ran 1 test in 135.300s

OK
Loading config for openbgpd 6.2, IPv4 & IPv6: 120 seconds

=====
Real configs: INEX, BIRD, IPv4, build ... ok
Real configs: INEX, BIRD, IPv6, build ... ok

-----

Ran 2 tests in 55.797s

OK
Building config for bird, IPv4: 36 seconds
Building config for bird, IPv6: 20 seconds

=====
Real configs: INEX, OpenBGPD 6.0, build ... ok

-----

Ran 1 test in 73.090s

OK
Building config for openbgpd 6.0, IPv4 & IPv6: 73 seconds

=====
Real configs: INEX, OpenBGPD 6.2, build ... ok

-----

Ran 1 test in 73.857s

OK
Building config for openbgpd 6.2, IPv4 & IPv6: 73 seconds
```

(continues on next page)

(continued from previous page)

```
=====
Real configs: INEX, BIRD, IPv4, load ... ok
Real configs: INEX, BIRD, IPv6, load ... ok

-----

Ran 2 tests in 18.180s

OK
Loading config for bird, IPv4: 3 seconds
Loading config for bird, IPv6: 3 seconds

=====

Real configs: INEX, OpenBGPD 6.0, load ... ok

-----

Ran 1 test in 44.156s

OK
Loading config for openbgpd 6.0, IPv4 & IPv6: 39 seconds

=====

Real configs: INEX, OpenBGPD 6.2, load ... ok

-----

Ran 1 test in 155.570s

OK
Loading config for openbgpd 6.2, IPv4 & IPv6: 138 seconds

=====

Real configs: LONAP, BIRD, IPv4, build ... ok
Real configs: LONAP, BIRD, IPv6, build ... ok

-----

Ran 2 tests in 109.184s

OK
Building config for bird, IPv4: 77 seconds
Building config for bird, IPv6: 32 seconds

=====

Real configs: LONAP, OpenBGPD 6.0, build ... ok

-----

Ran 1 test in 135.533s

OK
Building config for openbgpd 6.0, IPv4 & IPv6: 136 seconds

=====

Real configs: LONAP, OpenBGPD 6.2, build ... ok
```

(continues on next page)

(continued from previous page)

```
-----  
Ran 1 test in 137.810s  
  
OK  
Building config for openbgpd 6.2, IPv4 & IPv6: 138 seconds  
  
=====
```

Real configs: LONAP, BIRD, IPv4, load ... ok  
Real configs: LONAP, BIRD, IPv6, load ... ok

```
-----  
Ran 2 tests in 18.279s  
  
OK  
Loading config for bird, IPv4: 4 seconds  
Loading config for bird, IPv6: 4 seconds  
  
=====
```

Real configs: LONAP, OpenBGPD 6.0, load ... ok

```
-----  
Ran 1 test in 86.025s  
  
OK  
Loading config for openbgpd 6.0, IPv4 & IPv6: 81 seconds  
  
=====
```

Real configs: LONAP, OpenBGPD 6.2, load ... SKIP: Lack of resources

```
-----  
Ran 1 test in 0.005s  
  
OK (SKIP=1)  
  
=====
```

Real configs: SIX, BIRD, IPv4, build ... ok  
Real configs: SIX, BIRD, IPv6, build ... ok

```
-----  
Ran 2 tests in 244.815s  
  
OK  
Building config for bird, IPv4: 194 seconds  
Building config for bird, IPv6: 51 seconds  
  
=====
```

Real configs: SIX, OpenBGPD 6.0, build ... ok

```
-----  
Ran 1 test in 226.218s  
  
OK
```

(continues on next page)

(continued from previous page)

```
Building config for openbgpd 6.0, IPv4 & IPv6: 226 seconds
```

```
=====
Real configs: SIX, OpenBGPD 6.2, build ... ok
-----
```

```
Ran 1 test in 228.808s
```

```
OK
```

```
Building config for openbgpd 6.2, IPv4 & IPv6: 228 seconds
```

```
=====
Real configs: SIX, BIRD, IPv4, load ... ok
Real configs: SIX, BIRD, IPv6, load ... ok
-----
```

```
Ran 2 tests in 18.345s
```

```
OK
```

```
Loading config for bird, IPv4: 3 seconds
```

```
Loading config for bird, IPv6: 4 seconds
```

```
=====
Real configs: SIX, OpenBGPD 6.0, load ... ok
-----
```

```
Ran 1 test in 202.955s
```

```
OK
```

```
Loading config for openbgpd 6.0, IPv4 & IPv6: 198 seconds
```

```
=====
Real configs: SIX, OpenBGPD 6.2, load ... SKIP: Lack of resources
-----
```

