

asn_irr_coverage.py — IRR Coverage Metric

1. Purpose & Role in the Platform

`asn_irr_coverage.py` computes Internet Routing Registry (IRR) coverage for each ASN by comparing:

- The set of **announced prefixes that can be successfully parsed** for the ASN (retrieved from RIPEstat), with
- The set of **IRR route / route6 objects** registered for that ASN across multiple IRR WHOIS databases.

The metric quantifies:

- how many valid / parsable announced prefixes have **exact IRR route-object matches**
- how many valid / parsable announced prefixes are **covered by any IRR object** (exact or less-specific supernet)
- the **distribution of IRR objects by source registry** (as reported by WHOIS or server fallback)
- how many **WHOIS errors and timeouts** occurred during collection

The script does **not** attempt to validate correctness, authorization, or policy legitimacy of IRR objects beyond **prefix + origin ASN matching**.

The metric feeds:

- vulnerability scoring
- routing policy-compliance scoring
- prefix + ASN vulnerability modeling
- ML-based risk classification

- global routing quality assessments
-

2. High-Level Behavior

During execution, the script:

- Ensures the IRR results table exists in SQLite
- Reads ASNs from `asn_data` in large batches
- For each ASN:
 - Retrieves all announced prefixes from RIPEstat (`/announced-prefixes`)
 - Queries multiple IRR WHOIS servers **sequentially**:
 - `rr.ntt.net`
 - `whois.radb.net`
 - `whois.ripe.net`
 - Parses `route` / `route6` objects
 - Extracts prefix, origin ASN, and IRR source
 - Deduplicates IRR objects by `(prefix, origin, source)`
 - Computes:
 - exact-match coverage
 - covering-match (supernet) coverage
 - provenance counts
 - error / timeout counts

- Prints a per-ASN summary
- Writes results into `irr_results`

The script runs in **continuous rounds**.

Each round processes the full ASN set once, then sleeps **30 minutes** before starting the next round.

3. Metrics Produced

For each ASN, the following fields are computed and written to the database.

3.1 Coverage Percentages

`exact_pct`

Fraction of valid / parsable announced prefixes that have an **exact-matching** IRR route object.

`covering_pct`

Fraction of valid / parsable announced prefixes covered by **any IRR object** (exact or supernet).

3.2 Raw Counts

- `exact_covered` — number of valid announced prefixes with exact IRR matches
- `covering_covered` — number of valid announced prefixes covered by exact or supernet IRR objects
- `total` — total number of valid / parsable announced prefixes, defined as:

```
announced_nets = [ipnet(p) for p in announced if ipnet(p)]
```

```
total = len(announced_nets)
```

3.3 Provenance

IRR object counts by source registry, derived from:

- `source`: WHOIS fields when present, or
- the queried server's configured label (`NTT`, `RADB`, `RIPE`) as fallback

Also stored:

- total number of IRR objects considered
 - total number of valid announced prefixes (`total`)
-

3.4 Error Metrics

- `irr_errors_count` — number of non-timeout exceptions during WHOIS queries
 - `irr_timeouts_count` — number of WHOIS queries that raised `asyncio.TimeoutError`
-

3.5 Timestamp

- `updated_at` — UNIX timestamp (seconds)
-

4. Database Contract

4.1 Created Objects

The script creates (if missing):

1. **One destination table:**

```
irr_results (
```

```
asn INTEGER PRIMARY KEY,  
  
irr_coverage_exact_pct      REAL,  
  
irr_exact_covered           INTEGER,  
  
irr_total_advertised        INTEGER,  
  
irr_coverage_covering_pct   REAL,  
  
irr_covering_covered        INTEGER,  
  
irr_provenance_json         TEXT,  
  
irr_errors_count            INTEGER,  
  
irr_timeouts_count          INTEGER,  
  
updated_at                  INTEGER  
  
)
```

2. **One index** on the timestamp column:

```
CREATE INDEX IF NOT EXISTS idx_irr_results_updated ON  
irr_results(updated_at);
```

SQLite Initialization Details (Exact Behavior)

During database initialization (`ensure_db_ready()`), the script applies the following pragmas:

- `journal_mode = WAL`
- `synchronous = NORMAL`
- `temp_store = MEMORY`

- `page_size = 4096`
- `cache_size = -200000`

Persistence semantics:

- Pragmas that are **database-scoped** (`journal_mode`, `page_size`) persist for all connections.
- Pragmas that are **connection-scoped** (`synchronous`, `temp_store`, `cache_size`) are applied only during initialization and are **not explicitly re-applied** in per-ASN UPSERT connections.

As a result:

- All writes operate in **WAL mode**
- Other performance-related pragmas may vary by connection

4.2 Update Semantics

Per-ASN writes use **UPSERT** semantics:

```
INSERT ... ON CONFLICT(asn) DO UPDATE SET ...
```

Because `DEF_COMMIT_PER_ASN = True`, each ASN is written and committed immediately using:

```
BEGIN IMMEDIATE;
```

```
...
```

```
COMMIT;
```

No batching is used in the default configuration.

5. Core Concepts and Data Flow

5.1 Fetching Announced Prefixes (RIPEstat)

The script uses:

```
/data/announced-prefixes
```

Prefixes are extracted and normalized using:

```
ipaddress.ip_network(prefix, strict=False)
```

Prefixes that fail parsing are **discarded and never counted**.

