



RIPE NCC
RIPE NETWORK COORDINATION CENTRE

Querying the DFZ

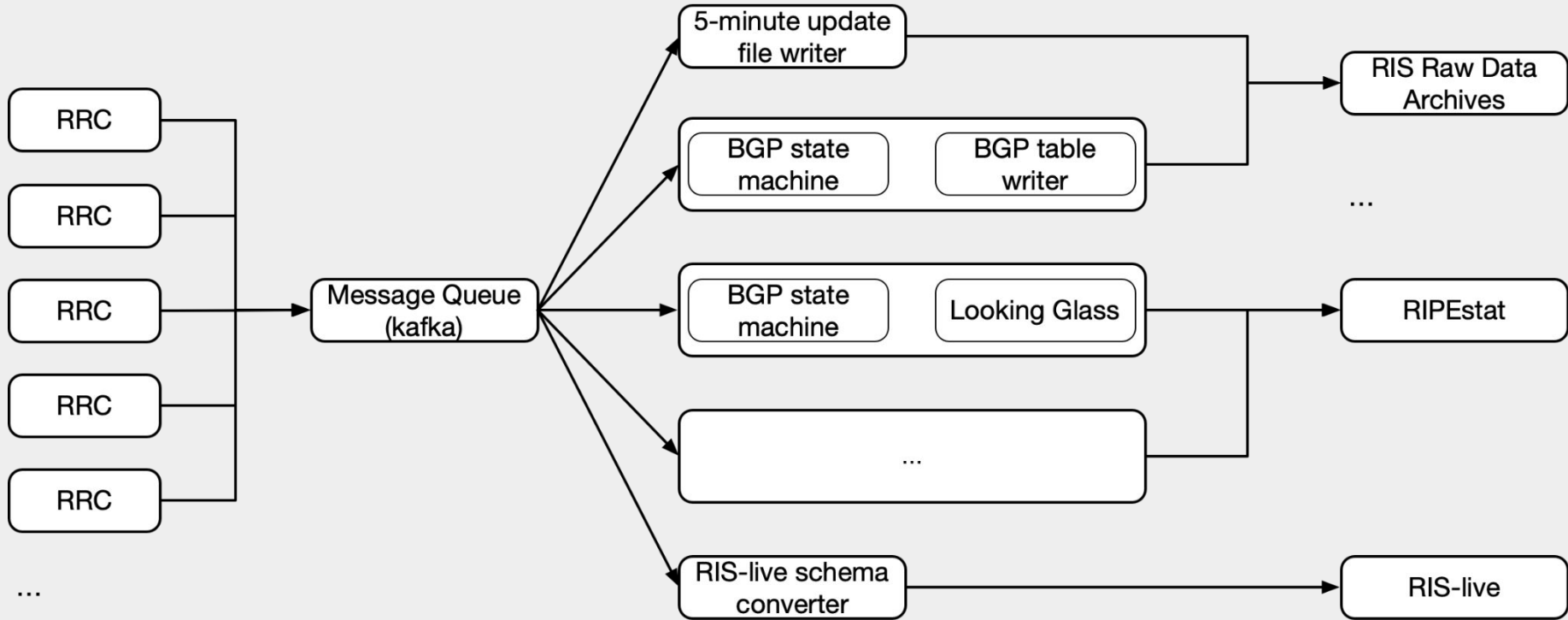
Analysing RIBs and updates using
SQL/dataframes



Recap: Route Collectors

- A route collector is a BGP speaker that has the goal of capturing BGP messages.
- The *number* of collectors is an infrastructure artifact.
- Users care about having relevant information and being able to process it.
- Route Collector projects are a collaboration between those who run them (RIPE NCC, NCSC) and the peers feeding the data.

RIS Architecture



<https://ris.ripe.net/docs/>



The raw data is available

- The BGP archives are available in MRT format at:
<https://ris.ripe.net/docs/mrt/>
- MRT files: RFC 6396 (file format)
 - MRT contains raw BGP messages
 - MRT parsers implement relevant BGP RFCs (e.g. for the definition of path attributes)
- Two types of files:
 - Full table dumps ("bview"/"rib") every [x hours] for [all peers of collector]
 - BGP updates for [time window] from [all peers of collector]
- One file per collector per time-period
 - 23 collectors (RIS)
- Data is processed
 - with CLI tools (monocle from BGPKIT, BGPReader, BGPStream, CAIDA)...
 - Custom scripts
 - Custom programmes