```
Ran 1 test in 0.004s
```

```
OK (SKIP=1)
```

```
=====
Real configs: STHIX, BIRD, IPv4, build ... ok
Real configs: STHIX, BIRD, IPv6, build ... ok
-----
```

```
Ran 2 tests in 115.921s
```

```
OK
```

```
Building config for bird, IPv4: 86 seconds
```

```
Building config for bird, IPv6: 30 seconds
```

(continues on next page)



(continued from previous page)

```
Real configs: STHIX, OpenBGPD 6.0, build ... ok

-----

Ran 1 test in 126.877s

OK
Building config for openbgpd 6.0, IPv4 & IPv6: 127 seconds

=====

Real configs: STHIX, OpenBGPD 6.2, build ... ok

-----

Ran 1 test in 127.479s

OK
Building config for openbgpd 6.2, IPv4 & IPv6: 128 seconds

=====

Real configs: STHIX, BIRD, IPv4, load ... ok
Real configs: STHIX, BIRD, IPv6, load ... ok

-----

Ran 2 tests in 18.163s

OK
Loading config for bird, IPv4: 3 seconds
Loading config for bird, IPv6: 3 seconds

=====

Real configs: STHIX, OpenBGPD 6.0, load ... ok

-----

Ran 1 test in 87.129s

OK
Loading config for openbgpd 6.0, IPv4 & IPv6: 82 seconds

=====

Real configs: STHIX, OpenBGPD 6.2, load ... SKIP: Lack of resources

-----

Ran 1 test in 0.004s

OK (SKIP=1)

=====

Real configs: SwissIX, BIRD, IPv4, build ... ok
Real configs: SwissIX, BIRD, IPv6, build ... ok

-----

Ran 2 tests in 195.399s
```

(continues on next page)

(continued from previous page)

```
OK
Building config for bird, IPv4: 152 seconds
Building config for bird, IPv6: 43 seconds

=====
Real configs: SwissIX, OpenBGPD 6.0, build ... ok

-----

Ran 1 test in 177.015s

OK
Building config for openbgpd 6.0, IPv4 & IPv6: 177 seconds

=====
Real configs: SwissIX, OpenBGPD 6.2, build ... ok

-----

Ran 1 test in 179.282s

OK
Building config for openbgpd 6.2, IPv4 & IPv6: 179 seconds

=====
Real configs: SwissIX, BIRD, IPv4, load ... ok
Real configs: SwissIX, BIRD, IPv6, load ... ok

-----

Ran 2 tests in 18.255s

OK
Loading config for bird, IPv4: 3 seconds
Loading config for bird, IPv6: 3 seconds

=====
Real configs: SwissIX, OpenBGPD 6.0, load ... ok

-----

Ran 1 test in 189.827s

OK
Loading config for openbgpd 6.0, IPv4 & IPv6: 185 seconds

=====
Real configs: SwissIX, OpenBGPD 6.2, load ... SKIP: Lack of resources

-----

Ran 1 test in 0.004s

OK (SKIP=1)

