

asn_reachability.py — Reachability Propagation Metric (Full Technical Documentation)

1. Purpose & Role in the Platform

`asn_reachability.py` computes a **real-time propagation profile per ASN** by observing how announcements for that ASN's prefixes spread across multiple RIPE RIS vantage points within fixed time windows.

It measures:

- how widely announcements are seen across collectors (coverage)
- how fast they reach those collectors (t50 / t90 propagation delay)
- how reliable the observation is (confidence)
- and combines these into a **reachability_propagation score** and a **long-term EMA score per ASN**.

This metric reflects **actual, empirical propagation behavior**, not routing policy on paper. It is a core feature for:

- attack propagation modeling
 - Vulnerability scoring
 - global influence estimation
 - prefix+ASN combined risk scoring
 - ML-based routing risk classification
-

2. High-Level Behavior

At a high level, the script:

1. Connects to **RIPE RIS Live** via WebSocket:
`wss://ris-live.ripe.net/v1/ws/?client=bgp-tool-rp-top`
2. Loads the mapping `asn → prefixes` from `prefix_table` (default `prefix_data`).
3. Builds RIS subscription messages depending on `match_mode`:
 - `prefix` mode: subscribe to specific prefixes
 - `origin` mode: subscribe to updates where the AS is the origin
4. For each incoming UPDATE:
 - identifies the origin ASN
 - extracts one or more prefixes
 - derives a vantage-point identifier (`rrc, peer_asn_or_ip, peer_ip`)
 - maps each prefix to the owning ASN (in `prefix` mode) or uses origin ASN (in `origin` mode)
 - assigns the UPDATE to an **event window** identified by (`asn, prefix`)
5. For each event window:
 - tracks which vantage points were **active** (saw any announcement for that prefix)
 - tracks which vantage points **saw that ASN as origin** and the time they first saw it
6. When a window ends (fixed duration `window_sec`):
 - computes coverage, t50, t90, confidence, and a propagation score
 - optionally persists the event to `asn_rp_events`

- updates per-ASN live metrics in `result_table` (`asn_data` by default) using an EMA

7. In parallel, a **maintenance loop**:

- periodically computes a global baseline `t90_ref:global` from all events
- smooths this baseline over time
- performs VACUUM/ANALYZE housekeeping on the DB.

The script is designed to run continuously and maintain a real-time propagation score for each ASN.

3. Core Concepts & Events

3.1 Event definition

An **event window** represents a real-time propagation episode for a specific ASN–prefix pair.

An event window is uniquely identified by the key:

`(asn, prefix)`

Each event window has:

- a fixed start timestamp (`start_ts`)
- a fixed end timestamp (`end_ts = start_ts + window_sec`)
- a set of vantage points that observed the event
- a designated *opening vantage point* (`first_vp`)

How an event window is created

`match_mode = "prefix"`

For each incoming BGP UPDATE announcement that touches a monitored prefix:

Determine the owning ASN:

```
asn_for_pfx = ASN owner from prefix_table
```

1.

Construct the event key:

```
key = (asn_for_pfx, prefix)
```

2.

3. If no event window is currently open for this key:

- create a new EventWindow with:

- `start_ts = ts` (timestamp of the triggering UPDATE)

- `end_ts = ts + window_sec`

- `first_vp = vp_id` (the vantage point that opened the event)

Subsequent UPDATES matching the same `(asn, prefix)` during the window lifetime are associated with the same event window.

match_mode = "origin"

For each incoming BGP UPDATE announcement where:

```
origin_asn ∈ monitored ASN set
```

Construct the event key:

```
key = (origin_asn, prefix)
```

1.

2. The same window-opening logic applies:

- if no event window is open for `(origin_asn, prefix)`, a new EventWindow is created

- `first_vp` is set to the vantage point that triggered the event

In both modes, **event windows are non-overlapping per key** and are debounced to avoid rapid reopenings for the same (`asn`, `prefix`).

3.2 Active vs seen vantage points

For each event window, the script maintains two distinct vantage point sets:

active_vps

`active_vps`

The set of vantage points that observed **any UPDATE announcement** related to the event's prefix during the window.

