

EDA of Billion Dollar Retail Store Outlet

This is an EDA done on the Android logs of “Firebase Performance” metrics for a Billion Dollar Retail Store Outlet.

event_timestamp	TIMESTAMP	Timestamp since Epoch when event started on client device (trace start, network start, etc). Timestamp since Epoch when event started on client device (trace start, network start, etc)
app_display_version	STRING	The display version of the application. VersionName for Android; CFBundleShortVersionString for iOS. E.g. "4.1.7". The display version of the application. VersionName for Android; CFBundleShortVersionString for iOS. E.g. "4.1.7".
app_build_version	STRING	The build version of the application. VersionCode for Android; CFBundleVersion for iOS. E.g. "1523456". The build version of the application. VersionCode for Android; CFBundleVersion for iOS. E.g. "1523456".
os_version	STRING	Android API level or iOS version. E.g. "26" (Android) or "11.4" (iOS). Android API level or iOS version. E.g. "26" (Android) or "11.4" (iOS).
device_name	STRING	Name of the client device. E.g. "Google Pixel". Name of the client device. E.g. "Google Pixel".
country	STRING	2-letter country code of the country from which the event took place. E.g. "US" or "ZZ" (unknown). 2-letter country code of the country from which the event took place. E.g. "US" or "ZZ" (unknown).
carrier	STRING	Carrier of the client device.
radio_type	STRING	Active radio type when the event took place. E.g. "WIFI". Active radio type when the event took place. E.g. "WIFI".
custom_attributes	RECORD	All custom attributes attached to this event. All custom attributes attached to this event.
event_type	STRING	Type of the event. Possible values: DURATION_TRACE (including app start, foreground, background, and all developer instrumented traces); SCREEN_TRACE (traces spanning the lifetime of screens); TRACE_METRIC (developer instrumented metrics that are associated with traces, previously known as counters); NETWORK_REQUEST. Type of the event. Possible values: DURATION_TRACE (including app start, foreground, background, and all developer instrumented traces); SCREEN_TRACE (traces spanning the lifetime of screens); TRACE_METRIC (developer instrumented metrics that are associated with traces, previously known as counters); NETWORK_REQUEST.
event_name	STRING	Name of the event. For DURATION_TRACE, the trace name. For SCREEN_TRACE: "_st_" followed by the trace name. For NETWORK_REQUEST: the network request url pattern. For TRACE_METRIC: the metric name. Name of the event. For DURATION_TRACE, the trace name. For SCREEN_TRACE: "_st_" followed by the trace name. For NETWORK_REQUEST: the network request url pattern. For TRACE_METRIC: the metric name.
parent_trace_name	STRING	Name of the parent trace that carries the trace metric. Only present for TRACE_METRIC. Name of the parent trace that carries the trace metric. Only present for TRACE_METRIC.
trace_info	RECORD	Only present for DURATION_TRACE, SCREEN_TRACE, TRACE_METRIC. Only present for DURATION_TRACE, SCREEN_TRACE, TRACE_METRIC.

