Multi-IRR Source Query Architecture

1. Overview

The IRR Automation tool queries 7 different Internet Routing Registry (IRR) sources to get comprehensive routing prefix data. Each source may have different or overlapping data, so results are merged using a SET UNION operation to eliminate duplicates.

This document explains each of the 7 IRR sources, the protocols used to query them, and how results are combined to produce a complete list of prefixes for any given ASN.

2. The 7 IRR Sources

Source 1: RIPE (Reseaux IP Europeens)

Property	Value
Type	Regional Internet Registry (RIR)
Coverage	Europe, Middle East, Central Asia
Protocol	REST API
Endpoint	https://rest.db.ripe.net/search.json

Query Method: HTTP GET request with inverse lookup by origin ASN.

```
GET https://rest.db.ripe.net/search.json?
    source=ripe&
    query-string=AS15169&
    inverse-attribute=origin&
    type-filter=route
```

Response: Structured JSON with nested objects containing route attributes.

Source 2: RADB (Routing Assets Database)

Property	Value
Type	Internet Routing Registry
Coverage	Global (Merit Network)
Protocol	WHOIS (TCP port 43)
Server	whois.radb.net

Query Method: Socket connection to port 43 with inverse origin lookup.

```
Connect to whois.radb.net:43
Send: "-i origin AS15169\r\n"
Receive: Text with route/route6 objects
```

Source 3: ARIN (American Registry for Internet Numbers)

Property	Value
Type	Regional Internet Registry (RIR)

IRR Automation - Multi-Source Query Architecture

Coverage	North America
Protocol	WHOIS (TCP port 43)
Server	rr.arin.net

Query Method: Socket connection to port 43 with inverse origin lookup.

```
Connect to rr.arin.net:43
Send: "-i origin AS15169\r\n"
Receive: Text with route/route6 objects
```

Source 4: APNIC (Asia-Pacific Network Information Centre)

Property	Value
Type	Regional Internet Registry (RIR)
Coverage	Asia Pacific
Protocol	WHOIS (TCP port 43)
Server	whois.apnic.net

Query Method: Socket connection to port 43 with inverse origin lookup.

```
Connect to whois.apnic.net:43
Send: "-i origin AS15169\r\n"
Receive: Text with route/route6 objects
```

Source 5: LACNIC (Latin America and Caribbean NIC)

Property	Value
Type	Regional Internet Registry (RIR)
Coverage	Latin America & Caribbean
Protocol	WHOIS (TCP port 43)
Server	irr.lacnic.net

Query Method: Socket connection to port 43 with inverse origin lookup.

```
Connect to irr.lacnic.net:43
Send: "-i origin AS15169\r\n"
Receive: Text with route/route6 objects
```

Source 6: AFRINIC (African Network Information Centre)

Property	Value
Type	Regional Internet Registry (RIR)
Coverage	Africa
Protocol	WHOIS (TCP port 43)
Server	whois.afrinic.net

Query Method: Socket connection to port 43 with inverse origin lookup.

```
Connect to whois.afrinic.net:43
Send: "-i origin AS15169\r\n"
Receive: Text with route/route6 objects
```

Source 7: NTTCOM (NTT Communications)

Property	Value
Type	Internet Routing Registry
Coverage	NTT Communications network
Protocol	WHOIS (TCP port 43)
Server	rr.ntt.net

Query Method: Socket connection to port 43 with inverse origin lookup.

```
Connect to rr.ntt.net:43
```

```
Send: "-i origin AS15169\r\n"
Receive: Text with route/route6 objects
```

3. Union Merge Strategy

Why Use Set Union?