This represents the **opportunity space**:

vantage points that *could* have seen the origin announcement during this time interval.

seen_first_ts

`seen_first_ts[vp] = timestamp`

The timestamp of the **first time each vantage point (excluding the opener VP)** observed the ASN as origin for this prefix during the event window.

Important rules:

- The **opening vantage point** (`first_vp`) is explicitly excluded from `seen_first_ts`.
- A vantage point is considered “seen” only if:
 - it is **different from** `first_vp`, and

the observation occurs **after a minimum delay**:

`(timestamp - start_ts) > T_EPS`

- where `T_EPS = 0.5s`, to avoid zero-delay noise and self-observation artifacts.

Interpretation

This separation allows the metric to:

- distinguish between:
 - vantage points that were **active** (had visibility during the event), and
 - vantage points that actually **received propagated routing information**
- compute propagation **coverage** as a ratio, not a raw count
- measure **propagation delay** from the event start to each *distinct downstream vantage point*

The propagation coverage for an event is defined as:

```
coverage_final = |seen_vps| / |active_vps|
```

Where:

- `seen_vps` excludes the opener vantage point
- `active_vps` includes all vantage points active during the window (including the opener)

As a result, coverage reflects **true multi-vantage propagation**, not mere observation by the initiating collector.

4. Metrics Produced

4.1 Per-event metrics (asn_rp_events)

Each closed event window produces a record in `asn_rp_events`:

- `asn`
- `prefix`
- `event_ts` (closure time in ISO format)

- `window_sec` (duration in seconds)
- `active_vps` (how many vantages were active for this prefix)
- `seen_vps` (how many vantages actually saw this ASN as origin)
- `coverage_final` ($\text{seen_vps} / \text{active_vps}$)
- `t50` (median propagation delay, when available)
- `t90` (90th percentile propagation delay, when available)
- `rp_score` (per-event propagation score)
- `ci_low`, `ci_high` (currently unused; set to NULL)
- `confidence` (`ok`, `low_seen`, `low_active`, `no_active_vp`)

4.2 Per-ASN live metrics (`result_table`, typically `asn_data`)

In `result_table` (default: `asn_data`), the script maintains:

- `reachability_propagation` — latest event's `rp_score`
- `rp_t50` — latest event's t50 (if any)
- `rp_t90` — latest event's t90 (if any)
- `rp_coverage` — latest event's `coverage_final`
- `rp_last_updated` — timestamp
- `rp_confidence` — confidence label from the latest event
- `rp_score_ema` — **exponentially weighted moving average** of `rp_score` per ASN

`rp_score_ema` is the **long-term propagation metric** used by the platform as the stable, noise-reduced value.

4.3 Global baseline (kv_store)

The script also maintains in `kv_store`:

- `t90_ref:global` — smoothed global reference for T90 (seconds)
 - `k_t50_over_t90` — ratio used when falling back to t50-based scoring
 - `ema:<asn>` — raw EMA values per ASN
 - `last_daily_agg` — last date aggregated into `asn_rp_daily`
-

5. Database Contract

5.1 Tables created/assumed

The script ensures existence of:

result_table (default: `asn_data`)

```
asn INTEGER PRIMARY KEY,  
reachability_propagation REAL,  
rp_t50 REAL,  
rp_t90 REAL,  
rp_coverage REAL,  
rp_last_updated TEXT,  
rp_confidence TEXT,  
rp_score_ema REAL
```

asn_rp_events

```
id INTEGER PRIMARY KEY AUTOINCREMENT,  
asn INTEGER,  
prefix TEXT,  
event_ts TEXT,  
window_sec INTEGER,  
active_vps INTEGER,  
seen_vps INTEGER,
```

```
coverage_final REAL,  
t50 REAL,  
t90 REAL,  
rp_score REAL,  
ci_low REAL,  
ci_high REAL,  
confidence TEXT
```

asn_rp_daily

```
asn INTEGER,  
date TEXT,  
avg_rp REAL,  
avg_cov REAL,  
avg_t90 REAL,  
samples INTEGER,  
last_aggregated TEXT,  
PRIMARY KEY (asn, date)
```

kv_store

```
key TEXT PRIMARY KEY,  
value TEXT
```

5.2 Update semantics

Per-event persistence

For each finalized event that passes quality filters:

- an **INSERT** is performed into **asn_rp_events**, recording:
 - **(asn, prefix)**
 - event timestamp
 - window size
 - number of active and seen vantage points

- final coverage
- computed `t50`, `t90` (if available)
- per-event reachability propagation score
- confidence classification

Events are persisted only if:

- `confidence == 'ok'` (when `PERSIST_REQUIRE_CONF_OK = True`), and
- `coverage_final >= PERSIST_MIN_COVERAGE`.

