fetchmesh

Maxime Mouchet

Aug 30, 2022

Contents

1

Foreword

fetchmesh is a Python library and a command-line tool for working with RIPE Atlas measurements. More specifically, it focuses on the *anchoring mesh measurements*, the periodic measurements between every pairs of anchors. In the latest versions, the scope of fetchmesh increased to include BGP collectors and PeeringDB IXP data. fetchmesh is a complex tool and I tried my best to keep the code readable, documented and tested. Unfortunately there is always more to do than time allows. The adventurous reader is encouraged to read the source code, and make the appropriate changes if need be.

2 Getting Started

2.1 Requirements

fetchmesh is tested on Linux, macOS and Windows (see *GitHub workflows*). It should work on any platform supported by Python.

fetchmesh requires at-least **Python 3.7** (released in June 2018), notably due to the use of dataclasses.

If need be, a Conda environment with Python 3.7 can be created as follow:

```
conda create -n python37 python=3.7
conda activate python37
python --version # Python 3.7.6
```

2.2 Installation

For now, fetchmesh is a private tool, and as such it is not publised on PyPI, the public Python package index. Instead, fetchmesh can be installed with *pip* directly from GitHub:

```
pip install --upgrade pip
pip install --upgrade git+ssh://git@github.com/SmartMonitoringSchemes/fetchmesh.git
```

or from a local copy:

```
git clone git@github.com:SmartMonitoringSchemes/fetchmesh.git
cd fetchmesh; pip install .
```

To verify the installation:

```
fetchmesh --help
# Usage: fetchmesh [OPTIONS] COMMAND [ARGS]...
```

If you want to make changes to the library, see the *Development* chapter.

2.3 Usage

Table 1: fetchmesh commands overview

Command	Description	Input	Output
describe	Anchoring mesh overview	N/A	N/A
fetch	Fetch measurements results	N/A	One `ndjson` file per measurement
unpack	Split measurement results by pairs	`ndjson` files	One `ndjson` file per origin-destination pair
csv	Convert measurement results to CSV	`ndjson` files	One or more `csv` files, depending on the mode

Use the `--help` flag to get more informations on a command, e.g. `fetchmesh fetch --help`.

2.3.1 Example workflow

A typical workflow involves the following steps:

1. Fetch *raw* measurements results from Atlas API in `ndjson` format (one file per measurement, one measurement result per line).
2. Convert these measurement results in `csv` format, either in `split` mode (one file per origin-destination pair, two columns: timestamp, rtt), or in `merge` mode (one file, one line per origin-destination pair, one column per timestamp).

```
# Fetch IPv4 ping results for 1% of the origin-destination pairs for the 1st of February␣
↪2020,
# excluding self measurements and "reverse" measurements, using 4 concurrent requests.
fetchmesh fetch --af 4 --type ping --no-self --half --sample-pairs 0.01 \
  --start-date 2020-02-01 --stop-date 2020-02-02 --jobs 4

ls -lh ping_v4_1580511600_1580598000/
# total 169M
# -rw-r--r--. 1 maxmouchet maxmouchet 1.1M Mar  3 15:48 ping_v4_1580511600_1580598000_
↪10105927_anchors.ndjson
# -rw-r--r--. 1 maxmouchet maxmouchet 180K Mar  3 15:49 ping_v4_1580511600_1580598000_
↪10206810_anchors.ndjson
# ...

head -n 1 ping_v4_1580511600_1580598000/ping_v4_1580511600_1580598000_1042404_anchors.
↪ndjson
# {"af": 4, "avg": 24.0117783333, "dst_addr": "213.225.160.239", "dst_name": "213.225.
↪160.239", "dup": 0, "from": "193.135.150.58", "fw": 4970, "group_id": 1042404, "lts":␣
↪41, "max": 24.066907, "min": 23.976115, "msm_id": 1042404, "msm_name": "Ping", "prb_id
↪": 6533, "proto": "ICMP", "rcvd": 3, "result": [{"rtt": 24.066907}, {"rtt": 23.976115},
↪ {"rtt": 23.992313}], "sent": 3, "size": 32, "src_addr": "193.135.150.58", "step": 240,
↪ "stored_timestamp": 1580511732, "timestamp": 1580511644, "ttl": 61, "type": "ping"}
```

```
# Generate a single CSV files with all the time series
fetchmesh csv ping --mode merge ping_v4_1580511600_1580598000/*
```

(continues on next page)

(continued from previous page)

```
head -n 2 merge_1583317062.csv
# pair,1580511600,1580511840,1580512080,...
# 1042404_6533,23.976115,24.019383,24.106377,...
```

3 Command Line Reference

This chapter is generated from the *fetchmesh.commands* module source code.