Four Approaches to Using Route Collector Data



Web Applications

Use pre-processed data in a website



CLI

Process MRT on the CLI



Scripts

Scripts that parse MRT files and extract information



Data Science

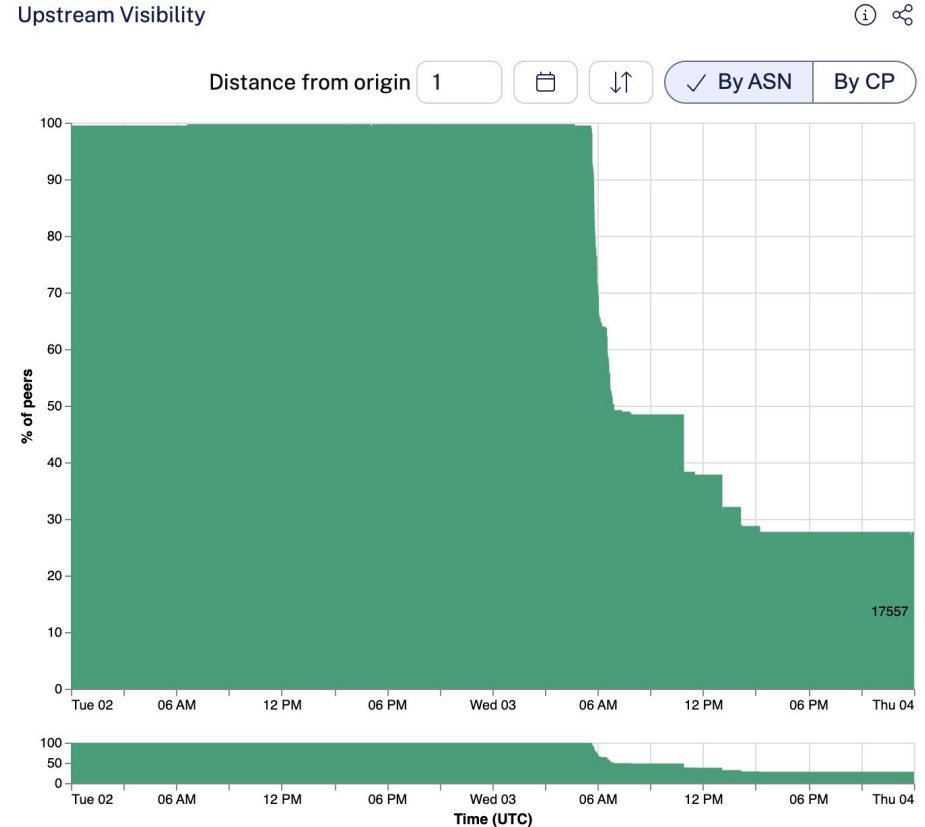
View BGP data as a dataset. Use SQL/...

Web Application Example: RIPEstat - Upstream Visibility



Upstream Visibility

- Shows the distribution of the upstream ASNs transiting a prefix, as seen by RIS.
- *Very useful* when visibility changes.
- Example: prefix became RPKI invalid (AS 0 ROA)



Showing results for 115.186.103.0/24 from 2025-09-02 00:00:00 to 2025-09-04 00:00:59

Web Application Example: RIPEstat - Upstream Visibility



Upstream Visibility

- Shows the distribution over upstreams ASNs.
- *Very useful* when visibility changes.
- Example: prefix became RPKI invalid (AS 0 ROA).
- Most upstreams of AS 17557 filtered, some stayed.