Per-ASN live metric

After a qualifying event is persisted:

- the per-ASN live row in `result_table` is updated via:

```
INSERT OR IGNORE (asn)

UPDATE result_table SET

    reachability_propagation,

    rp_t50,

    rp_t90,

    rp_coverage,

    rp_confidence,

    rp_last_updated,

    rp_score_ema
```

Exponential Moving Average (EMA)

A per-ASN exponential moving average is maintained in `kv_store` under:

```
key = "ema:<asn>"
```

with:

```
ema_now = (1 - EMA_ALPHA) * ema_prev + EMA_ALPHA * rp_score
```

The EMA value is mirrored into `result_table.rp_score_ema` and represents the platform's primary long-term propagation metric.

Baseline (t90_ref) maintenance

Daily, at a fixed UTC time, the daemon recomputes a global `t90_ref` baseline from historical events:

- using the median of valid `t90` samples from the previous day,
- applying percentile trimming and exponential smoothing.

The resulting reference value is stored in:

```
kv_store["t90_ref:global"]
```

Note:

While the schema includes the `asn_rp_daily` table, daily aggregation of per-ASN statistics is currently **defined but not executed automatically** in this version of the daemon.

6. Step-by-Step Calculation of the Metric

This section describes **exactly how the metric is computed**, step by step, in the same order and with the same conditions as in the code.

6.1 Event window lifecycle

Opening

When the first qualifying UPDATE for a given `(asn, prefix)` arrives:

```
start_ts = ts
end_ts   = ts + window_sec
```

A new `EventWindow` object is created with:

- `asn`
- `prefix`
- `start_ts`
- `end_ts`
- `first_vp` = the vantage point that triggered the event

The event key is always:

```
(asn, prefix)
```

Events are **debounced**:

- if a window for the same key was closed recently, reopening is delayed by:

```
DEBOUNCE_MULT × window_sec
```

This prevents repeated re-triggering from the same routing churn.

During the window

For every UPDATE touching that `(asn, prefix)` **within the active window**:

1. Build vantage-point ID

Each UPDATE is mapped to a unique vantage-point identifier:

```
vp_id = (rrc_id, peer_asn_or_ip, peer_ip)
```

This guarantees per-vantage visibility tracking and avoids collapsing distinct collectors.

2. Mark vantage as active

For every matching UPDATE:

```
ev.add_active(vp_id, now_ts)
```

If $\text{now_ts} \in [\text{start_ts}, \text{end_ts}]$, the vantage point is added to `active_vps`.

This defines the **set of vantage points that could have seen the event**.

3. Deduplication (important fix)

Each UPDATE is deduplicated **per vantage point**, using the signature:

```
(vp_id, prefix, origin_asn, head_asn)
```

This ensures:

- duplicate UPDATES from the same vantage do not inflate metrics
- **multi-vantage propagation is preserved** (not deduped away)

4. Marking “seen” propagation

A vantage point is marked as “**seen**” only when it represents **downstream propagation**, not the event opener.

Conditions (common):

- $\text{vp_id} \neq \text{first_vp}$
- $(\text{now_ts} - \text{start_ts}) > T_EPS$ ($T_EPS = 0.5s$)

Additional condition by match mode:

- In `match_mode = 'prefix'`:
The UPDATE must have `origin_asn == asn` (the ASN owning the prefix being

measured).

- In `match_mode = 'origin'`:
The origin ASN condition is already enforced by the subscription/filter and does not need to be checked again.