duration_us	INTEGER	For DURATION_TRACE, SCREEN_TRACE: duration that this trace lasts. For TRACE_METRIC: duration that the parent trace lasts. Unit: microsecond. For DURATION_TRACE, SCREEN_TRACE: duration that this trace lasts. For TRACE_METRIC: duration that the parent trace lasts. Unit: microsecond.
screen_info	RECORD	Only present for SCREEN_TRACE. Only present for SCREEN_TRACE.
slow_frame_ratio	FLOAT	The ratio of slow frames for this screen trace, between 0 and 1. E.g. a value of 0.05 means 5% of the frames for this screen instance took more than 16ms to load. The ratio of slow frames for this screen trace, between 0 and 1. E.g. a value of 0.05 means 5% of the frames for this screen instance took more than 16ms to load.
frozen_frame_ratio	FLOAT	The ratio of frozen frames for this screen trace, between 0 and 1. E.g. a value of 0.05 means 5% of the frames for this screen instance took more than 700ms to load. The ratio of frozen frames for this screen trace, between 0 and 1. E.g. a value of 0.05 means 5% of the frames for this screen instance took more than 700ms to load.
metric_info	RECORD	Only present for TRACE_METRIC. Only present for TRACE_METRIC.
metric_value	INTEGER	Value of the trace metric.
network_info	RECORD	Only present for NETWORK_REQUEST. Only present for NETWORK_REQUEST.
response_code	INTEGER	HTTP response code for the network response. E.g. 200 or 404. HTTP response code for the network response. E.g. 200 or 404.
response_mime_type	STRING	MIME type of the network response. E.g. "text/html". MIME type of the network response. E.g. "text/html".
request_http_method	STRING	HTTP method of the network request. E.g. "GET" or "POST". HTTP method of the network request. E.g. "GET" or "POST".
request_payload_bytes	INTEGER	Size of the network request payload. Unit: byte. Size of the network request payload. Unit: byte.
response_payload_bytes	INTEGER	Size of the network response payload. Unit: byte. Size of the network response payload. Unit: byte.
request_completed_time_us	INTEGER	Microseconds after event_timestamp when network request sending is complete. Unit: microsecond. Microseconds after event_timestamp when network request sending is complete. Unit: microsecond.
response_initiated_time_us	INTEGER	Microseconds after event_timestamp when network response is initiated. Unit: microsecond. Microseconds after event_timestamp when network response is initiated. Unit: microsecond.
response_completed_time_us	INTEGER	Microseconds after event_timestamp when network response is completed. Unit: microsecond. Microseconds after event_timestamp when network response is completed. Unit: microsecond.

Big Query Console: <https://console.cloud.google.com/welcome>

Big Query Documentation: https://cloud.google.com/bigquery/docs/reference/standard-sql/date_functions#extract

How to setup data in BigQuery:

- All Firebase data is Stored in Google Cloud infrastructure.
- Firebase retains performance data for 30 days.
- **Go to Firebase Console** - <https://console.firebase.google.com>



- Go back to Firebase → **Project Settings** → **Integrations**
- Find **BigQuery** and click "**Link**".
- After linking, Firebase will begin exporting daily **Analytics event data** into BigQuery automatically.
- Go to BigQuery Console - <https://console.cloud.google.com/bigquery>
- Required Permissions for Linking Firebase to BigQuery - **Firebase Admin** or **Owner** role on the Firebase project for linking services.
- In Google Cloud Console, go to **IAM & Admin** → Search your user email. Confirm if you have both Firebase Admin and BigQuery Admin/Data Editor roles.
- Make sure **BigQuery API** is enabled in Google Cloud Console.
- Go to analytics.google.com and select the GA4 property connected to your app.
- Use Project ID or property ID in Firebase Analytics or GA4 to **SEARCH** in Bigquery after you integrate Bigquery in both of them. You will find Project ID in settings.
- They are like plugins that must be enabled in Integrations tab on firebase and Admin -> product Links -> Bigquery Links in GA4.
- Wait at least 24 hrs after events are fired from Frontend.

Select a resource

No organization ▼

Search projects and folders

Recent Starred All

	Name	Type	ID
✓ 	No organization 	Organization	0

Explorer

+ Add data



Search BigQuery resources



☐ Show starred only

- ▼ [redacted]t--prod ☆ ⋮
 - ▶ 📁 Repositories ⋮
 - ▶ 🔍 Queries ⋮
 - ▶ 📖 Notebooks ⋮
 - ▶ 📊 Data canvases ⋮
 - ▶ 🛠 Data preparations ⋮
 - ▶ 📡 Pipelines ⋮
 - ▶ 🔌 External connections ⋮
- ▼ 🏠 firebase_messaging ☆ ⋮
 - 📄 data ☆ ⋮
- ▼ 🏠 firebase_performance ☆ ⋮
 - 📄 com_[redacted]. ☆ ⋮
 - 📄 com_[redacted]. ☆ ⋮

Note: Only the first 10 rows are shown in the screenshot. To see the full picture you must execute the query.

Total records

```
SELECT COUNT(*) FROM `project---prod.firebase_performance.com_project_ANDROIDID`
```

12464469 - 1.2 million records

Maybe we should optimise the number of events sent per device.