Upstream Visibility

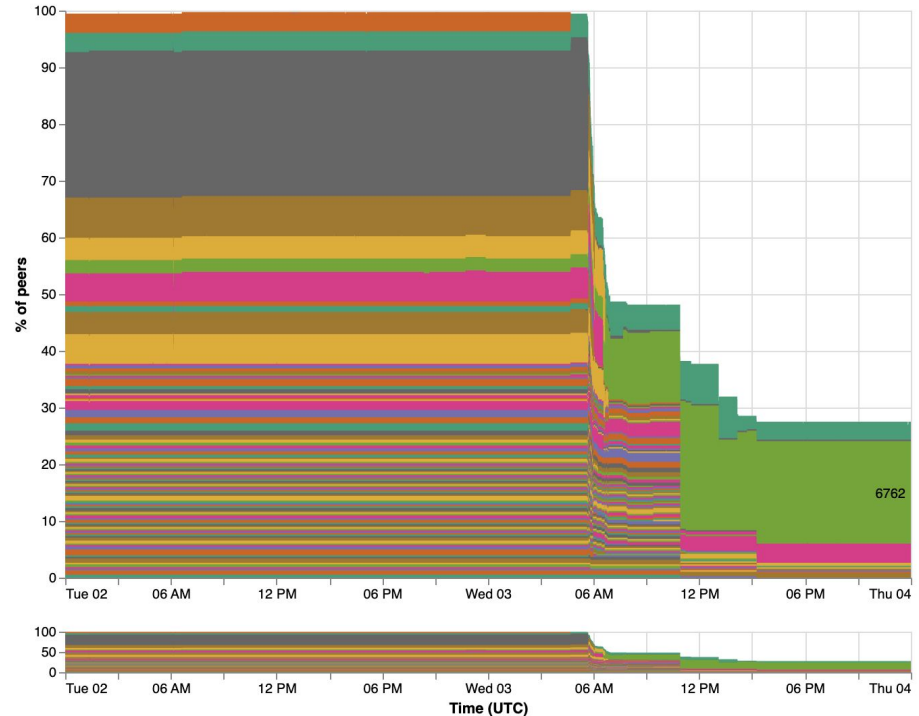


Distance from origin 2



By ASN

By CP



Showing results for 115.186.103.0/24 from 2025-09-02 00:00:00 to 2025-09-04 00:00:59

~/tmp

```

) wget https://data.ris.ripe.net/rrc00/2025.09/bview.20250905.0800.gz
--2025-09-05 12:07:50-- https://data.ris.ripe.net/rrc00/2025.09/bview.20250905.0800.gz
Resolving data.ris.ripe.net (data.ris.ripe.net)... 2001:67c:2e8:25::c100:b18, 193.0.11.24
Connecting to data.ris.ripe.net (data.ris.ripe.net)|2001:67c:2e8:25::c100:b18|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 406560243 (388M) [application/octet-stream]
Saving to: 'bview.20250905.0800.gz'

```

```

bview.20250905.0800.gz          100%[=====>] 387.73M  3.65MB/s   in 54s

```

```

2025-09-05 12:08:44 (7.18 MB/s) - 'bview.20250905.0800.gz' saved [406560243/406560243]

```

~/tmp 54s

```

) bgpreader -m -d singlefile -o rib-file=bview.20250905.0800.gz | grep '193.0.14.0/24' | head -n 100 | fold -w 120 -s
WARN: No time window specified, defaulting to all available data
TABLE_DUMP2|1757059200|B|80.77.16.114|34549|193.0.14.0/24|34549 25152|IGP|80.77.16.114|0|0|34549:200 34549:10000|NAG||
TABLE_DUMP2|1757059200|B|165.16.221.66|37721|193.0.14.0/24|37721 25152|IGP|165.16.221.66|0|0|37721:2000 37721:6002
37721:6003 37721:6004 37721:11000 37721:11100 37721:11101|NAG||
TABLE_DUMP2|1757059200|B|185.210.224.254|49432|193.0.14.0/24|49432 25152|IGP|185.210.224.254|0|0|1000||NAG||
TABLE_DUMP2|1757059200|B|193.33.94.251|58057|193.0.14.0/24|58057 1836 25152|IGP|193.33.94.251|0|0|1836:20000 1836:110
1836:3000 1836:3020 58057:65010 25152:1016|NAG||
TABLE_DUMP2|1757059200|B|202.150.221.37|38001|193.0.14.0/24|38001 136168 137955 25152|IGP|202.150.221.37|0|0||NAG||
TABLE_DUMP2|1757059200|B|5.255.90.109|202365|193.0.14.0/24|202365 43727 9198 25152|IGP|5.255.90.109|0|0|43727:0
43727:500|NAG||
TABLE_DUMP2|1757059200|B|45.61.0.85|22652|193.0.14.0/24|22652 6939 25152|IGP|45.61.0.85|0|0||NAG||
TABLE_DUMP2|1757059200|B|49.12.70.222|44393|193.0.14.0/24|44393 394256 25152|IGP|49.12.70.222|0|0|25152:1148|NAG||
TABLE_DUMP2|1757059200|B|89.234.186.6|204092|193.0.14.0/24|204092 6939 25152|IGP|89.234.186.6|0|0|150|64496:100
64496:2150|NAG||
TABLE_DUMP2|1757059200|B|165.254.255.2|15562|193.0.14.0/24|15562 2914 12859 25152|IGP|165.254.255.2|0|0|2914:410
2914:1206 2914:2203 2914:3200 12859:4000 25152:1|NAG||
TABLE_DUMP2|1757059200|B|193.163.86.231|34800|193.0.14.0/24|34800 58057 1103 25152|IGP|193.163.86.231|0|0|2603:665
2603:667 11537:40 11537:888 20965:3000 20965:5000 25152:1|NAG||
TABLE_DUMP2|1757059200|B|45.12.55.0|208972|193.0.14.0/24|208972 6939 25152|IGP|45.12.55.0|0|0||NAG||
TABLE_DUMP2|1757059200|B|102.217.156.3|328977|193.0.14.0/24|328977 25152|IGP|102.217.156.3|0|0|25152:1356|NAG||

