

asn_list.py — Global ASN Discovery & Initialization

Full Technical Documentation (Literal 100% Code-Accurate to the Provided Script)

1. Purpose & Role in the Platform

`asn_list.py` is the platform's foundational **ASN discovery and persistence** component.

Its purpose is to maintain a continuously updated SQLite table containing **ASNs actually observed** in real BGP data from:

- **RIPE RIS Live** (real-time UPDATE stream)
- **RouteViews2 RIB snapshots** (periodic full-table snapshots)

The script does **not** attempt to enumerate all theoretically allocated ASNs; it only stores ASNs that appear in the above data sources.

Responsibilities

The script:

- passively collects ASNs from live BGP UPDATE streams (RIPE RIS Live)
- periodically imports ASNs from RouteViews2 RIB snapshots
- deduplicates ASNs in-memory during runtime
- stores ASNs into a single SQLite table (`asn_data`)
- assigns a placeholder name `AS<asn>` for every stored ASN

This global ASN universe is intended as an input to higher-level modules such as:

- ROV enforcement analysis

- prefix collectors
 - propagation models
 - ML feature generation
 - ASN-level security scoring
-

2. High-Level Behavior

`asn_list.py` combines two independent ASN discovery pipelines that feed into a shared deduplication + insertion mechanism:

2.1 Live Collection (RIPE RIS Live)

- connects to RIPE RIS Live streaming endpoints (SSE preferred; JSON fallback only if SSE does not start)
- processes only BGP UPDATE messages
- extracts ASNs from the `path` field (AS_PATH-like list)
- enqueues either:
 - only the origin ASN (default), or
 - all ASNs in the path (`ENQUEUE_FULL_PATH=True`)
- persists newly discovered ASNs to the database via a writer queue/thread

2.2 Periodic Collection (RouteViews2 RIB)

- runs once on startup, then repeats every `RIB_REFRESH_HOURS` (default 12 hours)
- locates the newest RouteViews2 RIB snapshot URL (searching current month, then previous two months)
- downloads the `.bz2` RIB file to a temporary directory (with retry + optional resume)

- parses the RIB via `bgpdump -M`
 - extracts ASNs by finding numeric tokens in the AS_PATH field produced by `bgpdump`
 - processes the RIB stream in 10,000-line chunks using multiprocessing
 - inserts all extracted ASNs through the same deduplication/enqueue pipeline
-

3. What the Script Guarantees (Runtime Semantics)

Within a single process lifetime, the script ensures that:

- any ASN that is:
 - observed in RIPE RIS Live UPDATE `path`, or
 - extracted from the RouteViews2 RIB parsing pipeline and is a **positive integer** will be:
 - deduplicated in-memory (per runtime), and
 - inserted/upserted into the database with a placeholder name `AS<asn>`

Important clarification:

- Deduplication (`SEEN_ASNS`) is **in-memory** and only applies during the runtime of the process instance.
 - The database is updated via **idempotent UPSERT**, so re-running the script will safely update/overwrite placeholder names.
-

4. Database Contract

4.1 Database Path

The SQLite DB file is:

- `<PROJECT_ROOT>/database/asn_data.db`

4.2 Table Definition

The script creates exactly one table:

```
CREATE TABLE IF NOT EXISTS asn_data (  
    asn INTEGER PRIMARY KEY,  
    name TEXT  
);
```

No other tables are created or modified.

4.3 Insert / Update Behavior (UPSERT)

All inserts use idempotent UPSERT semantics:

```
INSERT INTO asn_data (asn, name) VALUES (?, ?)  
ON CONFLICT(asn) DO UPDATE SET name = excluded.name;
```

4.4 Writer Behavior

- writes are batched up to **1000 rows** per flush
- batches are written using `executemany()` inside a transaction context (`with DB_CONN:`)
- on batch failure:
 - falls back to per-row UPSERT attempts

5. RIPE RIS Live Processing Pipeline

5.1 Stream Endpoints

Preferred SSE endpoint:

- `https://ris-live.ripe.net/v1/stream/?format=sse&client=bgp-collector-ml`

JSON endpoint (used only if SSE does not start):

- `https://ris-live.ripe.net/v1/stream/?format=json&client=bgp-collector-ml`

5.2 Subscription Headers

Both stream modes send:

- `X-RIS-Subscribe: {"type":"UPDATE"}`

along with appropriate `Accept`, `Origin`, and `User-Agent` headers.