3.1 fetchmesh

fetchmesh is a Python library and a command line utility to facilitate
the use of the RIPE Atlas anchoring mesh measurements.

The documentation and the source code are available at
https://github.com/maxmouchet/fetchmesh.

```
fetchmesh [OPTIONS] COMMAND [ARGS]...
```

Options

`--debug`
: Set the logging level to DEBUG

  Default
  : False

3.1.1 csv

Convert measurement results from ND-JSON to CSV.

```
fetchmesh csv [OPTIONS] COMMAND [ARGS]...
```

ping

Convert ping results from ND-JSON to CSV.

> **Warning:** Results timestamps will be aligned on multiple of 240 seconds (4 minutes), even though they may have been recorded at ± 120 seconds.

Split Mode (*N* files, *T* rows, *2* columns): `timestamp, rtt`
Merge Mode (*N* rows, *T+2* columns): `msm_id, prb_id, from_ip, to_ip, rtt_t1, rtt_t2, rtt_t3, ...`

```
fetchmesh csv ping [OPTIONS] FILES...
```

Options

`--dir <dir>`
: Output directory.

  Default
  : .

`--mode <mode>`
: In split mode one file is created per pair, in merge mode a single file is created.

  Default
  : split

  Options
  : split | merge

Arguments

`FILES`
: Required argument(s)

traceroute

Convert traceroute results from ND-JSON to CSV.

> **Warning:** Late packets are dropped.

Columns: `timestamp, msm_id, prb_id, from_ip, to_ip, paris_id, hop1_1, ..., hop32_3`

```
fetchmesh csv traceroute [OPTIONS] FILES...
```

Options

`--drop-private`
: Remove private IP addresses (v4 and v6)

  Default
  : False

Arguments

`FILES`
: Required argument(s)

3.1.2 describe

Overview of the anchoring mesh at a given date.

```
fetchmesh describe [OPTIONS]
```

Options

`--date <date>`
: Keep only the pairs for which measurements were running on *date*.

  Default
  : now

3.1.3 fetch

Fetch measurement results from the anchoring mesh.

```
fetchmesh fetch [OPTIONS]
```

Options

`--af <af>`
: **Required** Measurement IP address family

`--type <type>`
: **Required** Measurement type

`--start-date <start_date>`
: Results start date (UTC)

  Default
  : last week

`--stop-date <stop_date>`

Results stop date (UTC)

Default
now

`--region <REGION>`

Keep only anchors located in the specified region (e.g. Europe)

`--split <HOURS>`

Split the results files every X hours

`--sample-pairs <sample_pairs>`

If <= 1, fraction of pairs to keep. If > 1, number of pairs to keep.

Default
1.0

`--no-self`

Omit the measurements for which the source and destination probes are the same

Default
False

`--only-self`

Omit the measurements for which the source and destination probes are **not** the same

Default
False

`--half`

Fetch only one measurement out of two for each pair (A<->B or B<->A), useful for RTT measurements

Default
False

`--jobs <N>`

Number of parallel jobs to run

Default
1

`--dir <dir>`

Output directory

`--dry-run`

Don't actually fetch results

Default
False

`--compress`

Compress the results with the Zstandard algorithm

Default
False

`--save-pairs`

Save metadata (*$dir.meta*) and pairs (*$dir.pairs*) for reproducibility.

Default
False

`--load-pairs <load_pairs>`
Load pairs from file (filters will still be applied!)

3.1.4 unpack

Split measurement results by origin-destination pairs.

SRC is a directory containing *.ndjson* files, and *DST* is an output directory.
By default, *DST* is set to *SRC_pairs*.

```
fetchmesh unpack [OPTIONS] SRC [DST]
```

Options

`--af <af>`
Filter measurements IP address family

`--type <type>`
Filter measurements type

`--start-date <start_date>`
Results start date

`--stop-date <stop_date>`
Results stop date

`--jobs <N>`
Number of parallel jobs to run

Default
1

`--mode <mode>`

Default
skip

Options
append | overwrite | skip

Arguments

`SRC`
: Required argument

`DST`
: Optional argument

3.1.5 upgrade

Upgrade fetchmesh to the latest version from GitHub.