```

```

1 import bgpkit
2 from collections import Counter
3
4 views = bgpkit.Broker().query(ts_start="2025-09-05T08:00:00", ts_end="2025-09-05T08:00:00", data_type="rib", project=
  "riperis")
5
6 upstreams = Counter()
7 for file_metadata in views:          # b.gtld-servers.net has address 192.33.14.30
8     parser = bgpkit.Parser(url=file_metadata.url, filters={"prefix": "192.33.14.0/24"})
9     for route in parser:
10         as_path = (route.as_path or "").split()
11         if len(as_path) > 1:
12             upstreams[as_path[-2]] += 1
13
14 print(f"[{elapsed()}] Results:")
15 for upstream, cnt in upstreams.most_common(10):
16     print(f"  AS{upstream:<8} {cnt:>4}")
17
18
19
20
21

```

```

> uv run python main.py
[ 0.4s] Reading 23 dump files...

```

```

[2133.8s] Results:

```

AS7342	77
AS396746	60
AS396761	46
AS396707	29
AS396688	17
AS396652	17
AS396675	14
AS396748	13
AS396686	12
AS396658	12



Workflow: parse mrt to CSV, import into database, query.

```
SELECT
  prefix,
  splitByChar(' ', as_path)[-2] AS first_upstream,
  count(*) AS cnt
FROM rib
WHERE prefix = '193.0.14.0/24'
GROUP BY ALL
ORDER BY cnt DESC
LIMIT 5
```

Query id: 02917ea2-4239-45cb-b21a-df41a04f9410

	prefix	first_upstream	cnt
1.	193.0.14.0/24	6939	70
2.	193.0.14.0/24	513	20
3.	193.0.14.0/24	20612	20
4.	193.0.14.0/24	28186	15
5.	193.0.14.0/24	1103	13

5 rows in set. Elapsed: 0.019 sec. Processed 4.64 million rows, 157.65 MB (250.59 million rows/s., 8.52 GB/s.)
Peak memory usage: 249.23 KiB.






Realisations

- Tools that are available on-line limit what information you can use.
- Parsing MRT files is slow and files are (relatively) big.
 - As a result, a lot of (academic) analysis of the Internet is done on limited information.
 - “... focusing on RRCs named RRC00, RRC01, and RRC10. ...”
- Analysis often require you to extract the relevant information first, and post-process it.
- Indexing MRT in a database (PostgreSQL, clickhouse) is expensive (disk space).
 - Might be the best solution for exact lookups.
 - PostgreSQL has good (`GiST inet_ops`) indexes for prefixes.



Alternative

- Summary: **parquet files presented here are available**
- We are now publishing `parquet` files that contain data from RIS
 - 18 month experiment, go/no-go after 12 months.
- `parquet` fits in the data science ecosystem. Especially with DuckDB .
- Two types of files:
 - *Daily* file for RIB at midnight.
 - *Hourly* files for updates.
- Short-term archive: 14 days (RIB/updates).

What you can do is not new. However, *it* got more accessible.



Alternative

- The first (older) files are available at
 - RIB:
<https://data.ris.ripe.net/derived/prototypes/parquet/v0/bview/>
 - Updates:
<https://data.ris.ripe.net/derived/prototypes/parquet/v0/updates/>
- Waiting for an infrastructure change for automatic publication of files.
- Contact me if you need files for different periods for research.



What is included?

Columns in parquet file

- `ts`
- `operation (A/W)`
- `prefix (str)`
- `origin_as (str)`
- `as_path (str[])`
- `peer_ip (str)`
- `peer_asn (u32)`
- `communities (str[])`
- ...

RIB

- ~445M rows for RIS
- ~4.5GB
- Slightly smaller than the MRT inputs

Updates

- Size depends on BGP update volume
- Typically
 - ~40-50M rows
 - ~200-325MB



Design goals

- Contain the *relevant* data.
- Model data at the BGP update/announce level.
- Correct for (academic) analysis.
 - i.e. AS sets need to be supported.
- Ergonomic for queries.
- Compact file size.
- Evolve the format



Design goals

- Contain the *relevant* data.
- Model data at the BGP update/announce level.
- Correct for (academic) analysis.
 - i.e. AS sets need to be supported.
- Ergonomic for queries.
- Compact file size.