=====
```

## 3.9 Future work

### 3.9.1 Short term

- RTT-based communities: extend support to add NO\_EXPORT / NO\_ADVERTISE
- Informative community with the measured RTT of the announcing peer
- New feature: CLI option to build configs based on templates/groups only and avoid client specific settings

### 3.9.2 Mid term

- OpenBGPD: consider dropping the use of macros for ASN and prefix lists
- New feature: group clients by AFI/ASN (OpenBGPD only)
- Split configuration in multiple files
- Doc: better documentation
- Doc: contributing section
- Doc: schema of data that can be used within J2 templates

### 3.9.3 Long term

- New feature: path-hiding mitigation technique on OpenBGPD
- New feature: routing policies based on RPSL import-via/export-via
- New feature: other BGP speakers support (GoBGP, ...)
- New feature: balance clients among  $n$  different configurations (for multiple processes - see [Scaling BIRD Route-servers](#))

## 3.10 Contributing

CODEDOC\_enrichers

LIVETESTS\_CODEDOC

## 3.11 Change log

---

**Note: Upgrade notes:** after upgrading, run the `arouteserver setup-templates` command to sync the local templates with those distributed with the new version. More details on the [Upgrading](#) section of the documentation.

---

### 3.11.1 v0.18.0

- New: add support for BIRD 1.6.4 and OpenBGPD/OpenBSD 6.3.

This release **breaks backward compatibility** (OpenBGPD configs only): the default target version used to build OpenBGPD configurations (when the `--target-version` argument is not given) is now 6.2; previously it was 6.0. Use the `--target-version 6.0` command line argument to build 6.0 compatible configurations.

- Improvement: transit-free ASNs filters are applied also to sessions toward transit-free peers.

Related: [issue #21 on GitHub](#).

- Fix (minor): better handling of user answers in `configure` and `setup` commands.
- Fix: `clients-from-peeringdb`, list of IXPs retrieved from PeeringDB and no longer from IXFDB.

### 3.11.2 v0.17.3

- Fix: `clients-from-euroix` command, use the configured cache directory.

### 3.11.3 v0.17.2

- Fix: `configure` command, omit extended communities for OpenBGPD configurations.

This is to avoid the need of using the `--ignore-issues extended_communities` command line argument.

- Improvement: environment variables expansion when YAML configuration files are read.

### 3.11.4 v0.17.1

- Fix: minor installation issues.

### 3.11.5 v0.17.0

- New feature: allow to set the source of IRR objects.

AS-SETs can be prepended with an optional source: `RIPE::AS-FOO`, `RIPE::AS64496:AS-FOO`.

- New feature: support for RPKI-based Origin Validation added to OpenBGPD configurations.

RPKI ROAs must be loaded from a RIPE RPKI Validator cache file (local or via HTTP). Mostly inspired by Job Snijders' tool <https://github.com/job/rtrsub>

- Improvement: RPKI ROAs can be loaded from a local file too.

The file must be in RIPE RPKI Validator cache format.

- Fix (minor): remove internal communities before accepting blackhole routes tagged with a custom blackhole community.

This bug did not affect routes tagged with the BLACKHOLE community; anyway, the internal communities were scrubbed before routes were announced to clients.

### 3.11.6 v0.16.2

- Fix: avoid empty lists of prefixes when a client's `white_list_pref` contains only prefixes for an IP version different from the current one.

### 3.11.7 v0.16.1

- Fix: handle the new version of the JSON schema built by [arin-whois-bulk-parser](#).

### 3.11.8 v0.16.0

- Improvement: OpenBGPD, more flexibility for inbound communities values.

This allows to use inbound 'peer\_as' communities which overlap with other inbound communities whose last *part* is a private ASN.

- New feature: use ARIN Whois database dump to authorize routes.

This feature allows to accept those routes whose origin ASN is authorized by a client AS-SET, whose prefix has not a corresponding route object but is covered by an ARIN Whois record for the same origin ASN.

- Improvement: extend the use of *RPKI ROAs as route objects* and *ARIN Whois database dump* to `tag_as_set-only` mode.

Before of this, the *RPKI ROAs as route objects* and *ARIN Whois DB dump* features were used only when origin AS and prefix enforcing was set. Starting with this release they are used even when enforcing is not configured and only the `tag_as_set` mode is used.

### 3.11.9 v0.15.0

- New feature: `configure` and `show_config support` commands.
  - `configure`: it can be used to quickly generate a route server policy definition file (`general.yml`) on the basis of best practices and suggestions.
  - `show_config`: to display current configuration settings and also options that have been left to their default values.
- New feature: `ixf-member-export` command, to build [IX-F Member Export JSON files](#) from the list of clients.
- Improvement: cache expiry time values can be set for each external resource type: PeeringDB info, IRR data, ...

### 3.11.10 v0.14.1

- Fix: BIRD, “Unknown instruction 8574 in same (~)” error when reloading IPv6 configurations.

A [missing case](#) for the `!~` operator triggers this bug when neighbors are established and trying to reload `bird6` configuration.

Related: [issue #20 on GitHub](#).