Please make sure that your SSH key associated to GitHub is present in your SSH agent.
See https://docs.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent.

```
fetchmesh upgrade [OPTIONS]
```

4 Python Library Reference

4.1 Anchoring Mesh

class fetchmesh.mesh.AnchoringMesh(*data*)

Anchoring Mesh wrapper.

```
from datetime import datetime
from fetchmesh.atlas import MeasurementAF, MeasurementType
from fetchmesh.filters import MeasurementDateFilter, MeasurementTypeFilter
from fetchmesh.mesh import AnchoringMesh

mesh = AnchoringMesh.from_api()

# Keep anchors running between 2019-01-01 and 2019-01-02
mesh = mesh.filter(MeasurementDateFilter.running(
    datetime(2019,1,1), datetime(2019,1,2)
))

# Keep anchors participating in IPv6 ping measurements
mesh = mesh.filter(MeasurementTypeFilter(
    MeasurementAF.IPv6, MeasurementType.Ping
))
```

classmethod from_api(*client=<fetchmesh.atlas.client.AtlasClient object>*)

Instantiate the AnchoringMesh from `anchor-measurements/?include=target,measurement`.

anchors

Anchoring mesh anchors.

measurements

Anchoring mesh measurements.

class fetchmesh.mesh.AnchoringMeshPairs(*pairs*)

Anchoring Mesh pairs container.

```
from fetchmesh.filters import PairSampler, SelfPairFilter
from fetchmesh.mesh import AnchoringMesh
```

(continues on next page)

(continued from previous page)

```
mesh = AnchoringMesh.from_api()
pairs = mesh.pairs

# Keep 10% of the pairs
pairs = pairs.filter(PairSampler(0.1))

# Remove "self" measurements
pairs = pairs.filter(SelfPairFilter())
```

4.2 Filters

4.2.1 Anchors

class fetchmesh.filters.anchor.AnchorFilter

class fetchmesh.filters.anchor.AnchorPairFilter

class fetchmesh.filters.anchor.AnchorRegionFilter(*region: str*)

class fetchmesh.filters.anchor.HalfPairFilter

Keep only one of the two measurement for each pair (i.e. A->B or B->A, but not both).

```
filter = HalfPairFilter()
```

class fetchmesh.filters.anchor.PairRegionSampler(*k: Union[float, int]*, *regions: List[str]*)

class fetchmesh.filters.anchor.PairSampler(*k: Union[float, int]*)

Take a random sample of the anchor pairs.

```
# Keep 75% of the pairs
filter = PairSampler(0.75)

# Keep 200 pairs
filter = PairSampler(200)
```

k: `Union[float, int]`

If `k` is a float between 0.0 and 1.0, it will sample `k*len(data)` pairs.
If `k` is an integer greater than or equal to 0, it will sample `k` pairs.

class fetchmesh.filters.anchor.SelfPairFilter(*reverse: bool = False*)

Drop (or keep only) measurements where the origin anchor is equal to the destination anchor.

```
# Drop self pairs
filter = SelfPairFilter()

# Keep only self pairs
filter = SelfPairFilter(reverse=True)
```

reverse: bool = False

reverse = False: drop self pairs
reverse = True: keep only self pairs

4.2.2 Measurements

class fetchmesh.filters.measurement.MeasurementDateFilter(*start_date_gte: Optional[datetime] = None*, *start_date_lte: Optional[datetime] = None*, *stop_date_gte: Optional[datetime] = None*, *stop_date_lte: Optional[datetime] = None*)

Keep measurements where start_date_gte <= start_date <= start_date_lte and stop_date_gte <= stop_date <= stop_date_lte.

class fetchmesh.filters.measurement.MeasurementFilter

class fetchmesh.filters.measurement.MeasurementTypeFilter(*af:* fetchmesh.atlas.objects.MeasurementAF, *type:* fetchmesh.atlas.objects.MeasurementType)