- Evolve the format:
 - Can make internal changes without a version change
 - Can add fields without a version change
 - Removing data or format is a new version

```
D select prefix, peer_ip, peer_asn, as_path, communities from '2025-08-01T-bview-bview-rrc-all.parquet' where prefix = '192.33.14.0/24';
```

100%

prefix varchar	peer_ip varchar	peer_asn uint32	as_path varchar[]	communities varchar[]
192.33.14.0/24	185.1.8.3	48237	[48237, 26415, 396...	NULL
192.33.14.0/24	185.1.8.50	200612	[200612, 174, 2107...	['0:0', '174:17417', '20061:2', '20061:192']
192.33.14.0/24	185.1.8.59	49544	[49544, 26415, 396...	['49544:24000', '49544:24039']
192.33.14.0/24	193.242.98.141	29680	[29680, 26415, 396...	['64512:11', '64512:21', '60082:64512:11', ...
192.33.14.0/24	193.242.98.98	60082	[26415, 396624, 39...	['64512:11', '64512:21', '0:2:64512:000000...
192.33.14.0/24	198.32.160.103	13030	[13030, 2914, 7342...	['2914:410', '2914:1203', '2914:2201', '29...
192.33.14.0/24	198.32.160.12	199524	[199524, 11164, 23...	NULL
192.33.14.0/24	198.32.160.137	19151	[19151, 26415, 396...	NULL
192.33.14.0/24	198.32.160.168	51185	[51185, 26415, 396...	['51185:1', '51185:575', '51185:666', '511...
192.33.14.0/24	198.32.160.182	9002	[9002, 1299, 7342, ...	NULL
192.33.14.0/24	198.32.160.221	11164	[11164, 2381, 2641...	['2381:55038', '11164:1140', '11164:7500']
192.33.14.0/24	198.32.160.242	24482	[24482, 2381, 2641...	['2381:55038', '24115:2381', '24115:65023'...
192.33.14.0/24	198.32.160.39	9304	[9304, 26415, 3967...	NULL
192.33.14.0/24	198.32.160.42	2497	[2497, 3356, 7342, ...	NULL
192.33.14.0/24	198.32.160.58	21700	[21700, 174, 2107, ...	['174:21101', '174:22022', '21700:101:4', ...
192.33.14.0/24	198.32.176.128	11164	[11164, 2381, 2641...	['2381:55038', '11164:1140', '11164:7500']
192.33.14.0/24	198.32.176.14	2914	[2914, 7342, 36617]	['2914:410', '2914:1008', '2914:2000', '29...
192.33.14.0/24	198.32.176.164	19151	[19151, 26415, 396...	NULL
192.33.14.0/24	198.32.176.177	7575	[7575, 4608, 26415...	NULL
192.33.14.0/24	198.32.176.226	199524	[199524, 7575, 460...	NULL
192.33.14.0/24	2001:67c:ec:253::1	44097	[44097, 26415, 396...	['64512:20002', '44097:65000', '51706:1000...
192.33.14.0/24	23.147.168.62	200454	[200454, 26415, 39...	['200454:0:2', '200454:1:40542']
192.33.14.0/24	23.159.16.22	835	[835, 1299, 7342, ...	['835:10004', '1299:431', '1299:888', '129...
192.33.14.0/24	2602:fc26:12::1	1003	[1003, 26415, 3966...	['1003:1300', '1003:1360', '1003:1361', '6...
192.33.14.0/24	31.15.255.209	3212	[3212, 26415, 3967...	['3212:1121']
192.33.14.0/24	45.10.88.1	59514	[59514, 26415, 396...	['59613:666']
192.33.14.0/24	45.67.83.250	207010	[207010, 24010, 26...	['124010:65000', '124010:65020', '124010:6503...

```
D SELECT
  count(*) cnt,
  as_path[-2:-2]
FROM '2025-08-01T-bview-bview-rrc-all.parquet'
WHERE prefix = '192.33.14.0/24'
GROUP BY ALL ORDER BY 1 DESC LIMIT 10;
```

cnt int64	as_path[-2:-2] varchar[]
83	[7342]
56	[396746]
43	[396761]
29	[396707]
19	[396688]
17	[396659]
14	[396675]
13	[396748]
11	[396686]
11	[396654]

10 rows 2 columns

```
D EXPLAIN ANALYZE SELECT count(*) cnt, as_path[-2:-2] FROM '2025-08-01T-bview-bview-rrc-all.parquet' WHERE prefix = '192.33.14.0/24' GROUP BY ALL ORDER BY 1 DESC LIMIT 10;
```