Date range of this data?

```
SELECT
  MIN(EXTRACT(DATE FROM event_timestamp)) AS `min_date`,
  MAX(EXTRACT(DATE FROM event_timestamp)) AS `max_date`
FROM
  `project---prod.firebase_performance.com_project_ANDROIDID`
```

Row	min_date ▼	max_date ▼
1	2025-03-20	2025-05-21

Past 2 months from today 22 May 2025

Display versions

```
SELECT
  app_display_version,
  COUNT(1) AS `count`,
  ROUND(
    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---prod.firebase_performance.com_project_ANDROIDID`), 2
  ) AS `percent`
FROM
  `project---prod.firebase_performance.com_project_ANDROIDID`
GROUP BY 1
ORDER BY 2 DESC
```

Row	app_display_version	count	percent
1	1.2.35	10930172	87.88
2	1.2.31	1442403	11.6
3	1.2.43	33719	0.27
4	1.2.39	6152	0.05
5	1.2.34	5279	0.04
6	1.2.36	2688	0.02
7	1.2.42	2422	0.02
8	1.2.21	2304	0.02
9	1.2.46	2303	0.02
10	1.2.44	1985	0.02

1.2.21 to 1.2.46 a total of 22 records

1.2.46 has 2178 hits which is the latest version.

Most used is 1.2.35 with 10796488 hits followed by 1.2.31 with 1616318.

Interestingly version 1.2.21 has 2235 hits. Old version is still being used. No wonder we see the same errors on crashlytics.

Build Versions

```

SELECT
  app_build_version,
  COUNT(1) AS `count`,
  ROUND(
    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---
prod.firebase_performance.com_project_ANDROID`), 2
  ) AS `percent`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC

```

Row	app_build_version	count	percent
1	2387	10929299	87.88
2	2383	1443189	11.6
3	2395	33683	0.27
4	2391	6143	0.05
5	2386	5279	0.04
6	2388	2688	0.02
7	2394	2433	0.02
8	2374	2427	0.02
9	2398	2303	0.02
10	2396	1985	0.02

2374 to 2398 a total of 22 records.

Latest build 2398 has 2178 hits.

Most used build is 2387 with 10795615 followed by 2383 with 1617104 hits.

Interestingly build 2374 still has 2358 hits in the past 2 months. Which shows why we still get old errors on crashlytics. Old versions are still used.