4.2.3 Records

class fetchmesh.filters.record.ProbeIDRecordFilter(*probe_ids: Set[int]*)

class fetchmesh.filters.record.RecordFilter

class fetchmesh.filters.record.RecordTypeFilter(*type:* fetchmesh.atlas.objects.MeasurementType)

class fetchmesh.filters.record.SelfRecordFilter

4.3 Transformers

4.3.1 Records

class fetchmesh.transformers.record.PingMinimumTransformer

class fetchmesh.transformers.record.RecordTransformer

class fetchmesh.transformers.record.TracerouteFlatIPTransformer(*as_set: bool = False*, *drop_dup: bool = False*, *drop_late: bool = False*, *drop_private: bool = False*, *extras_fields: Tuple[str, ...] = ()*, *insert_none: bool = True*)

`as_set: bool = False`

Return each hop as a set instead of a list of addresses.

`drop_dup: bool = False`

Drop duplicate results:

```
{'dup': True, 'from': '203.181.249.93', 'rtt': 280.552, 'size': 28, 'ttl': 231}
 ^^^^^^^^^^^
```

`drop_late: bool = False`

Drop late results:

```
{"from":"4.68.72.66","late":2,"size":68,"ttl":56}
                      ^^^^^^^^
```

`drop_private: bool = False`

Drop private IP addresses:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16
- fd00::/8

`extras_fields: Tuple[str, ...] = ()`

List of additional response fields to include (e.g. *asn*, *ix* ...)

`insert_none: bool = True`

Insert *None* if the *from* (or extra) field is absent.

`class fetchmesh.transformers.record.TracerouteMapASNTransformer(`*tree: radix.Radix*`)`

`class fetchmesh.transformers.record.TracerouteMapIXTransformer(`*tree: radix.Radix*`)`

4.4 RIPE Atlas Objects

`class fetchmesh.atlas.objects.AtlasAnchor(`*id: int*, *probe_id: int*, *fqdn: str*, *country: Country*, *as_v4: Optional[int]*, *as_v6: Optional[int]*`)`

A RIPE Atlas anchor.

`as_v4: Optional[int]`

Anchor IPv4 autonomous system number (if applicable).

`as_v6: Optional[int]`

Anchor IPv6 autonomous system number (if applicable).

`country: Country`

Anchor country.

`fqdn: str`

Anchor fully-qualified domain name.

id: int
Anchor ID.

probe_id: int
Anchor **probe** ID.

class fetchmesh.atlas.objects.AtlasMeasurement(*id: int*, *af:* MeasurementAF, *type:* MeasurementType, *status:* MeasurementStatus, *start_date: Optional[datetime]*, *stop_date: Optional[datetime]*, *description: str*, *tags: Tuple[str, ...]*)

A RIPE Atlas measurement.

classmethod from_dict(*d: dict*)
Build from a dict following the Atlas API format.

to_dict()
Convert to a dict following the Atlas API format.

af: *MeasurementAF*
Measurement IP address family.

anchor_name
Target anchor name, extracted from the tags or from the description.

anchor_probe
Target anchor probe ID, extracted from the tags.

description: str
Measurement description.

id: int
Measurement ID.

is_anchoring
Whether the measurement is part of the *anchoring mesh* or the *anchoring probes* measurements or not.

is_anchoring_mesh
Whether the measurement is part of the *anchoring mesh* measurements.

is_anchoring_probes
Whether the measurement is part of the *anchoring probes* measurements.

start_date: Optional[datetime]
Measurement start date (if started).

status: *MeasurementStatus*
Measurement status.

stop_date: Optional[datetime]
Measurement stop date (if stopped).

tags: Tuple[str, ...]
Measurement tags.

type: *MeasurementType*
Measurement type.

class fetchmesh.atlas.objects.MeasurementAF(*value*)

An enumeration.

IPv4 = 4

IPv6 = 6

class fetchmesh.atlas.objects.MeasurementStatus(*value*)

An enumeration.

Archived = 8

Failed = 7

ForcedToStop = 5

NoSuitableProbes = 6

Ongoing = 2

Scheduled = 1

Specified = 0

Stopped = 4

class fetchmesh.atlas.objects.MeasurementType(*value*)

An enumeration.