Different IRR sources may contain:
- Overlapping data: Same prefix registered in multiple IRRs
- Unique data: Prefixes only in regional RIR (e.g., ARIN-only routes)
- Stale data: Old entries not yet cleaned up

Using SET UNION ensures:
1. No duplicates in the final result
2. Complete coverage from all sources
3. Simple and fast merge operation

Implementation

```
def fetch_prefixes(self, target: str, irr_sources: List[str]) -> PrefixResult:
    result = PrefixResult()

    for source in irr_sources:
        try:
            v4, v6 = self._query_source(target, source)

            # UNION: Add all prefixes from this source
            result.ipv4_prefixes.update(v4)  # Set union
            result.ipv6_prefixes.update(v6)  # Set union

            result.sources_queried.append(source)
        except RADBClientError as e:
            result.errors.append(f"Failed to query {source}: {e}")

    return result
```

Example Merge

Source 1 (RIPE): {8.8.8.0/24, 8.8.4.0/24}
Source 2 (RADB): {8.8.8.0/24, 172.217.0.0/16} <- 8.8.8.0/24 is duplicate
Source 3 (ARIN): {8.34.208.0/20}

Union Result: {8.8.8.0/24, 8.8.4.0/24, 172.217.0.0/16, 8.34.208.0/20}
(4 unique prefixes - duplicate removed)

4. Protocol Details

REST API (RIPE only)

RIPE is the only source using REST API. This provides:
- Structured JSON responses
- HTTP caching friendly
- Easy error handling with status codes

```
url = "https://rest.db.ripe.net/search.json"
params = {
    "source": "ripe",
    "query-string": "AS15169",
    "inverse-attribute": "origin",
```

```
    "type-filter": "route",
}
response = session.get(url, params=params)
```

WHOIS Protocol (All other sources)

All non-RIPE sources use WHOIS protocol (TCP port 43):
- Raw socket connection
- Text-based request/response
- Requires parsing RPSL objects

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(60)
sock.connect((server, 43))
sock.sendall(f"-i origin {target}\r\n".encode("utf-8"))

# Receive response
response = b""
while True:
    chunk = sock.recv(4096)
    if not chunk:
        break
    response += chunk
sock.close()
```

WHOIS Response Parsing

Extract prefixes from RPSL format using regex:

```
def _parse_whois_response(response: str):
    ipv4 = set()
    ipv6 = set()

    # Extract IPv4: "route: 8.8.8.0/24"
    for match in re.finditer(r"^route:\s+(\S+)", response, re.MULTILINE):
        ipv4.add(match.group(1))

    # Extract IPv6: "route6: 2001:4860::/32"
    for match in re.finditer(r"^route6:\s+(\S+)", response, re.MULTILINE):
        ipv6.add(match.group(1))

    return ipv4, ipv6
```

5. Error Handling

Each source is queried independently. If one fails, others continue:
- Errors are logged but do not stop the process
- Partial results are returned even if some sources fail
- The errors list in the result contains details of any failures

```
for source in irr_sources:
    try:
        v4, v6 = self._query_source(target, source)
        result.ipv4_prefixes.update(v4)
        result.ipv6_prefixes.update(v6)
        result.sources_queried.append(source)
    except RADBClientError as e:
        # Log error but continue with other sources
        result.errors.append(f"Failed to query {source}: {e}")

# Return partial results even if some sources failed
return result
```

6. Performance Comparison

Source	Protocol	Typical Latency	Reliability
RIPE	REST	100-500ms	High
RADB	WHOIS	200-800ms	High
ARIN	WHOIS	200-600ms	High
APNIC	WHOIS	300-1000ms	Medium
LACNIC	WHOIS	400-1200ms	Medium
AFRINIC	WHOIS	500-1500ms	Medium
NTTCOM	WHOIS	200-800ms	High

Optimization Tips

- Put fastest sources first in configuration
- Set appropriate timeout (default: 60 seconds)
- Use retry logic for transient failures
- Consider removing slow/unreliable sources if not needed

7. Configuration

In config.yaml, specify which sources to query:

```
# Query order matters - faster/more reliable sources first
irr_sources:
  - RIPE      # Primary - REST API, fast
  - RADB      # Global coverage
  - ARIN      # North America
  - APNIC     # Asia Pacific
  - LACNIC    # Latin America
  - AFRINIC   # Africa
  - NTTCOM    # NTT network
```