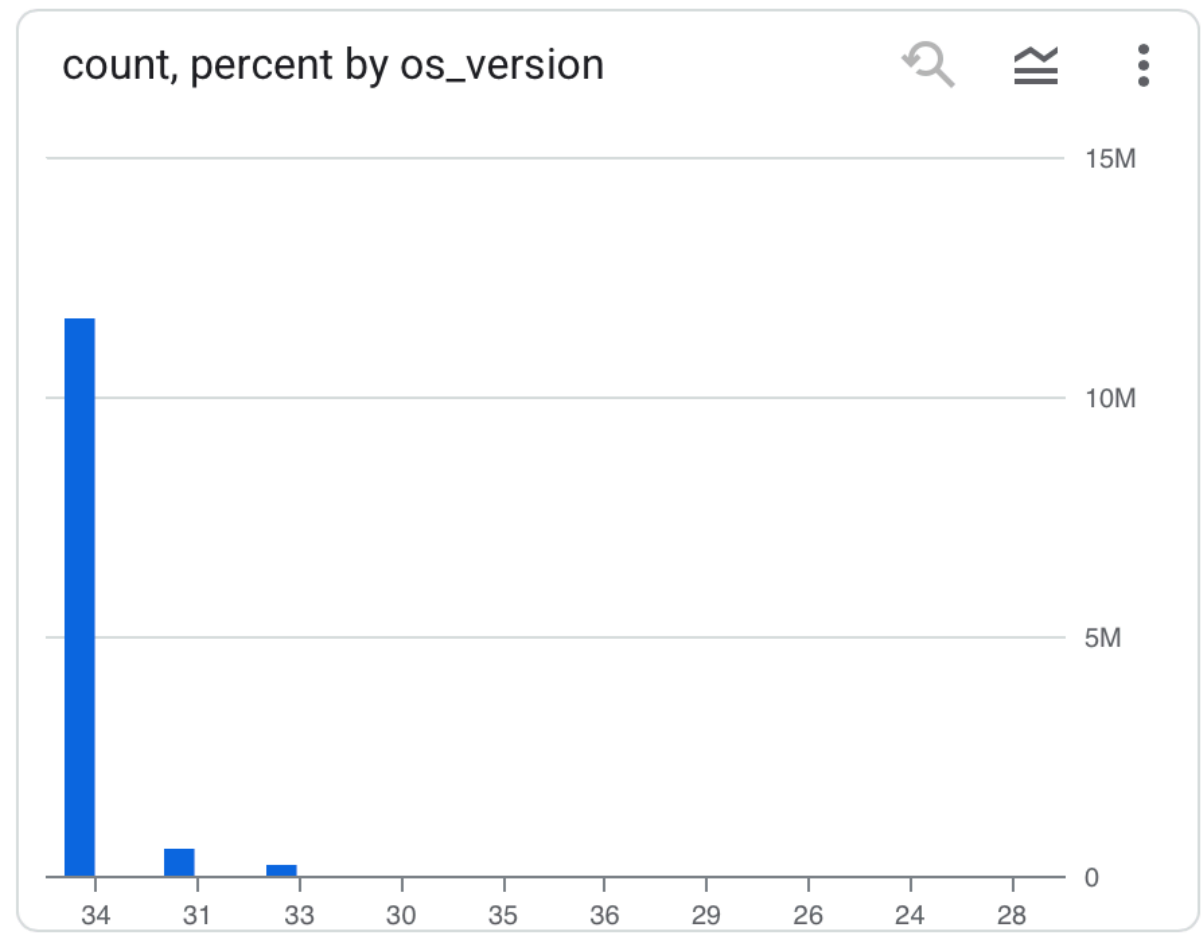
Most used OS?

```

SELECT
  os_version,
  COUNT(1) AS `count`,
  ROUND(
    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---
prod.firebase_performance.com_project_ANDROID`), 2
  ) AS `percent`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC

```

Row	os_version	count	percent
1	34	11656061	93.72
2	31	538843	4.33
3	33	236813	1.9
4	30	3135	0.03
5	35	1514	0.01
6	36	514	0.0
7	29	36	0.0
8	26	10	0.0
9	24	7	0.0
10	28	6	0.0



We can set min SDK for android to 30 to leverage all the new features and better security. This also makes the dev process easier as we don't have to support older APIs.

Highly recommend the client to transition all devices to Android 30 or higher for better security and perf.

Key security enhancements in **Android 11 (API level 30)**:

- **One-time permissions** for location, camera, and microphone.
- **Auto-reset of permissions** for unused apps.
- **Scoped storage enforcement** to isolate app data.
- **Stricter background location access** (requires user to enable in settings).
- **Foreground service requirement** for background access to sensitive data.
- **Data access auditing APIs** for tracking how apps access user data.
- **Components default to non-exported** unless explicitly declared.
- **Incremental APK installation** for safer and faster app installs.
- **Kernel security hardening** with improved memory safety and patching.
- **Identity Credential API** for secure digital IDs (e.g., mobile driver's licenses).

Most used devices

```
SELECT
  device_name,
  COUNT(1) AS `count`,
  os_version
```

```
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1, 3
ORDER BY 2 DESC
```