DNS = 'dns'

HTTP = 'http'

NTP = 'ntp'

Ping = 'ping'

SSL = 'sslcert'

Traceroute = 'traceroute'

WiFi = 'wifi'

4.5 Autonomous Systems

4.5.1 BGP Collectors

class fetchmesh.bgp.collectors.Collector

Base class for Remote Route Controllers (RRCs).

```
>>> from datetime import datetime
>>> from fetchmesh.bgp import Collector
>>> collector = Collector.from_fqdn("route-views2.routeviews.org")
>>> collector.table_name(datetime(2020, 1, 1, 8))
'rib.20200101.0800.bz2'
>>> collector.table_url(datetime(2020, 1, 1, 8))
'http://archive.routeviews.org/bgpdata/2020.01/RIBS/rib.20200101.0800.bz2'
```

`download_rib`(*t: datetime*, *directory: Union[Path, str]*, *name: Optional[str] = None*) → Path
Download the Routing Information Base (RIB) at time *t* in *directory*.

`abstract table_name`(*t: datetime*) → str
Return the file name for the RIB at time *t*.

`abstract table_url`(*t: datetime*) → str
Return the URL for the RIB at time *t*.

`class fetchmesh.bgp.collectors.RISCollector`(*name: str*, *extension: str = 'gz'*)
A Remote Route Collector (RRC) from the RIPE Routing Information Service (RIS).

```
from fetchmesh.bgp import RISCollector
collector = RISCollector("rrc00")
```

`download_rib`(*t: datetime*, *directory: Union[Path, str]*, *name: Optional[str] = None*) → Path
Download the Routing Information Base (RIB) at time *t* in *directory*.

`table_name`(*t: datetime*) → str
Return the file name for the RIB at time *t*.

`table_url`(*t: datetime*) → str
Return the URL for the RIB at time *t*.

`class fetchmesh.bgp.collectors.RouteViewsCollector`(*name: str*, *extension: str = 'bz2'*)
A Remote Route Collector (RRC) from the University of Oregon Route Views Project.

```
from fetchmesh.bgp import RISCollector
collector = RouteViewsCollector("route-views2")
```

`download_rib`(*t: datetime*, *directory: Union[Path, str]*, *name: Optional[str] = None*) → Path
Download the Routing Information Base (RIB) at time *t* in *directory*.

`table_name`(*t: datetime*) → str
Return the file name for the RIB at time *t*.

`table_url`(*t: datetime*) → str
Return the URL for the RIB at time *t*.

4.6 Internet Exchanges

`class fetchmesh.peeringdb.PeeringDB`(*objects: List[Object]*)
An object-oriented interface to PeeringDB.

```
from fetchmesh.peeringdb import PeeringDB
peeringdb = PeeringDB.from_api()

peeringdb.objects[0]
# Object(ix=IX(id=1, name='Equinix Ashburn'), prefixes=[
#     Prefix(id=2, ixlan_id=1, prefix='2001:504:0:2::/64'),
#     Prefix(id=386, ixlan_id=1, prefix='206.126.236.0/22')
# ])
```

(continues on next page)

(continued from previous page)

```
ixtree = peeringdb.radix_tree()
ixtree.search_best("37.49.236.1").data["ix"]
# IX(id=359, name='France-IX Paris')
```

classmethod from_api(*client=<fetchmesh.peeringdb.client.PeeringDBClient object>*) → *PeeringDB*

Load PeeringDB from the PeeringDB API.

radix_tree() → Radix

Return a radix tree (from the py-radix library) for fast IP-IX lookups.

4.7 Input / Output

class fetchmesh.io.AtlasRecordsReader(*file: ~pathlib.Path*, *filters: ~typing.List[~fetchmesh.filters.abstract.StreamFilter[dict]] = <factory>*, *transformers: ~typing.List[~fetchmesh.transformers.record.RecordTransformer] = <factory>*)

Read Atlas results in ND-JSON format. Automatically handles compressed files.

```
from fetchmesh.io import AtlasRecordsReader

# From a single file.
with AtlasRecordsReader("results.ndjson") as r:
    for record in r:
        print(record)

# From multiple files.
r = AtlasRecordsReader.all(["results1.ndjson", "results2.ndjson"])
for record in r:
    print(record)

# From a glob pattern.
r = AtlasRecordsReader.glob("mydir/", "*.ndjson")
for record in r:
    print(record)
```

classmethod all(*files*, ***kwargs*)

Read multiple files.

