

asn_aspath_len.py — AS Path Length Metric (Full Technical Documentation)

1. Purpose & Role in the Platform

asn_aspath_len.py computes real-time AS-path length statistics per origin ASN by consuming BGP UPDATE messages from RIPE RIS Live.

This metric provides a direct, empirical measurement of how “far” an ASN lies in the global AS topology, based on the hop-count of AS_PATH attributes observed across many independent vantage points.

This matters because:

- Shorter AS paths typically indicate well-connected, central ASNs whose announcements propagate widely.
- Longer AS-paths are typically associated with ASNs that are topologically peripheral or have limited upstream/peering connectivity.
- Path length does not automatically imply instability, but longer paths generally correlate with lower centrality and lower propagation influence in the global routing system.

AS-path length is an operational indicator of:

- Vulnerability (how likely a forged origin is to be accepted)
- propagation potential (how far an attack can spread)
- routing influence (how much visibility an ASN has in the global control-plane)

Thus, this metric is a core feature in the platform’s security, propagation, and influence modeling.

2. High-Level Behavior

During each 120-second real-time measurement window, the script:

1. Opens a WebSocket streaming session to:
 - `wss://ris-live.ripe.net/v1/ws/?client=aspath-now-main`
2. Subscribes to UPDATE messages with `announcements` required:
 - Subscribe message:

```
{"type": "ris_subscribe", "data": {"type": "UPDATE", "require": "announcements"}}
```
3. For each received RIS frame:
 - The script processes only frames of type `ris_message`.
 - From each `ris_message`, it processes only `data` payloads where:
 - `data["type"] == "UPDATE"`, and
 - `path` is present/non-empty, and
 - `announcements` is present/non-empty (otherwise the update is ignored).
4. For each valid UPDATE:
 - Extracts the AS_PATH (`path = data.get("path")`)
 - Computes hop count via `count_aspath_hops(path)`:
 - each integer ASN counts as 1 hop
 - each AS-SET represented as a Python `list` counts as 1 hop
 - other element types do not contribute
 - Identifies the origin ASN:
 - `origin = last element of AS_PATH only if it is an int`

- otherwise `origin = None`
 - Identifies the vantage point id:
 - `vp_id = str(peer_asn) if present else str(peer)`
 - (the vantage id is stored as a string)
 - Determines AFI from prefix format:
 - `v6` if prefix contains “.”
 - else `v4`
 - For each announced prefix in `announcements[*].prefixes[*]`:
 - stores `(origin, hops, afi)` into `self.current[(vp_id, prefix)]`
even if `origin` is `None`
 - **only if `origin` is an integer:**
 - appends hop count into:
 - `lengths_per_peer[origin][afi][vp_id].append(hops)`
 - recomputes the full statistical row for that origin ASN
 - prints it and writes it to DB (details below)
5. It continues streaming until the 120s window ends (or ends early on some WS errors).

Important runtime behavior (code-accurate):

- The script runs continuously in an internal infinite loop (`while True`).
- Each cycle runs a 120-second live sampling round, then sleeps 30 minutes, then repeats.
- This produces repeated, high-resolution “now snapshots” of AS-path lengths per ASN.

3. Metrics Produced

3.1 Per-AFI Statistical Metrics (stored in `asn_aspath_len_now`)

For every ASN, separately for IPv4 and IPv6:

Column	Meaning
<code>as_path_len_p50_v4</code>	Median AS-path length across v4 vantage-point medians (median-of-medians)
<code>as_path_len_p90_v4</code>	90th percentile of v4 vantage-point medians
<code>as_path_len_min_v4</code>	Minimum of v4 vantage-point medians, truncated to integer: <code>int(min(peer_meds))</code>
<code>as_path_len_max_v4</code>	Maximum of v4 vantage-point medians, truncated to integer: <code>int(max(peer_meds))</code>
<code>peers_v4</code>	Number of v4 vantage points contributing medians

(Identical set for IPv6: `*_v6`.)

These metrics are recalculated repeatedly during the active 120-second window **whenever** a new hop observation is appended for that ASN.

3.2 Derived ASN-Level Metric (stored in `asn_data`)

A single representative scalar is written:

- `as_path_len = p50_v4 if present, else p50_v6`
- Implemented as:
 - `path_len = v4_p50 or v6_p50`

This value represents the representative AS-path length for that ASN during the sampling interval.