5.3 Message Filtering

The script processes:

- only objects where:
 - top-level `type == "ris_message"`
 - inner `data.type == "UPDATE"`

It logs and ignores:

- `type == "ris_error"` (logged as an error)
- any non-UPDATE messages

5.4 UPDATE Parsing Logic

For each UPDATE:

- reads `data["path"]`
- ignores the message if:
 - `path` is not a list, or
 - `path` is empty
- extracts ASNs from `path` elements that are:
 - `int`, or
 - a digit-only string (`str.isdigit()`)

Origin ASN derivation:

- origin ASN is taken as the **last element of `path`**, if that last element is numeric (int or digit-string)

5.5 Enqueue Semantics

- if `ENQUEUE_FULL_PATH == True`:
 - enqueues **all extracted ASNs** from the path
- else (default):
 - enqueues **only the origin ASN**, if it was successfully derived

Note:

- `peer` / `peer_asn` fields are not used by this script.

5.6 Idle Timeout & Reconnection

During streaming:

- if no stream lines are received for `STREAM_IDLE_MAX_SEC` seconds (default 120s):

- the stream loop logs an idle warning and exits the current stream attempt

Retry behavior:

- after each stream attempt ends (for any reason), `ris_stream_loop()` waits using a **backoff value** that:
 - starts at `STREAM_BACKOFF_START` (default 2)
 - increases by doubling each loop iteration
 - is capped at `STREAM_BACKOFF_MAX` (default 60)
 - includes a small random jitter
- the backoff value is **not reset** after successful streaming.

JSON fallback detail (code-accurate):

- JSON streaming is attempted only when `_sse_stream_once()` returns `False` (i.e., SSE did not start).
- If SSE starts and later ends (idle/disconnect), the next loop iteration attempts SSE again.

6. RouteViews2 RIB Processing Pipeline

6.1 Scheduling

- A RouteViews RIB import is attempted once at startup (inside `run_service_once()`).
- A background refresher thread then triggers additional imports every:
 - `RIB_REFRESH_HOURS * 3600` seconds (default: 12 hours).

6.2 Preconditions

If `bgpdump` is not installed (`shutil.which("bgpdump")` returns false):

- the script logs:
 - `"bgpdump is not installed – skipping RIB snapshot."`
- and does not attempt RIB processing.

6.3 Finding the Latest RIB URL

The script searches for the latest RIB in:

- current month
- then previous month
- then two months back

For each candidate month directory:

- `<ROUTEVIEWS_BASE>/<YYYY.MM>/RIBS/`

It fetches the HTML and extracts RIB filenames matching:

- `rib.(YYYYMMDD.HHMM).bz2`

It selects the lexicographically greatest timestamp (sorted list, last element).

6.4 Download Behavior

The RIB is downloaded into a temporary directory:

- `tempfile.TemporaryDirectory()`

Download implementation (`download_file_with_retry`):

- uses a **new `requests.Session()`** (separate from the global `SESSION`)
- mounts the same `adapter` (same pool + Retry config)

- implements an explicit retry loop (`max_retries`, default 5)
- supports resuming partial downloads by sending a `Range` header if a partial file exists
- streams response content in chunks (default 256 KiB)

If all retries fail:

- it raises an exception.