classmethod glob(*path*, *pattern*, ***kwargs*)

Read multiple files from a glob pattern.

file: Path

Input file path.

filters: List[StreamFilter[dict]]

List of filters to apply when reading the records.

transformers: List[*RecordTransformer*]

List of transformers to apply when reading the records.

class fetchmesh.io.AtlasRecordsWriter(*file: ~pathlib.Path*, *filters: ~typing.List[~fetchmesh.filters.abstract.StreamFilter[dict]] = <factory>*, *append: bool = False*, *log: bool = False*, *compression: bool = False*)

Write Atlas results in ND-JSON format.

```
from fetchmesh.io import AtlasRecordsWriter
with AtlasRecordsWriter("results.ndjson") as w:
    w.write({"msm_id": 1001, "prb_id": 1, "...": "..."})
```

write(*record: dict*)

Write a single record.

writeall(*records: Iterable[dict]*)

Write all the records.

append: bool = False

Whether to create a new file, or to append the records to an existing file. If append is set to false, and the output file already exists, it will be deleted. When append is set to false, the output file will be deleted if an exception happens.

compression: bool = False

Compresse the records using zstandard. We use the one-shot compression API and write one frame per record. This results in larger files than a single frame for all the records, but it allows us to build an index and make the file seekable. We use a pre-built dictionary (see *dictionary*) to reduce the size of the compressed records.

file: Path

Output file path.

filters: List[StreamFilter[dict]]

List of filters to apply before writing the records.

log: bool = False

Record the size (in bytes) of each record. See *LogEntry*.

property log_file: Path

Path to the (optional) log file.

fetchmesh.io.LogEntry = <_struct.Struct object>

Binary structure containing the size, the measurement ID, and the probe ID for a record. This is useful for indexing the content of a result file without decompressing and parsing the JSON. If the file is compressed, the size is the size of the zstandard frame. The fields are unsigned longs of 8 bytes each : *size_bytes*, *msm_id*, *prb_id*.

fetchmesh.io.dictionary = PosixPath('/home/runner/work/fetchmesh/fetchmesh/fetchmesh/mocks/dictionary')

Path to the zstandard dictionary used to compress the records. Useful to decompress manually the records.

4.8 Metadata

fetchmesh *metadata* classes contains informations about the content of a file. They can be (de)serialized from/to a filename.

`class fetchmesh.meta.AtlasResultsMeta(`*af:* MeasurementAF, *type:* MeasurementType, *msm_id:* *int*, *start_date:* *datetime*, *stop_date:* *datetime*, *compressed:* *bool*`)`

Measurement results file metadata. A results file contain results from a single measurement, with potentially multiple sources.

`class fetchmesh.meta.IPASNMeta(`*collector:* Collector, *datetime:* *datetime*`)`

Metadata for IPASN files generated by *pyasn*.

`class fetchmesh.meta.RIBMeta(`*collector:* Collector, *datetime:* *datetime*`)`

Metadata for RIB files downloads from BGP collectors.

`fetchmesh.meta.meta_from_filename(`*name:* *Union[Path, str]*`)`

Try to find the metadata corresponding to *name*, return None otherwise.

5 Recipes

5.1 Alias resolution with kapar

https://www.caida.org/tools/measurement/kapar/

5.1.1 Install kapar

```
# Download and compile patched version (CAIDA version has a bug and doesn't compile)
git clone https://github.com/maxmouchet/kapar.git
cd kapar
./configure
make

# Download bogons files
wget https://www.team-cymru.org/Services/Bogons/bogon-bn-agg.txt

# Verification
./kapar/kapar --help
```

5.1.2 Fetch some traceroutes

```
fetchmesh fetch --type traceroute --af 4 --start-date "2020-07-20 12:00" --stop-date
↪"2020-07-20 12:30" --sample-pairs 0.01 --jobs 4
```