4. Database Contract

4.1 Required Tables

The script creates or ensures existence of:

- `asn_aspath_len_now`
- `asn_data` (ensures `asn` `INTEGER PRIMARY KEY` exists; adds `as_path_len` `REAL` column if missing)

Table created by script:

```
CREATE TABLE IF NOT EXISTS asn_aspath_len_now (  
  asn INTEGER PRIMARY KEY,  
  as_path_len_p50_v4 REAL,  
  as_path_len_p90_v4 REAL,  
  as_path_len_min_v4 INTEGER,  
  as_path_len_max_v4 INTEGER,  
  peers_v4 INTEGER,  
  as_path_len_p50_v6 REAL,  
  as_path_len_p90_v6 REAL,  
  as_path_len_min_v6 INTEGER,  
  as_path_len_max_v6 INTEGER,  
  peers_v6 INTEGER,  
  calc_timestamp INTEGER  
);
```

4.2 Update Semantics

After each `(origin, prefix, vantage)` observation that yields an integer origin ASN:

1. A row is immediately written/updated in `asn_aspath_len_now` using:
 - `INSERT INTO ... VALUES (...) ON CONFLICT(asn) DO UPDATE SET ...`

- The upsert is wrapped in:
 - `BEGIN IMMEDIATE; ... COMMIT;`
- 2. The representative metric (`as_path_len`) is written to `asn_data`:
 - first: `INSERT OR IGNORE INTO asn_data(asn) VALUES (?)`
 - then: `UPDATE asn_data SET as_path_len=? WHERE asn=?`
 - This update is also wrapped in:
 - `BEGIN IMMEDIATE; ... COMMIT;`

Important note (code-accurate):

- The script performs **immediate, small transactions** and does **not** batch writes.
- For each printed/upserted row, it typically performs **two separate SQLite transactions**:
 - one for `asn_aspath_len_now`
 - one for `asn_data`

5. Data Flow Overview

5.1 Input Source

Real-time RIS Live WebSocket (`UPDATE` events delivered inside `ris_message` frames).

5.2 Path Extraction

```
path    = data.get("path")
hops    = count_aspath_hops(path)
origin = origin_from_path(path) # last element if int, else None
```

5.3 Attribution Logic

Each AS_PATH observation is attributed to the origin ASN only if `origin` is an integer (`int`).

5.4 Per-Vantage Statistics

The script maintains:

- `lengths_per_peer[asn][afi][vantage] = list_of_hop_counts`

Where:

- `asn` is the integer origin ASN
- `afi` is "v4" or "v6"
- `vantage` is `vp_id = peer_asn` (string) or fallback `peer` (string)

5.5 Statistical Aggregation

For each ASN and AFI:

1. Compute per-vantage medians:
 - For each vantage: `median(list_of_hops)`
2. From the list of vantage medians (`peer_meds`), compute:
 - p50 (median-of-medians)
 - p90 (via the script's percentile function)
 - min, max (of vantage medians, truncated to int)
 - count of vantage points

5.6 Output Destinations

- `asn_aspath_len_now`: full per-AFI metrics (+ `calc_timestamp`)
- `asn_data.as_path_len`: single numeric scalar used by downstream models

6. Performance & Architecture Notes

- Single-threaded design
- Blocking WebSocket I/O (`websocket-client`)
- Immediate SQLite transactions on each printed/upserted update (no batching)
- No automatic reconnection logic inside a sampling round:
 - on non-timeout WS errors, it prints a warning and ends the round early
- 120-second live runtime window per round
- Built-in continuous operation loop:
 - runs a round (`DURATION_SEC = 120`)
 - sleeps 30 minutes (`SLEEP_BETWEEN_ROUNDS_SEC = 1800`)
 - repeats forever

7. Control Loop

High-level control flow in this codebase:

1. Open DB (`sqlite3.connect(DB_PATH, timeout=120)`)
2. Ensure tables exist / columns exist
3. Connect to WebSocket
4. Subscribe to UPDATE messages (require announcements)
5. Read `ris_message` frames
6. Ingest UPDATES; recompute + print + upsert after each valid observation

7. Stop after 120 seconds (or earlier on certain WS errors)
8. Close DB and socket
9. Sleep 30 minutes
10. Repeat forever

Note (code-accurate): The script is self-scheduling via `while True`; you do not need cron/systemd timers for periodic re-execution (though you may still run it under systemd for supervision).