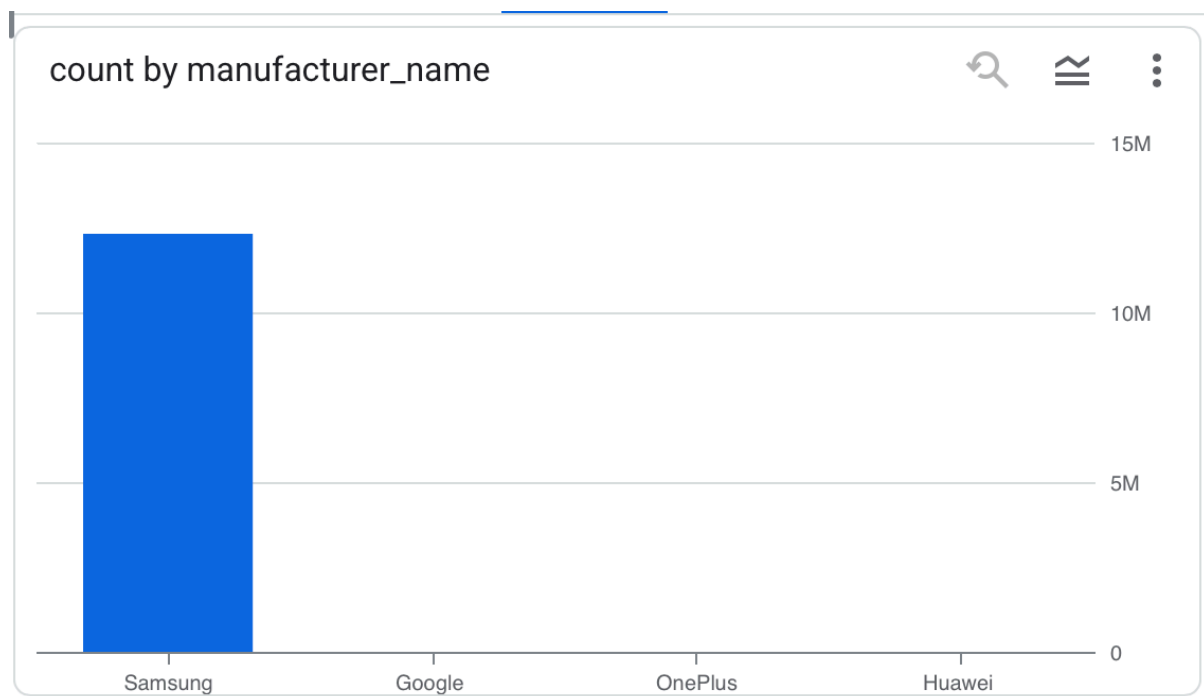
Top 10 are Samsung devices.

Row	device_name	count	min_os_version
1	Samsung Galaxy A54 5G	4653518	33
2	Samsung Galaxy A52 5G	3380897	31
3	Samsung Galaxy A53 5G	3027255	31
4	Samsung Galaxy A55 5G	797295	34
5	Samsung Galaxy A53 5G UW	554837	31
6	Samsung Galaxy A51	32620	29
7	Samsung Galaxy A34 5G	3662	33
8	Samsung Galaxy S24 Ultra	3533	34
9	Samsung Galaxy A15 5G	3016	34
10	Samsung Galaxy A52	2663	30

Most used Manufacturers

```
SELECT
  CASE
    WHEN STRPOS(device_name, ' ') > 0 THEN SUBSTR(device_name, 1,
STRPOS(device_name, ' ') - 1)
    ELSE device_name
  END AS `manufacturer_name`,
  COUNT(1) AS `count`,
  ROUND(
    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---
prod.firebase_performance.com_project_ANDROID`), 2
  ) AS `percent`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC
```

Row	manufacturer_name	count	percent
1	Samsung	12287327	99.98
2	Google	2806	0.02
3	OnePlus	247	0.0
4	Huawei	9	0.0