### 3.11.11 v0.14.0

This release **breaks backward compatibility** (OpenBGPD configs only): for OpenBGPD configurations, starting with this release the Site of Origin Extended BGP communities in the range 65535:\* (soo 65535:\*) are reserved for internal reasons.

- New feature: use RPKI ROAs as if they were route objects.

This feature allows to accept those routes whose origin ASN is authorized by a client AS-SET, whose prefix is not but it is covered by a RPKI ROA for the same origin ASN.

Related: [issue #19 on GitHub](#).

- New feature: automatic checking for new releases.

This can be disabled by setting `check_new_release` to `False` in `arouteserver.yml`.

- Improvement: routes accepted solely because of a `white_list_route` entry are now tagged with the `route_validated_via_white_list` BGP community.
- Fix: on OpenBGPD configurations, in case of duplicate definition of a client's AS-SETs, duplicate BGP informational communities were added after the IRR validation process.

### 3.11.12 v0.13.0

- New feature: an option to set RFC1997 well-known communities (NO\_EXPORT/NO\_ADVERTISE) handling policy: pass-through or strict RFC1997 behaviour.

This **breaks backward compatibility**: previously, NO\_EXPORT/NO\_ADVERTISE communities were treated accordingly to the default implementation of the BGP speaker daemon (BIRD, OpenBGPD). Now, ARoute-Server's default setting is to treat routes tagged with those communities transparently, that is to announce them to other clients and to pass-through the original RFC1997 communities.

- Improvement: when using PeeringDB records to configure the max-prefix limits, a margin is taken into account to accommodate networks that fill the PeeringDB records with their exact route announcement count.

This **breaks backward compatibility**: if using max-prefix from PeeringDB, current limits will be raised by the default increment values (+100, +15%): this behaviour can be reverted to the pre-v0.13.0 situation by explicitly setting the `max_prefix.peering_db.increment` configuration section to 0/0.

Related: [issue #12 on GitHub](#).

- New feature: client-level white lists for IRRdb-based filters.

This allows to manually enter routes that must always be accepted by IRRdb-level checks and prefixes and ASNs that must be treated as if they were included within client's AS-SETs.

Related: [issue #16 on GitHub](#).

### 3.11.13 v0.12.3

- Improvement: always take the AS\*n\* macro into account when building IRRdb-based filters.

Related: [issue #15 on GitHub](#).

### 3.11.14 v0.12.2

- Fix: an issue on OpenBGPD builder class was preventing features offered via large BGP communities only from being actually implemented into the final configuration.

Related: [issue #11 on GitHub](#).

### 3.11.15 v0.12.1

- Fix an issue that was impacting templates upgrading under certain circumstances.

Related: [issue #10 on GitHub](#).

### 3.11.16 v0.12.0

- OpenBGPD 6.2 support.
- New feature: [Graceful BGP session shutdown](#) support, to honor GRACEFUL\_SHUTDOWN communities received from clients and also to perform graceful shutdown of the route server itself (`--perform-graceful-shutdown` [command line argument](#)).

### 3.11.17 v0.11.0

- Python 3.4 support.
- Improvement: GT registry removed from the sources used to gather info from IRRDB.  
Related: [PR #8 on GitHub](#).
- Improvement: multiple AS-SETs used for the same client are now grouped together and queried at one time. This allows to leverage bgpq3's ability and speed to aggregate results in order to have smaller configuration files.

### 3.11.18 v0.10.0

- New feature: when IRRDB-based filters are enabled and no AS-SETs are configured for a client, if the `cfg.filtering.rrdb.peering_db` option is set ARouteServer tries to fetch their values from the client's ASN record on PeeringDB.  
Related: [issue #7 on GitHub](#).
- Improvement: config building process performances,
  - reduced memory consumption by moving IRRDB information from memory to temporary files;
  - responses for empty/missing resources are also cached;
  - fix a wrong behaviour that led to multiple PeeringDB requests for the same ASN.
- Improvement: `clients-from-euroix` command, the new `--merge-from-peeringdb` option can be used to integrate missing information into the output clients list by fetching AS-SETs and max-prefix limit from PeeringDB.

### 3.11.19 v0.9.3

- Fix: OpenBGPD, an issue was causing values > 65535 to be used in standard BGP communities matching.

### **3.11.20 v0.9.2**

- Fix: remove quotes from clients description.
- Fix: OpenBGPD, syntax error for prefix lists with ‘range X - X’ format.
- Fix: `clients-from-euroix` command, members with multiple `vlan` objects with the same `vlan_id` were not properly listed in the output, only the first object was used.

### **3.11.21 v0.9.1**

- Improvement: BIRD, new default debug options (`states`, `routes`, `filters`, `interfaces`, `events`, was `all`).

If needed, they can be overwritten using the header `custom.local` file.

- Fix: `enrichers` errors handling reported a generic message with no further details.
- Fix: HTTP 404 error handling for “Entity not found” error from PeeringDB.
- Fix: OpenBGPD, large prefix lists were causing a “string too long” error.
- Fix: OpenBGPD, clients descriptions longer than 31 characters were not properly truncated.

### **3.11.22 v0.9.0**

- New feature: RTT-based communities to control propagation of routes on the basis of peers round trip time.