6.5 Decompression & bgpdump Streaming

The script produces a stream of `bgpdump -M` lines using one of two modes:

- If `lbzip2` or `pbzip2` exists:
 - runs `<lbzip2|pbzip2> -dc <rib.bz2>` and pipes into:
 - `bgpdump -M -`
- else:
 - runs `bgpdump -M <rib.bz2>` directly

6.6 Parsing Logic (Exact to Code)

For each `bgpdump -M` output line:

- splits by ' | '
- requires at least 7 fields
- takes `parts[6]` as the AS_PATH field
- extracts numeric tokens using regex:
 - `\b\d+\b`
- converts tokens to integers

- keeps only values `> 0`
- aggregates all such ASNs into a set

Note on “origin”:

- `_parse_bgpdump_line_fast()` also computes `origin` as the last numeric token encountered in the `AS_PATH` field.
- However, the multiprocessing chunk worker (`_worker_parse_chunk`) **does not use the origin value**; it only aggregates the ASN set.
- Therefore, the effective output of RIB parsing is **the set of all ASNs found in `AS_PATH` numeric tokens**.

6.7 Multiprocessing Chunking

- reads the bgpdump stream line-by-line
- groups lines into in-memory chunks of:
 - `CHUNK = 10000`
- submits each chunk to a multiprocessing pool:

Pool sizing:

- `cpu = max(2, mp.cpu_count() - 1)`

Each worker returns a `set` of ASNs for its chunk, and the parent aggregates them all into `aggregated_seen`.

6.8 Insertion Consistency

After parsing:

- all ASNs in `aggregated_seen` are passed to:

```
enqueue_asns(aggregated_seen, 'RouteViews RIB')
```

This ensures the same deduplication rules and database writing pipeline as the RIS Live path.

7. Deduplication & ASN Queueing

7.1 Global Deduplication Set

```
SEEN_ASNS = set()
SEEN_LOCK = threading.Lock()
```

Deduplication rule:

- an ASN is enqueued only if:
 - it is an `int`
 - `asn > 0`
 - it is not already present in `SEEN_ASNS`

All checks and updates happen under `SEEN_LOCK`.

If new ASNs are enqueued, the script logs:

- count added from the source tag
- total distinct in `SEEN_ASNS`

7.2 ASN_QUEUE

New ASNs are placed into:

- `ASN_QUEUE = queue.Queue()`

Workers consume from this queue.

8. Worker & “Enrichment” Execution Model (As Implemented)

8.1 Worker Threads

- `WORKER_THREADS` daemon threads run `worker(stop_event)`
- each worker accumulates ASNs into an in-memory batch of size:
 - `BATCH_SIZE = 128`
- when the batch is full (or on timeouts), it submits:

```
ENRICH_EXECUTOR.submit(process_batch, batch)
```

8.2 What “process_asn” Actually Does

Despite the naming, there is no external enrichment.

For each ASN:

- constructs `{'name': f'AS{asn}'}`
- calls `save_asn(asn, data)`
- which enqueues `(asn, name)` into `WRITE_QUEUE`

So the only “enrichment” is placeholder name generation.

9. HTTP Behavior

9.1 Global HTTP Session (`SESSION`)

Used by `_http_get()` for:

- RIS Live SSE/JSON streams

- RouteViews directory listing pages (HTML)

Key properties:

- uses `HTTPAdapter` with pooling:
 - `pool_connections=POOL_SIZE`
 - `pool_maxsize=POOL_SIZE`
- uses `urllib3 Retry(total=HTTP_RETRIES, backoff_factor=0.5, status_forcelist=...)`
- disables proxy environment usage if `DISABLE_PROXY=True`:
 - `SESSION.trust_env = False`
- sets `User-Agent: bgp-collector-m1/1.0`

9.2 Forced IPv4 Resolution (RIS Live)

When `FORCE_IPV4=True`, the script temporarily monkey-patches:

- `socket.getaddrinfo`

to force IPv4 resolution during RIS stream calls only.

9.3 RIB Downloads Use a Separate Session

RIB downloads do not use the global `SESSION`.