If RIPEstat fetching fails for any reason, the function returns an empty list, resulting in:

```
total == 0
```

5.2 Querying IRR Databases

For each ASN, the script queries each IRR WHOIS server sequentially using:

```
-T route,route6 -i origin AS<asn>
```

Parsed fields:

- `route:` / `route6:` → prefix
- `origin:` → origin ASN
- `source:` → IRR database name

If `source` is missing, the server's configured label is used as fallback.

5.3 Coverage Logic (Exact Implementation)

Given:

- `announced` = prefixes returned by RIPEstat
- `announced_nets` = successfully parsed prefixes
- `irr_objs` = (`prefix`, `origin_asn`, `source`) from IRR WHOIS

Coverage rules:

Exact match

```
irr_prefix == announced_prefix
```

Covering match (supernet)

```
irr_prefix.version == announced_prefix.version
```

```
AND announced_prefix.subnet_of(irr_prefix)
```

Counts:

```
exact_covered    = number of exact matches
```

```
covering_covered = exact + supernet matches
```

```
total            = len(announced_nets)
```

Percentages:

```
exact_pct        = 100 * exact_covered / total
```

```
covering_pct     = 100 * covering_covered / total
```


If `total == 0`:

- `exact_pct = None`
 - `covering_pct = None`
 - `all counts = 0`
 - `provenance = empty`
-

5.4 Provenance Tracking

For each IRR object:

- increment count by source registry

Stored JSON structure:

```
{  
  "sources": { "RIPE": X, "RADB": Y, "NTT": Z },  
  "objects_total": N,  
  "announced_total": total  
}
```

6. Error & Timeout Accounting

For each IRR WHOIS query inside `process_one_asn()`:

- **Timeout** → `irr_timeouts_count += 1`
Triggered **only** when the outer call raises:

```
asyncio.wait_for(query_whois(...), timeout=req_timeout + 3)
```

including connection-timeouts.

- **Other exceptions** → `irr_errors_count += 1`

Important:

Read-timeouts inside `query_whois()` (during `reader.read()`) are caught internally and terminate reading without raising an exception; therefore, **they are not counted** as timeouts.

7. Final Per-ASN Summary & DB Write

After computing metrics, a per-ASN summary is printed:

```
AS123: exact=42.00% (21/50), covering=86.00% (43/50), errors=0,  
timeouts=1
```

Then:

```
upsert_one_asn(...)
```

Exactly **one transaction per ASN**.

8. Token Bucket (RPS Enforcement)

The script enforces `--rps` only for RIPEstat HTTP requests using a token bucket:

- `rate = rps`

- `capacity = 2 * rps`

Every RIPEstat HTTP request must acquire a token.

WHOIS requests are not rate-limited by the token bucket.

9. Concurrency Model (Exact Behavior)

9.1 Per-ASN Concurrency

Because:

```
DEF_COMMIT_PER_ASN = True
```

ASNs are processed **sequentially**, one at a time:

```
for asn in chunk:
    done += await handle(asn)
```

No parallel ASN processing occurs.

9.2 Async I/O Inside Each ASN

Within a single ASN:

- RIPEstat HTTP request is asynchronous
 - IRR WHOIS queries are asynchronous but awaited **sequentially per server**
 - No WHOIS parallelism exists
-

9.3 Optional Multi-ASN Concurrency (Disabled)

If `DEF_COMMIT_PER_ASN = False`, the script would use:

```
asyncio.gather(...)
```

enabling concurrent processing of multiple ASNs.

A semaphore is instantiated but has **no practical effect** when `DEF_COMMIT_PER_ASN = True`; it becomes relevant only in the `asyncio.gather` execution path.

Summary:

- Per-ASN concurrency: **OFF**
 - Per-ASN async I/O: **ON**
 - Multi-ASN concurrency: **Disabled**
-

10. CLI Arguments

- `--db` — SQLite database (must contain `asn_data`)
 - `--table` — destination table (default: `irr_results`)
 - `--where` — optional SQL WHERE clause
 - `--req-timeout` — request timeout (default: 12s)
 - `--max-retries` — RIPEstat retries (default: 4)
 - `--backoff-base` — exponential backoff base
 - `--rps` — RIPEstat requests per second
 - `--batch-fetch` — ASNs per DB batch (default: 50,000)
-

11. Performance & Architecture Notes

- Async HTTP for RIPEstat
 - Async TCP WHOIS (sequential per server)
 - Exponential backoff applies only to RIPEstat
 - One SQLite commit per ASN
 - WAL mode enabled
 - Designed for large ASN sets (50k–100k per round)
-

12. Control Loop Summary

1. Initialize DB
2. Fetch ASNs in batches
3. For each ASN:
 - get RIPEstat prefixes
 - query IRR WHOIS
 - parse & dedupe objects
 - compute exact / covering coverage
 - compute provenance
 - write DB
 - print summary
4. End round
5. Sleep 30 minutes

6. Repeat indefinitely
-

13. Limitations

- Prefix-only comparison (no path / RPKI logic)
 - No normalization against global IRR corpus
 - WHOIS outages reduce completeness
 - No IRR conflict detection
 - Dependent on RIPEstat availability
-

14. Why These Data Sources Were Selected

RIPEstat

- authoritative
- broad IPv4 / IPv6 visibility
- widely trusted

IRR Databases (RIPE, RADB, NTT)

- widely deployed
- standard route-object references
- global operational relevance