- Improvement: in conjunction with the “tag” reject policy, the `rejected_route_announced_by` BGP community can be used to track the ASN of the client that announced an invalid route to the server.
- Fix: when the “tag” reject policy is used, verify that the `reject_cause` BGP community is also set.

### **3.11.23 v0.8.1**

- Fix: default user configuration path not working.

### **3.11.24 v0.8.0**

- New feature: `reject policy` configuration option, to control how invalid routes must be treated: immediately discarded or kept for troubleshooting purposes, analysis or statistic reporting.
- New tool: `invalid routes reporter`.
- Fix: the following networks have been removed from the `bogons.yml` file: 193.239.116.0/22, 80.249.208.0/21, 164.138.24.80/29.

### **3.11.25 v0.7.0**

- New feature: `custom BGP communities` can be configured on a client-by-client basis to tag routes entering the route server (for example, for informative purposes).
- Fix: validation of BGP communities configuration for OpenBGPD.

Error is given if a peer-AS-specific BGP community overlaps with another community, even if the last part of the latter is a private/reserved ASN.



- Improvement: the custom `!include <filepath>` statement can be used now in YAML configuration files to include other files.

More details [here](#).

- Improvement: IRRDB-based filters can be configured to allow more specific prefixes (`allow_longer_prefixes` option).

### 3.11.26 v0.6.0

- OpenBGPD 6.1 support: enable large BGP communities support.
- Improvement: the `clients-from-peeringdb` command now uses the [IX-F database](#) to show a list of IXP and their PeeringDB ID.
- Improvement: enable NEXT\_HOP rewriting for IPv6 blackhole filtering requests on OpenBGPD after [OpenBSD 6.1 fixup](#).  
Related: [issue #3](#).
- Improvement: BIRD, client-level `.local` file.
- Improvement: next-hop checks, the `authorized_addresses` option allows to authorize IP addresses of non-client routers for NEXT\_HOP attribute of routes received from a client.

### 3.11.27 v0.5.0

- Fix: avoid the use of standard communities in the range 65535:x.
- Improvement: option to set max-prefix restart timer for OpenBGPD.
- Deleted feature: tagging of routes a' la RPKI-Light has been removed.
  - The `reject_invalid` flag, that previously was on general scope only, now can be set on a client-by-client basis.
  - The `roa_valid`, `roa_invalid`, and `roa_unknown` communities no longer exist.

Related: [issue #4 on GitHub](#)

This **breaks backward compatibility**.

- New feature: [BIRD hooks](#) to add site-specific custom implementations.
- Improvement: [BIRD local files](#).

This **breaks backward compatibility**: previously, `*.local`, `*.local4` and `*.local6` files that were found in the same directory where the BIRD configuration was stored were automatically included. Now, only the `header([4|6]).local` and `footer([4|6]).local` files are included, depending on the values passed to the `--use-local-files` command line argument.

- Improvement: `setup` command and program's configuration file.

The default path of the cache directory (`cache_dir` option) has changed: it was `/var/lib/arouteserver` and now it is `cache`, that is a directory which is relative to the `cfg_dir` option (by default, the directory where the program's configuration file is stored).

### 3.11.28 v0.4.0

- OpenBGPD support (some [limitations](#) apply).
- Add MD5 password support on clients configuration.
- The `build` command used to generate route server configurations has been removed in favor of BGP-speaker-specific sub-commands: `bird` and `openbgpd`.

### 3.11.29 v0.3.0

- New `--test-only` flag for builder commands.
- New `--clients-from-euroix` [command](#) to build the `clients.yml` file on the basis of records from an [Euro-IX member list JSON file](#).  
This also allows the [integration](#) with [IXP-Manager](#).
- New BGP communities: add `NO_EXPORT` and/or `NO_ADVERTISE` to any client or to specific peers.
- New option (set by default) to automatically add the `NO_EXPORT` community to blackhole filtering announcements.

### 3.11.30 v0.2.0

- `setup-templates` command to just sync local templates with those distributed within a new release.
- Multithreading support for tasks that acquire data from external sources (IRRDB info, PeeringDB max-prefix).  
Can be set using the `threads` option in the `arouteserver.yml` configuration file.
- New `template-context` command, useful to dump the list of context variables and data that can be used inside a template.
- New empty AS-SETs handling: if an AS-SET is empty, no errors are given but only a warning is logged and the configuration building process goes on.  
Any client with IRRDB enforcing enabled and whose AS-SET is empty will have its routes rejected by the route server.

### 3.11.31 v0.1.2

- Fix local files usage among IPv4/IPv6 processes.

Before of this release, only `.local` files were included into the route server configuration, for both the IPv4 and IPv6 configurations. After this, `.local` files continue to be used for both the address families but `.local4` and `.local6` files can also be used to include IP version specific options, depending on the IP version used to build the configuration. Details [here](#).

To upgrade:

```
# pull from GitHub master branch or use pip:
pip install --upgrade arouteserver

# install the new template files into local system
arouteserver setup
```

### 3.11.32 v0.1.1

- Add local static files into the route server's configuration.

### 3.11.33 v0.1.0

- First beta version.

### 3.11.34 v0.1.0a11

- The `filtering.rps1` section of general and clients configuration files has been renamed into `filtering.irrdb`.
- The command line argument `--template-dir` has been renamed into `--templates-dir`.
- New options in the program's configuration file: `bgpq3_host` and `bgpq3_sources`, used to set `bgpq3 -h` and `-S` arguments when gathering info from IRRDBs.

### 3.11.35 v0.1.0a10

- New command to build textual representations of configurations: `html`.

### 3.11.36 v0.1.0a9

- New command to initialize a custom live test scenario: `init-scenario`.

### 3.11.37 v0.1.0a8

- New feature: selective path prepending via BGP communities.
- The `control_communities` general option has been removed: it was redundant.

### 3.11.38 v0.1.0a7

- Improved communities configuration and handling.
- Fix issue on standard communities matching against 32-bit ASNs.
- Fix issue on IPv6 prefix validation.

### 3.11.39 v0.1.0a6

- New feature: RPKI-based filtering/tagging.

### 3.11.40 v0.1.0a5

- New feature: transit-free ASNs filtering.
- Program command line: `subcommands + clients-from-peeringdb`.
- More logging and some warning.

### **3.11.41 v0.1.0a4**

- Fix issue with GTSM default value.
- Add default route to bogons.
- Better as-sets handling and cache handling.
- Config syntax change: clients 'as' -> 'asn'.
- AS-SETs at AS-level.
- Live tests: path hiding mitigation scenario.
- Improvements in templates.

### **3.11.42 v0.1.0a3**

- Fix some cache issues.

### **3.11.43 v0.1.0a2**

- Packaging.
- System setup via `arouteserver --setup`.

### **3.11.44 v0.1.0a1**

First push on GitHub.

## CHAPTER 4

---

### Presentations

---

- RIPE74, 10 May 2017, Connect Working Group: [video \(9:53\)](#), [slides \(PDF\)](#)
- Salottino MIX, 30 May 2017: [slides](#)

Mentions / endorsements:

- Job Snijders, LINX99, 20 November 2017: [slides](#)



## CHAPTER 5

---

### Who is using ARouteServer?

---

- IX-Denver, BIRD.
- PIT-IX, BIRD.
- YYCIX, OpenBGPD.

Are you using it? Do you want to be listed here? [Drop me a message!](#)





## CHAPTER 6

---

Status

---

**Beta testing**, looking for testers and reviewers.

Anyone who wants to share his/her point of view, to review the output configurations or to test them is **more than welcome!**



## CHAPTER 7

---

Bug? Issues? Support requests?

---

But also suggestions? New ideas?

Please create an [issue on GitHub](#) or [drop me a message](#).

A Slack channel is also available on the [network.toCode\(\)](#) community: **arouteserver**.



## CHAPTER 8

---

Author

---

Pier Carlo Chiodi - <https://pierky.com>

Blog: <https://blog.pierky.com> Twitter: @pierky



## H

HOOKS (pierky.arouteserver.builder.BIRDConfigBuilder  
attribute), 23

## L

LOCAL\_FILES\_IDS (pierky.arouteserver.builder.BIRDConfigBuilder  
attribute), 22

LOCAL\_FILES\_IDS (pierky.arouteserver.builder.OpenBGPDCConfigBuilder  
attribute), 22