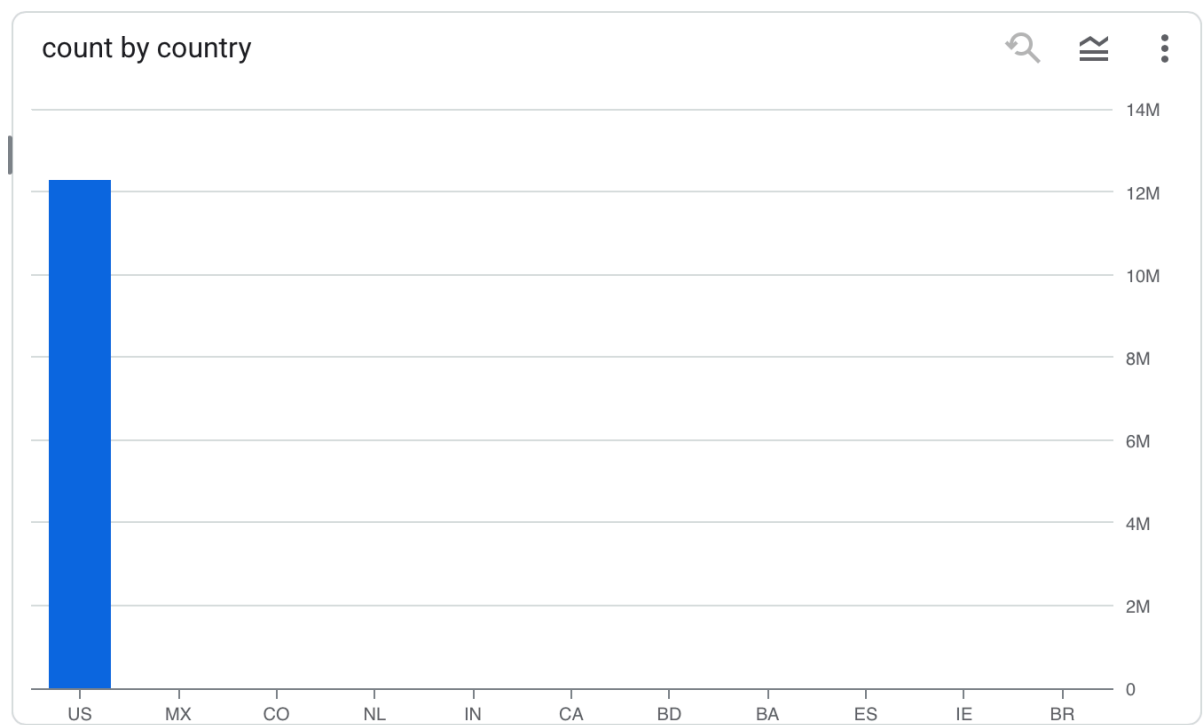


Samsung accounts for 99.9% of all devices used. So devs should focus on optimising for Samsung as it has some implementations slightly different from standard Android.

Country

```
SELECT
  country,
  COUNT(1) AS `count`,
  ROUND(
    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---
prod.firebase_performance.com_project_ANDROID`), 2
  ) AS `percent`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC
```

Row	country ▼	count ▼	percent ▼
1	US	12288544	99.98
2	MX	1513	0.01
3	CO	228	0.0
4	NL	28	0.0
5	IN	27	0.0
6	CA	14	0.0
7	BD	11	0.0
8	BA	9	0.0
9	ES	6	0.0
10	IE	5	0.0
11	BR	4	0.0



99.9 % usage is in US unsurprisingly.

Firestore uses the IP address of the device making the request to infer the **country, region, and city**, using standard IP geolocation services. If a user in India uses a VPN server located in Germany, Firestore will likely report the country as Germany for that session. This method respects user privacy because it doesn't use GPS or device-based location services—only the IP address, which is considered less precise and more privacy-preserving. This IP-based geolocation is part of the default behavior and can't be disabled selectively.

Top 3 are United States, Mexico, Colombia. It's interesting that this is being used in other countries as well as I thought Retail Outlet had stores only in US. Firebase's country detection is based on IP address and can be manipulated using a VPN.

Country Vs OS

```
SELECT
  country,
  COUNT(1) AS `count`,
  os_version
FROM
  `project---prod.firebaseio.com_project_ANDROID`
GROUP BY 1, 3
ORDER BY 2 DESC
```

Row	country	count	os_version
1	US	11679145	34
2	US	539788	31
3	US	238610	33
4	US	3002	30
5	US	1506	35
6	MX	1504	34
7	US	514	36
8	CO	144	33
9	CO	74	31
10	US	36	29