Query Profiling Information

```
EXPLAIN ANALYZE SELECT count(*) cnt, as_path[-2:-2] FROM '2025-08-01T-bview-bview-rrc-all.parquet' WHERE prefix = '192.33.14.0/24' GROUP BY ALL ORDER BY 1 DESC LIMIT 10;
```

Total Time: 1.69s

More Complicated: |prefixes| Transiting Carrier Networks

```
D PIVOT (  
  SELECT  
    DISTINCT prefix,  
    CASE WHEN ':' in prefix THEN 'ipv6' ELSE 'ipv4' END AS afi,  
    asn,  
    CASE WHEN regexp_matches(collector, '^rrc[0-9][0-9]') THEN 'ris' ELSE 'rv' END AS project  
  FROM tier_one  
  LEFT JOIN '20250901T0800-ris-rv-bview-rrc-all.parquet' ris  
  ON asn IN ris.asn  
  ) ON afi, project USING count(distinct prefix) ORDER BY asn::int ASC;  
100%  
100%  
100%
```

asn varchar	ipv4_ris int64	ipv4_rv int64	ipv6_ris int64	ipv6_rv int64
174	991237	988554	202320	203857
701	103197	77941	10516	209380
1299	988154	989396	214794	214317
2914	988596	989217	214359	214352
3257	987782	988186	214158	214171
3320	987190	987260	187992	188000
3356	997073	998822	217189	217886
3491	990749	990924	214453	214444
5511	631821	840536	163185	152402
6453	988158	989627	217803	214985
6461	988548	988737	215490	215465
6762	988990	981200	214670	185481
6830	987531	987569	215544	215589
6939	1010929	1011308	219976	219998
7018	986438	986668	213852	213850
12956	989209	983480	213985	205658
16 rows		5 columns		



MRT as Data Science/SQL problem

- The previous examples used relatively simple SQL.
- Selecting data can also be the first step in deeper analysis.
- Parquet is a common format for data science.
 - Efficient implementations: e.g., reads only the selected data.
 - BGP analysis as data science. ✨



MRT as Data Science/SQL problem

- The previous examples used relatively simple SQL.
- Selecting data can also be the first step in deeper analysis.
- Parquet is a common format for data science.
 - Efficient implementations: e.g., reads only the selected data.
 - BGP analysis as data science. ✨
- Computationally friendly
 - You can work with a month of all RIS updates

```
[1]: import polars as pl
import duckdb
import matplotlib.pyplot as plt
```

```
[2]: duckdb.sql("select count(*) from read_parquet('/data/tdecock/2025-02-10-updates.*.parquet')")
```

```
[2]:
```

count_star() int64
1644977253

```
[3]: duckdb.sql("""
CREATE TEMPORARY TABLE freqs AS
SELECT
count(*) update_count, prefix
FROM
read_parquet('/data/tdecock/2025-02-10-updates.*.parquet')
group by all order by 1 desc;
""")
```

```
[4]: duckdb.sql("select * from freqs limit 5")
```

```
[4]:
```

update_count int64	prefix varchar
12888703	169.145.140.0/23
8115627	2a03:eec0:3212::/48
7580594	160.238.104.0/22
6129279	172.224.198.0/24
5696689	104.206.88.0/22