Then:

```
ev.mark_seen(vp_id, now_ts)
```

If this is the **first time** that vantage point observes the origin during this event:

```
seen_first_ts[vp_id] = now_ts
```

This ensures:

- the opening vantage point is never counted as propagation
- zero-delay artifacts and self-observation are filtered
- propagation reflects **actual multi-vantage distribution of the origin**, not generic prefix visibility

Closing

When the event window expires:

```
now_ts >= end_ts
```

The event is finalized:

```
ev.done(now_ts) → True  
_finalize_event(asn, prefix)
```

6.2 Coverage calculation

At event finalization time:

```
n_active = len(ev.active_vps)
n_seen   = len(ev.seen_first_ts)
```

If:

```
n_active == 0
```

Then:

- `coverage_final = 0.0`
- `confidence = 'no_active_vp'`
- the event is ignored for scoring

Otherwise, coverage is computed as a **ratio**:

```
coverage_final = n_seen / n_active
```

This is intentional and fundamental:

- coverage is normalized by the number of vantage points that *could* have seen the event
- sparse visibility does not unfairly penalize the ASN
- the metric measures **fractional propagation**, not raw reach

6.3 Propagation delays and t50 / t90

For each vantage point that saw the origin:

```
delay = seen_first_ts[vp] - start_ts
```

The script builds a list of delays:

```
raw_times = [dt for dt in delays if dt >= 0]
times = sorted([dt for dt in raw_times if dt > T_EPS])
```

Where:

`T_EPS = 0.5 seconds`

This removes:

- zero-delay opener artifacts
- timestamp jitter and noise

t50 (median propagation delay)

`t50` is computed **only if**:

- `coverage_final ≥ 0.5`
- `len(times) ≥ MIN_SEEN_FOR_T50` (default: 5)

t90 (90th percentile propagation delay)

`t90` is computed **only if**:

- `coverage_final ≥ 0.9`
- `len(times) ≥ 10`

Percentiles are computed by index selection on the sorted list.

This guarantees:

- `t50` is not computed from weak or partial visibility
 - `t90` is computed only for **highly global propagation events**
-

6.4 Confidence classification

After computing coverage and delays:

```
if n_active == 0:
    confidence = 'no_active_vp'
elif n_active < MIN_ACTIVE_VPS:
    confidence = 'low_active'
elif coverage_final < MIN_SEEN_RATIO:
    confidence = 'low_seen'
else:
    confidence = 'ok'
```

Where:

- `MIN_ACTIVE_VPS = 1`
- `MIN_SEEN_RATIO = 0.1`

The confidence label is therefore **directly tied to observation quality**, not heuristics.

6.5 Propagation speed term (relative to baseline)

Propagation speed is measured **relative to a dynamic global baseline**, not a fixed threshold.

At event finalization:

```
t90_ref = kv_get('t90_ref:global', default=T90_REF_DEFAULT)
```

This baseline is computed daily from historical events (see Section 7).

The speed term is computed as:

If `t90` is available:

```
speed_term = max(0.0, 1.0 - (t90 / max(1.0, t90_ref)))
```

Else if `t50` is available:

```
t50_ref = k_ratio × t90_ref    (k_ratio defaults to 0.6)
speed_term = max(0.0, 1.0 - (t50 / max(1.0, t50_ref)))
```

Else:

```
speed_term = 0.0
```

Interpretation:

- Faster-than-baseline propagation → speed_term → 1
- Equal-to-baseline → speed_term ≈ 0
- Slower propagation → speed_term → 0

For low-confidence events:

```
if conf != 'ok':
    speed_term = min(speed_term, SPEED_CAP_LOWCONF)    # default: 0.60
```

This prevents over-trusting timing derived from weak visibility.

6.6 Coverage vs speed weighting and final per-event score

Weights are defined as:

```
W_COV_OK          = 0.50
W_COV_LOWCONF     = 0.70
```

Weight selection:

```
if conf == 'ok':
    w_cov = 0.50
else:
    w_cov = 0.70
w_spd = 1.0 - w_cov
```

Final per-event score:

```
rp_score = w_cov × coverage_final + w_spd × speed_term
```

Key properties:

- High-confidence events balance **coverage and speed**
- Low-confidence events emphasize **coverage over speed**
- Speed contribution is capped when confidence is weak

Persistence gating

Events are persisted **only if**:

- `conf == 'ok'` (when `PERSIST_REQUIRE_CONF_OK = True`)
- `coverage_final ≥ PERSIST_MIN_COVERAGE` (default: 0.35)

All other events are discarded from per-ASN statistics.