8. CLI Arguments

The script has no CLI arguments.

Runtime duration is fixed by constants:

- `DURATION_SEC = 120`
 - `SLEEP_BETWEEN_ROUNDS_SEC = 30 * 60`
-

Exact How the Metric is Computed

Step 1 — Collect UPDATE messages

For every BGP UPDATE received from RIS Live (inside a `ris_message` frame):

- The script processes only if:
 - `data["type"] == "UPDATE"`
 - `path` exists and is non-empty
 - `announcements` exists and is non-empty

- Extract:
 - `path` (list containing integers and/or lists representing AS-SETs)
 - `announcements` (required by subscription and required by processing)
 - vantage identification:
 - `vp_id = str(peer_asn) if present else str(peer)` (string id)
- Identify:
 - `origin = last element of AS_PATH` only if it is an integer, else `origin = None`
 - `afi = "v6"` if prefix contains ":", else `"v4"`

Origin handling (code-accurate):

- If origin is not an integer, the update does **not** contribute to `lengths_per_peer` statistics.
- The script still records `(origin, hops, afi)` into:
 - `self.current[(vp_id, prefix)]`
even if `origin` is `None`.

Step 2 — Compute hop count

Hop count is computed by `count_aspath_hops(path)`:

- each element that is an `int` counts as 1 hop
- each element that is a `list` counts as 1 hop
- other element types do not contribute

Examples:

- `[3356, 1299, 6453] → 3 hops`
- `[3356, [6453, 2914], 6939] → 3 hops` (AS-SET list counts as 1)

Step 3 — Store hop count per vantage

If origin is an integer:

For each announced prefix in `announcements[*].prefixes[*]`:

- compute `afi` from prefix string
- append hop count:
 - `lengths_per_peer[origin][afi][vp_id].append(hops)`

If a vantage sees 4 updates for the same ASN within 120s, stored values look like:

- `[5, 4, 5, 4]`

Step 4 — Compute per-vantage median

For each vantage:

- `median_vantage = median(list_of_hops)`

Examples:

- `[5, 4, 5, 4] → median = 4.5`
- `[7, 7, 8] → median = 7`
- `[3] → median = 3`

Step 5 — Build the vantage-median list

For a given ASN and AFI:

- `peer_meds = [median_vp1, median_vp2, median_vp3, ...]`

If 9 vantages observed the ASN:

- `peer_meds` has length 9.

Step 6 — Compute global statistics

Based on `peer_meds`, compute:

- `p50 = median(peer_meds)`
- `p90 = percentile(peer_meds, 90)`
(script percentile uses sorted list + linear interpolation)
- `min_val = int(min(peer_meds))`
- `max_val = int(max(peer_meds))`
- `peers = len(peer_meds)`

These 5 values are the entire per-AFI metric.

Example:

- `peer_meds = [4, 5, 4, 6, 5, 4, 5]`
 - `p50 = 5`
 - `p90 = 6`
 - `min = 4`
 - `max = 6`
 - `peers = 7`

Step 7 — Write output (print + DB upsert + DB update)

When a new row is produced for an ASN, the script checks whether to skip duplicates:

- It compares the newly computed full row tuple against `last_printed[asn]`.
- **Important code-accurate nuance:** the row includes `calc_timestamp = int(time.time())`.
Because the timestamp changes over time, rows will typically differ even when metric values are unchanged, so print/upsert may occur very frequently during an active round.

If not skipped, it performs:

1. Write full metrics to `asn_aspath_len_now`
 - Stores all AFI-specific stats above plus `calc_timestamp` (epoch seconds).
 - Done via `INSERT ... ON CONFLICT(asn) DO UPDATE ...`
 - Wrapped in `BEGIN IMMEDIATE; ... COMMIT;`
2. Write representative scalar to `asn_data`
 - Selection logic:
 - `as_path_len = p50_v4` if present, else `p50_v6`
 - implemented as `v4_p50` or `v6_p50`
 - Done via `INSERT OR IGNORE` then `UPDATE`
 - Wrapped in a separate `BEGIN IMMEDIATE; ... COMMIT;`