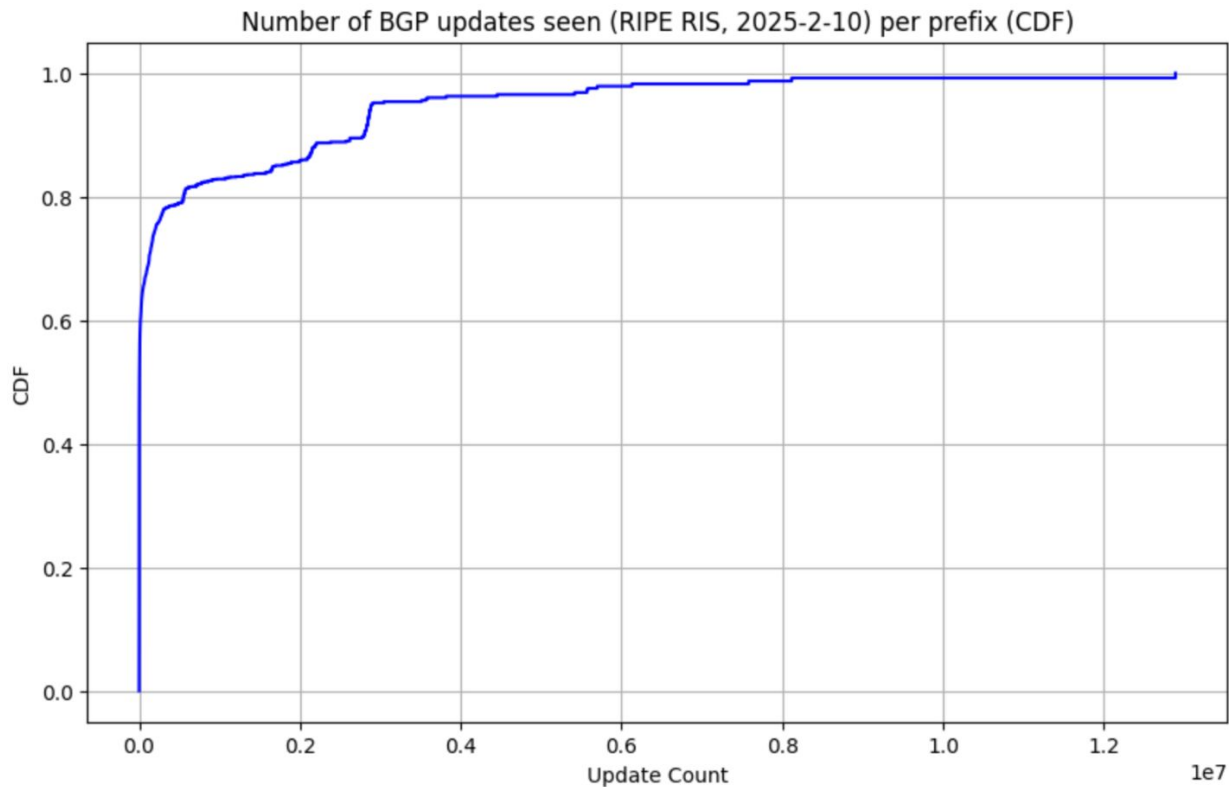
▼ Create a histogram of the number of BGP updates per prefix

```
[8]: df = duckdb.sql("""
SELECT
    update_count, prefix,
    SUM(update_count) OVER (ORDER BY update_count) AS cumulative_sum,
    SUM(update_count) OVER (ORDER BY update_count) / SUM(update_count) OVER () AS cdf,
FROM
    freqs;
""").df()
df.sample(10)
```

```
[8]:
```

	update_count	prefix	cumulative_sum	cdf
367905	486	181.29.48.0/24	123293709.0	0.074952
847871	676	2400:3ca0:45::/48	391394238.0	0.237933
1195147	1262	51.179.38.0/23	677362784.0	0.411776
556275	531	165.227.64.0/20	218410150.0	0.132774
152719	426	102.22.95.0/24	23740173.0	0.014432
699055	584	45.234.123.0/24	297786057.0	0.181027
574264	537	218.202.214.0/23	228943411.0	0.139177
18503	1	195.150.48.0/20	20664.0	0.000013
1010027	768	2804:390:2290::/45	508760855.0	0.309281
291163	470	187.195.250.0/23	87952974.0	0.053468

```
[9]: plt.figure(figsize=(10, 6))
plt.step(df["update_count"], df["cdf"], where='post', linestyle='-', color='b')
plt.xlabel('Update Count')
plt.ylabel('CDF')
plt.title('Number of BGP updates seen (RIPE RIS, 2025-2-10) per prefix (CDF)')
plt.grid(True)
plt.show()
```