6.7 Per-ASN exponential moving average (EMA)

Each persisted event updates two values:

- instantaneous score:

```
reachability_propagation = rp_score
```

- long-term smoothed score:

```
rp_score_ema
```

EMA is computed per ASN:

```
ema_prev = kv_get('ema:<asn>')
```

```
if ema_prev is None:
    ema_now = rp_score
else:
    ema_now = (1 - EMA_ALPHA) × ema_prev + EMA_ALPHA × rp_score
```

Where:

`EMA_ALPHA = 0.10`

Then:

- `kv_store['ema:<asn>'] = ema_now`
- `result_table.rp_score_ema = ema_now`

Interpretation:

- `reachability_propagation` = most recent event
- `rp_score_ema` = stable, long-term propagation capability

The platform uses `rp_score_ema` as the primary per-ASN reachability propagation metric.

7. Daily Baseline & Global t90_ref

The **maintenance loop** runs in a separate task:

1. Once per day (at configured time):

Compute a new global baseline from `asn_rp_events`:

```
SELECT t90 FROM asn_rp_events
WHERE date = ? AND t90 BETWEEN 5 AND 3600
```

○

- Require at least 50 samples.
- Sort values, remove the bottom 5% and top 5% (trim outliers).
- Take the median of the trimmed set → `t90_new`.

Smooth the global reference:

```
prev = kv_get('t90_ref:global', default=T90_REF_DEFAULT)
t90_smooth = (1 - SMOOTH_T90REF_ALPHA) * prev + SMOOTH_T90REF_ALPHA *
t90_new
# SMOOTH_T90REF_ALPHA = 0.30
```

2.

Store the new baseline:

```
kv_set('t90_ref:global', round(t90_smooth, 2))
```

3.

This rolling baseline makes the metric **self-normalizing**:

- As the Internet evolves (topology, RIS coverage, propagation behavior), the scoring automatically adapts.
- Each event is judged relative to “how fast the Internet typically propagates events right now”, not compared to a fixed arbitrary number.

8. WebSocket & Event Management

8.1 Subscription behavior

The script does **not** listen to the entire Internet blindly. It subscribes specifically to:

- known prefixes from `prefix_table` (`match_mode='prefix'`), or
- ASNs of interest (`match_mode='origin'` using RIS `path` selection).

This makes the feed efficient and focused.

8.2 Duplicate suppression

A per-vantage duplicate-suppression cache is used to avoid reprocessing identical UPDATES while still allowing true multi-vantage propagation to be observed.

Each UPDATE is identified by the signature:

```
(vp_id, prefix, origin_asn, head_asn)
```

where:

- `vp_id` uniquely identifies the RIS vantage point (rrc, peer_asn or peer_ip),
- `prefix` is the announced prefix,
- `origin_asn` is the origin extracted from AS_PATH (or -1 if missing),
- `head_asn` is the first ASN in the AS_PATH (if present).

This design ensures that:

- duplicate UPDATES from the **same vantage point** are ignored,
- UPDATES for the **same (asn, prefix)** coming from **different vantage points** are preserved and counted as independent propagation observations.

The signature cache has a time-to-live (TTL) dynamically tied to the measurement window:

```
sig_ttl_sec = max(30s, window_sec)
```

allowing duplicates to expire naturally once the event window is no longer relevant.

This per-vantage deduplication is a critical correctness feature: it prevents artificial suppression of propagation speed and coverage while still protecting the system from excessive duplicate processing.

8.3 Event limit and eviction

To avoid unbounded growth:

- maximum number of concurrent events: `MAX_PARALLEL_EVENTS` (default 2000).
 - when the limit is reached, an **eviction priority function** chooses which event to close early (based on coverage and duration).
 - per-AS limits and cooldowns prevent a single ASN from dominating the event pool.
-

9. CLI & Runtime Configuration

Main arguments:

- `--db` — SQLite path (default `asn_data.db`)
 - `--prefix-table` — table with `(asn, prefix)` (default `prefix_data`)
 - `--result-table` — target table (default `asn_data`)
 - `--max-prefixes` — max prefixes per ASN (optional)
 - `--window-sec` — event window duration (default 600s)
 - `--t90-ref` — initial reference for t90 (default 600s)
 - `--match-mode` — `'prefix'` or `'origin'`
 - `--asns` — optional list of ASNs to restrict to
 - `--daily-ref-utc` — time for daily baseline (default `00:05`)
 - `--housekeep-utc` — time for VACUUM/ANALYZE (default `03:00`)
 - `--no-daemon` — run only the main daemon (no maintenance loop)
-

10. Why This Calculation Method Is Robust and Hard to Challenge

10.1 Multi-vantage, ratio-based coverage

- Coverage is not “how many collectors saw it”, but: “what fraction of the collectors that were active for this prefix saw this ASN as origin.”