They use a separate per-download session in `download_file_with_retry()`.

10. Database Writer Pipeline

10.1 Writer Thread

A dedicated daemon thread runs `db_writer_loop()`:

- consumes `(asn, name)` from `WRITE_QUEUE`
- batches up to 1000 items
- flushes when:
 - batch reaches 1000, or
 - shutdown is requested (`WRITER_STOP` set)

10.2 Flush Timing

Writer loop uses:

- `INTERVAL_SEC = 0.25`

It attempts to fetch queue items with timeout and sleeps when not flushing.

11. Status Monitor

A daemon thread runs `status_monitor(stop_event)` and every:

- `STATS_EVERY_SEC` seconds (default 60)

logs:

- total distinct ASNs (`len(SEEN_ASNS)`)
- `ASN_QUEUE.qsize()` (or `-1` if unsupported)
- pending work size from `ENRICH_EXECUTOR._work_queue.qsize()` (best-effort; otherwise `-1`)
- idle time since last enqueue (`LAST_NEW_ASN_TS`)

12. Script Lifecycle, Shutdown, and Restart

12.1 run_service_once()

Startup flow:

1. `init_db()`
2. installs SIGINT/SIGTERM handlers that set:
 - `_SHUTDOWN_REQUESTED`
 - `stop_event`
3. starts writer thread
4. starts worker threads
5. attempts initial RouteViews import (`routeviews_once_and_fill_counts()`)
6. sets `LAST_NEW_ASN_TS = time.time()`
7. starts:
 - RIS stream thread
 - RIB refresher thread
 - status monitor thread
8. blocks until `stop_event` is set

Shutdown flow (`finally` block):

- pushes `None` sentinel into `ASN_QUEUE WORKER_THREADS` times
- joins workers (timeout 5s)

- shuts down `ENRICH_EXECUTOR` (`wait=True`)
- sets `WRITER_STOP`
- joins writer thread (timeout 5s)
- logs final distinct count

12.2 main() and STAY_ALIVE

If `STAY_ALIVE=False`:

- runs `run_service_once()` once and exits.

If `STAY_ALIVE=True`:

- runs `run_service_once()` in a loop
 - on crash/exception:
 - sleeps with a restart backoff
 - doubles restart backoff up to `RESTART_BACKOFF_S[1]`
 - exits loop if `_SHUTDOWN_REQUESTED` is set, or after a normal termination.
-

13. Configuration Parameters

Used by the Code

- `WORKER_THREADS`
- `ASYNC_ENRICH_THREADS`
- `HTTP_TIMEOUT`

- HTTP_RETRIES
- POOL_SIZE
- STREAM_IDLE_MAX_SEC
- STREAM_BACKOFF_START
- STREAM_BACKOFF_MAX
- FORCE_IPV4
- DISABLE_PROXY
- ENQUEUE_FULL_PATH
- RIB_REFRESH_HOURS
- STATS_EVERY_SEC
- STAY_ALIVE
- RESTART_BACKOFF_S

Defined but Not Used

- BGPDUMP_THREADS (defined, but not referenced anywhere else)

14. Limitations (Exactly True for the Code)

- Only ASNs observed in:
 - RIS Live UPDATE `path`, or

- numeric AS_PATH tokens extracted from `bgpdump -M` output will be stored.
 - No ASN name enrichment is performed; names are always `AS<asn>`.
 - If `bgpdump` is missing, RouteViews RIB ingestion is skipped.
 - RouteViews “latest RIB” selection depends on parsing HTML directory listings.
 - In-memory deduplication (`SEEN_ASNS`) does not persist across process restarts (DB UPSERT provides safe re-runs).
 - Stream retry backoff increases over time and is capped; it is not reset automatically.
-

15. Integration Points

This module provides the platform’s base “ASN universe” used by:

- ROV enforcement engines
- prefix collectors
- propagation/reachability modeling
- ML feature generation
- ASN-level scoring pipelines