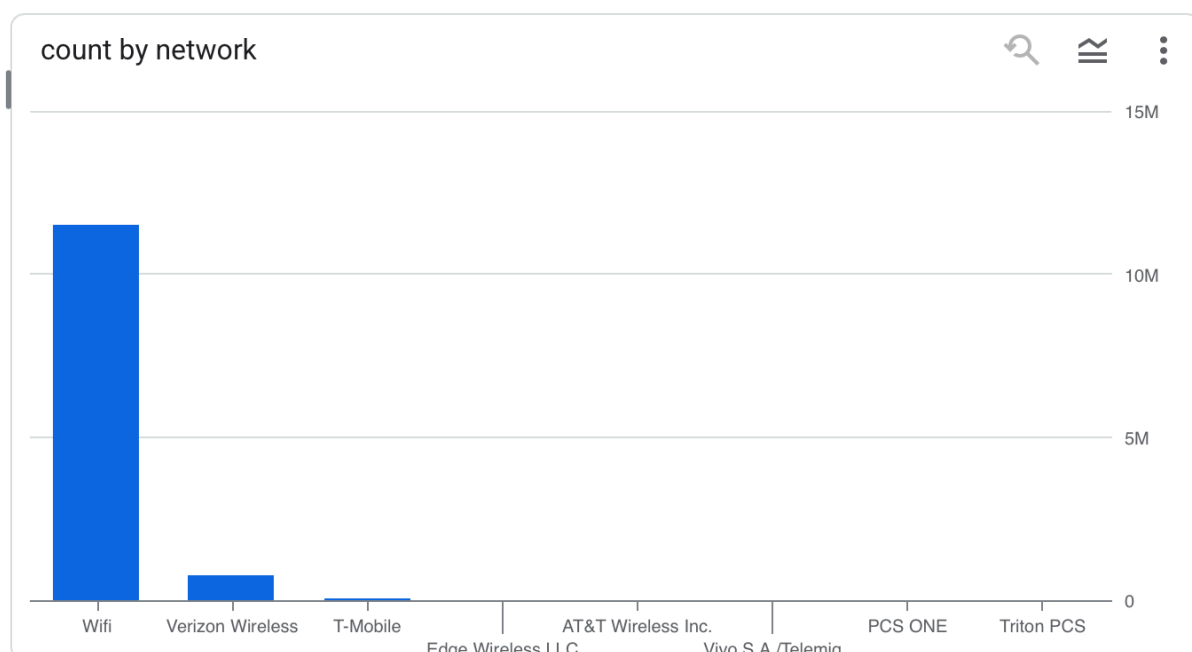
In US almost all devices are using Android 30 and above.

Carrier usage

```
SELECT
  CASE
    WHEN carrier = "[MISSING]" THEN "Wifi"
    ELSE carrier
  END AS network,
  COUNT(1) AS `count`,
  ROUND(
    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---prod.firebaseio.com_project_ANDROID`), 2
  ) AS `percent`
FROM
  `project---prod.firebaseio.com_project_ANDROID`
GROUP BY 1
```

ORDER BY 2 DESC

Row	network	count	percent
1	Wifi	11501520	93.58
2	Verizon Wireless	758526	6.17
3	T-Mobile	30222	0.25
4	Edge Wireless LLC	42	0.0
5	AT&T Wireless Inc.	32	0.0
6	Vivo S.A./Telemig	30	0.0
7	PCS ONE	9	0.0
8	Triton PCS	8	0.0



93.5% use Wifi to connect. Others used mobile network to connect. I am not sure why they did that given this is PROD data, I was under the assumption it will all be through Wifi.