5.1.3 Convert traceroutes to kapar input format

Input	Output
RIPE Atlas traceroute results in `ndjson` format	`paths.txt` in kapar format

```
from fetchmesh.io import AtlasRecordsReader
from fetchmesh.transformers import TracerouteFlatIPTransformer

def to_kapar_format(record, include_origin=True):
    lines = [
        # Only '#' is strictly necessary on this line. We include the metadata for␣
↪reference, if needed.
        f"# timestamp={record['timestamp']} measurement={record['msm_id']} probe=
↪{record['prb_id']}"
    ]
    if include_origin:
        lines.append(record["from"])
    for replies in record["hops"]:
        # Insert 0.0.0.0 in place of missing values (will be filtered by kapar)
        lines.append(replies[0] or "0.0.0.0")
    return "\n".join(lines)

# ------------------------------------------------------------------------------------
↪-----------------------
# ! Edit this line with the correct path to the downloaded traceroutes (directly in␣
↪`ndjson` format, not `csv`).
path = "/home/maxmouchet/Clones/github.com/maxmouchet/fetchmesh/traceroute_v4_1595289600_
↪1595334480/"
# ------------------------------------------------------------------------------------
↪-----------------------

transformers = [TracerouteFlatIPTransformer(drop_dup=True, drop_late=True, drop_
↪private=True)]
rdr = AtlasRecordsReader.glob(path, "*.ndjson", transformers=transformers)

# This will take some time (~30s)
output = [to_kapar_format(record) for record in rdr]

with open("paths.txt", "w") as f:
    f.write("\n".join(output))
```

5.1.4 Perform inference with kapar

Input	Output
`bogon-bn-agg.txt`, `paths.txt`	`kapar.aliases`, `kapar.ifaces`, `kapar.links`, `kapar.subnets`

```
./kapar/kapar -o alis -B bogon-bn-agg.txt -P paths.txt
```

5.1.5 Check the results

```
head kapar.aliases
# # found 8182 nodes, containing 12614 interfaces (0 redundant (omitted), 207 anonymous,␣
↪12407 named).
# node N1:  95.59.172.33 87.245.238.25 178.210.33.75 89.218.74.77 92.46.59.234
# node N2:  172.253.65.176 216.239.58.255 172.253.70.202 142.250.61.66 172.253.70.204
# ...

head kapar.links
# # found 9892 links, containing 22209 interfaces (9735 implicit, 0 redundant (omitted),␣
↪207 anonymous, 12267 named).
# link L1:  N5:129.143.66.64 N4763:129.143.66.65
# link L2:  N1159:213.91.165.184 N6916:213.91.165.185
# ...
```

6 Development

fetchmesh is developed on GitHub in the SmartMonitoringSchemes organization. It uses poetry for dependency management and packaging.

6.1 Workflow

A typical development workflow is as follows:

```
git clone git@github.com:maxmouchet/fetchmesh
cd fetchmesh/

# `poetry install` is required only once.
# If the pyproject.toml file is modified, run `poetry update` instead.
poetry install

# Setup pre-commit, required only once.
poetry run pre-commit install

# Run fetchmesh in poetry virtualenv, make sure it works.
poetry run fetchmesh

# Make some code changes
# [...]

# Run fetchmesh again to test your changes
poetry run fetchmesh ...

# Run the test suite
poetry run pytest

# Review and commit the changes
git diff
git add ...
git commit -m '...'
```

(continues on next page)

(continued from previous page)

```
# Fix pre-commit warnings if needed, and go back to the previous step.
# [...]

# Push the chnages
git push
```

6.2 Documentation

This documentation is built using sphinx. To build the documentation locally run the following:

```
poetry run make -C docs/ html
# The website will be found in docs/_build/html/

poetry run make -C docs/ latexpdf
# The PDF will be found at docs/_build/latex/fetchmesh.pdf
```

6.3 Tools

Tool	Usage	Command
black	Code formatting	`poetry run pre-commit run --all-files`
isort	Import sorting	`poetry run pre-commit run --all-files`
mypy	Static typing	`poetry run pre-commit run --all-files`
pylint	Linting	`poetry run pre-commit run --all-files`
pytest	Unit tests	`poetry run pytest`

6.4 Release

To create a release:

```
poetry version x.x.x
git commit -m 'Version x.x.x'
git tag vx.x.x
git push && git push --tags
```

6.5 GitHub workflows

Two GitHub Workflows are defined:

.github/workflows/ci.yml
: Run the tests on Linux, and check that the package installs correctly on Linux, macOS and Windows.

.github/workflows/documentation.yml
: Build this documentation, and upload it to the *gh-pages* branch.

Python Module Index

f

Index

Symbols

A

C

D

N

O

P

R

S

T

W