This is crucial because:

- RIS vantage distribution is not uniform.
- Some prefixes are visible only from a subset of RRCs.
- A raw count can be misleading; a ratio is fair and normalized.

This choice directly addresses the classic “single-view bias” problem in Internet measurement.

10.2 Confidence gating

The script explicitly refuses to trust poorly observed events:

- no events with `coverage_final < 0.35` contribute to the metric
- no events with `confidence != 'ok'` are persisted (by default)
- t50 and t90 are computed **only** when there are enough vantage points and enough samples.

This removes the usual weak point in measurements: drawing conclusions from low-signal, low-sample events.

10.3 Quantiles over means for delay

Propagation delay distributions are:

- skewed
- heavy-tailed
- affected by temporary anomalies, leaks, and prepending

Using **median (t50)** and **90th percentile (t90)** instead of mean:

- makes the metric robust to a small number of extreme delays
- captures both “typical speed” (t50) and “tail behavior” (t90)
- matches standard practice in latency / performance monitoring (p50 / p90).

This is a textbook approach for non-Gaussian, heavy-tailed distributions.

10.4 Relative, self-normalizing scoring

`rp_score` doesn’t compare t90 to some arbitrary fixed value. Instead:

- each event’s speed is evaluated relative to a **global t90 baseline** that:
 - is built from actual recent events
 - is trimmed (5–95% percentile range)
 - is smoothed with EMA.

This means:

- the metric automatically tracks “normal Internet behavior”
- changes in routing and topology over time are incorporated
- the score remains meaningful without manual retuning.

10.5 Balanced combination of coverage & speed

The final score:

`rp_score = coverage_weight * coverage + speed_weight * speed_term`

- High-confidence events use 50/50 coverage vs speed
- Low-confidence events emphasize coverage (70%) and cap speed

This weighting is intuitive and defensible:

- coverage answers: “how many vantage points accepted this?”
- speed answers: “how fast did it propagate once it started?”

The design says: *“When in doubt about timing, trust coverage more than speed.”*

10.6 Long-term EMA to stabilize per-ASN values

Per-ASN values are not just “last event wins”. They are:

- accumulated via an EMA with `EMA_ALPHA = 0.10`
- making the metric:
 - stable
 - resistant to short-lived spikes
 - still responsive to structural changes in propagation.

This combination of **real-time events + EMA + global baseline** yields a per-ASN propagation score that is:

- statistically sound
- operationally meaningful
- very difficult to attack with “you just picked arbitrary thresholds” type arguments.

11. Integration With the Platform

The metric feeds:

- attack propagation modeling engines
- Vulnerability estimation (how dangerous it is if this ASN is abused)

- global influence maps (which ASNs can “light up” a large portion of the Internet)
- ML features for ASN risk scoring
- combined prefix+ASN vulnerability scores.

It is especially powerful when correlated with:

- strict filtering metrics
- RPKI enforcement status
- AS-path length statistics.

12. Performance & Architecture Notes

- Single process, async I/O (WebSockets + aiosqlite)
- Controlled event pool size and per-AS rate limiting
- Periodic heartbeat for monitoring
- Daily maintenance to keep DB compact and baseline fresh

Designed for long-running operation under a supervisor.

13. Limitations

- Relies on RIS vantage coverage (not all networks are equally observed)
- Only events passing the coverage and confidence filters are used in scoring
- Event key is (asn, prefix), not per-subprefix or per-AS-path variant
- Daily baseline requires enough events per day to be meaningful.

14. Why RIPE RIS Live Was Selected

- Dozens of globally distributed RRC collectors
- Real-time UPDATE feed over WebSocket
- Full AS_PATH and origin information
- Excellent fit for near-real-time propagation analytics.