Short Explanation for Engineers

“For each ASN, the script aggregates all observed AS-path hop counts from many RIS vantage points during a 120-second window. For each vantage, it computes the median AS-path length,

then computes p50/p90/min/max across those vantage medians per AFI. It writes the per-AFI stats (plus a calc_timestamp) to asn_aspath_len_now and writes a single representative scalar (v4 p50 preferred, else v6 p50) into asn_data.as_path_len. The script repeats this sampling round forever, sleeping 30 minutes between rounds. Note: because calc_timestamp is part of the stored/compared row, printing/upserting can occur very frequently during a round.”

p50 (Median-of-medians)

- Represents the typical AS-path length, robust to outliers.

p90

- Highlights the “long-tail” vantage medians (often affected by prepending/asymmetry).

min / max

- Best-case / worst-case vantage medians.

peers (vantage count)

- Indicates how many distinct vantages contributed medians.

Together, these values describe:

- structural position
- routing consistency
- propagation efficiency
- visibility
- path diversity

9. Scientific & Methodological Justification (Corrected & Strengthened)

This section provides strong, defensible arguments for why the method of computing this metric is correct and robust.

9.1 Why AS-path length is a meaningful metric

Across decades of Internet research and operations, AS-path length is widely used as:

- an indicator of topological distance between ASNs
- a proxy for propagation quality
- a measure of network centrality
- a predictor of how widely an announcement is seen

The control-plane (BGP UPDATES) is the canonical source for AS_PATH attributes, so this metric is grounded in real routing behavior.

9.2 Why multi-vantage aggregation is mandatory

A single vantage point gives an incomplete and biased view of AS-path topology. Only multi-vantage aggregation produces stable and representative metrics.

The script:

- collects AS paths from many RIS vantage points
- aggregates per vantage
- computes global statistics from vantage medians

9.3 Why median-of-medians is the correct estimator (not the mean)

AS-path length distributions are often:

- heavy-tailed
- skewed
- affected by AS prepending, leaks, asymmetry, and instability

In such distributions:

- the mean is easily distorted by extreme values
- the median is robust and stable

The script's approach:

- median per vantage
- then median across vantages

This “double-median” method is appropriate when outliers must be neutralized.

9.4 Why a 120-second window is valid

This script is explicitly designed to measure AS-path length *now*, not long-term historical averages.

A short real-time window:

- captures current routing state
- reflects near-real-time topology conditions
- avoids stale bias
- is suited for security/propagation modeling where current conditions matter

Code-accurate scheduling note: the script repeats these windows continuously (with a 30-minute sleep between rounds).

9.5 Why the method is operationally correct

The metric reflects what routers see right now, aggregated across a diverse set of collectors.

This makes the metric:

- operationally meaningful
- actionable for threat modeling
- resistant to noise
- aligned with real control-plane behavior

10. Why This Data Source Was Selected

RIPE RIS Live is ideal for this metric because:

- globally distributed collectors
- real-time UPDATE events
- full AS_PATH visibility in the control-plane stream
- strong IPv4 + IPv6 coverage (though IPv6 can be sparser per ASN)
- high vantage diversity
- widely used, industry-standard research/operations dataset

11. Usage Examples

Running the script:

```
python3 asn_aspath_len.py
```

(You may also run it under a supervisor like systemd to auto-restart on crash, but the script itself already repeats rounds indefinitely.)

12. Integration With the Platform

The metric feeds directly into:

- ASN vulnerability modeling
- attack propagation analytics
- ML feature vector construction

- combined ASN+prefix vulnerability scoring
- global Internet risk surface mapping

When combined with:

- strict filtering behavior
- reachability propagation
- RPKI coverage

...it forms a highly informative picture of an ASN's real-world influence and observed path "distance".

13. Limitations

- Depends on RIS Live uptime/availability
- SQLite writes are immediate (not batched) and can be frequent
- No automatic WebSocket reconnection inside a sampling round (non-timeout errors end the round early)
- IPv6 data may be sparse for many ASNs
- The script's `current[(vantage,prefix)]` cache is maintained but not used for eviction/withdraw handling in this version (it can grow during a round)
- Because `calc_timestamp` is included in the stored/compared row tuple, the "skip identical rows" mechanism rarely suppresses repeated prints/upserts during an active window