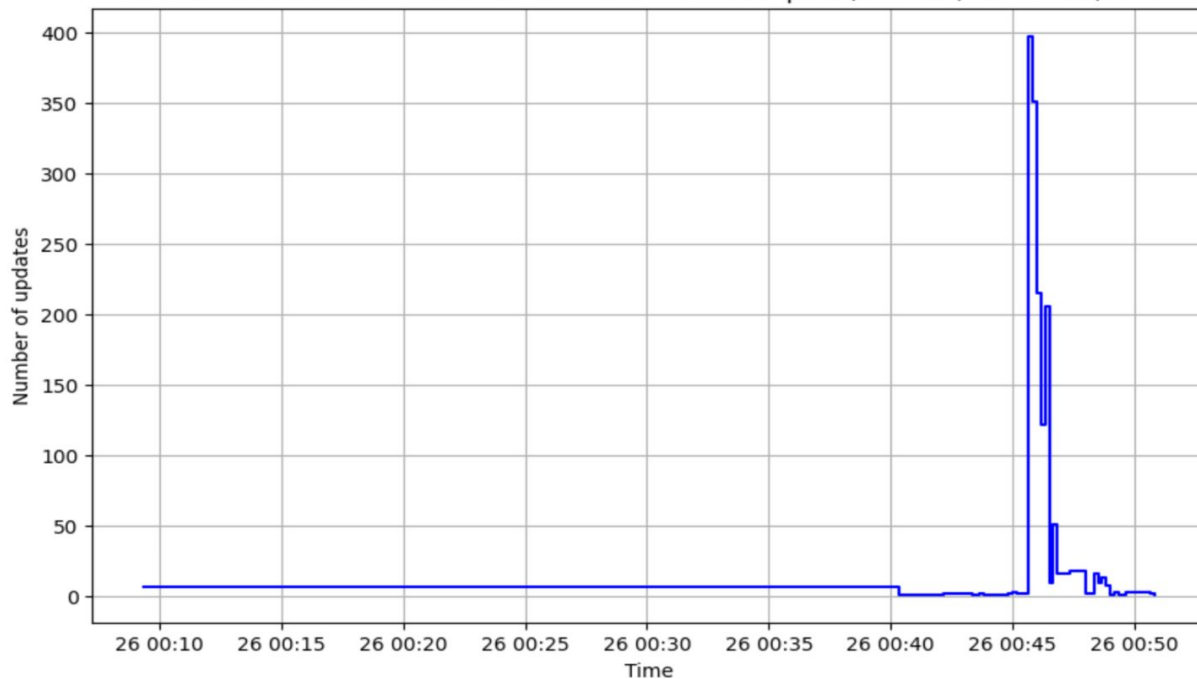


BGP Updates after Baltic Sea Cable Damage



```
[9]: # Plot the CDF
plt.figure(figsize=(10, 6))
plt.step(df["time_window_start"], df["update_count"], where='post', linestyle='-', color='b')
plt.xlabel('Time')
plt.ylabel('Number of updates')
plt.title('Number of BGP announcements seen in RIS with 1299 8194 in AS path (RIPE RIS, 2025-1-26, 0:00-01:30 UTC)')
plt.grid(True)
plt.show()
```

Number of BGP announcements seen in RIS with 1299 8194 in AS path (RIPE RIS, 2025-1-26, 0:00-01:30 UTC)





How to do this analysis

- The first files are available at
 - RIB:
<https://data.ris.ripe.net/derived/prototypes/parquet/v0/bview/>
 - Updates:
<https://data.ris.ripe.net/derived/prototypes/parquet/v0/updates/>
- Example python workbook:
 - <https://github.com/ties/mrt-parquet-workbook-samples/>
- We will publish automated updates soon.
- The files can be used from any tool that reads parquet.



Conclusion

- We introduced a prototype of parquet-based format as an alternative for raw MRT files
 - Separate files for RIBs (bview) and BGP updates.
- Route Collector data is more accessible than you would expect.
- Using SQL makes relatively complicated analysis possible.
- We **want to hear your feedback about these datasets.**
 - We will produce these files for at least 18 months.
 - We need to learn what schema works best.
 - If the schema changes, we will keep producing the old format for 3 months.
- We **want your input** on the peering strategy we are writing



Questions & Comments



tdekock@ripe.net

Bonus: LLMs Are Useful for Unfamiliar SQL



Claude Code v2.0.14
Sonnet 4.5 · Claude Pro

```
> SELECT
  count(*),
  CASE IF ':' IN prefix THEN 'v6' ELSE 'v4' END AS proto
  peer_ip,
  peer_asn
FROM
  '~/src/tmp/parquet/bview/year=2025/month=10/day=10/20251010.0000-ris-all-collectors.parquet'
GROUP BY
  ALL;
```

- I can see you have a SQL query with a syntax error. Let me help you fix it:

The issue is in the CASE statement - it should use WHEN...THEN syntax, not IF. Here's the corrected query:

```
SELECT
  count(*),
  CASE WHEN ':' IN prefix THEN 'v6' ELSE 'v4' END AS proto,
  peer_ip,
  peer_asn
FROM
  '~/src/tmp/parquet/bview/year=2025/month=10/day=10/20251010.0000-ris-all-collectors.parquet'
GROUP BY
  ALL;
```

The key changes:

1. Changed CASE IF to CASE WHEN
2. Added a comma after the proto column (it was missing in the original)

This query will:

- Count the number of rows for each unique combination of peer_ip and peer_asn
- Classify each row as 'v6' or 'v4' based on whether the prefix contains a colon
- Group by all non-aggregated columns (proto, peer_ip, peer_asn)