Network Type usage

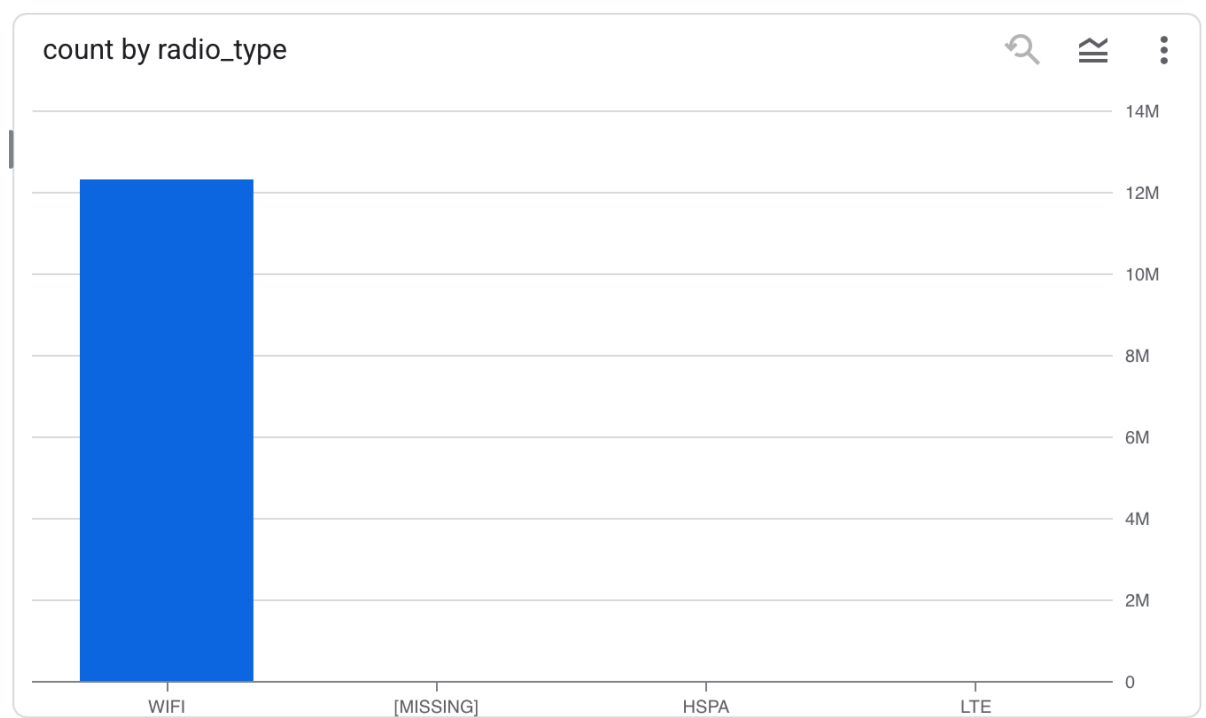
```
SELECT
  radio_type,
  COUNT(1) AS `count`,
  ROUND(
```

```

    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---
prod.firebase_performance.com_project_ANDROID`), 2
    ) AS `percent`
FROM
`project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC

```

Row	radio_type	count	percent
1	WIFI	12286746	99.97
2	[MISSING]	2912	0.02
3	HSPA	389	0.0
4	LTE	342	0.0



99.97 % are using Wifi to connect. However it's important to note 0.02% are using some other network.

The radio_type field typically refers to the **type of network connection** the user's device was on during the event, such as:

- wifi
- cell
- ethernet

- 4g, 5g, etc.

Missing Network Type

```
SELECT
  radio_type,
  COUNT(1) AS `count`,
  os_version,
  device_name
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1, 3, 4
HAVING
  radio_type LIKE "[MISSING]"
ORDER BY 2 DESC
```

Row	radio_type	count	os_version	device_name
1	[MISSING]	1042	34	Samsung Galaxy A52 5G
2	[MISSING]	676	34	Samsung Galaxy A53 5G
3	[MISSING]	651	34	Samsung Galaxy A54 5G
4	[MISSING]	237	31	Samsung Galaxy A53 5G UW
5	[MISSING]	130	34	Samsung Galaxy A55 5G
6	[MISSING]	60	31	Samsung Galaxy A52 5G
7	[MISSING]	59	33	Samsung Galaxy A53 5G UW
8	[MISSING]	48	33	Samsung Galaxy A52 5G
9	[MISSING]	24	33	Samsung Galaxy A51
10	[MISSING]	8	31	Samsung Galaxy A53 5G
11	[MISSING]	7	33	Samsung Galaxy A54 5G
12	[MISSING]	4	34	Google sdk_gphone64_arm64

Not sure when network would be missing.

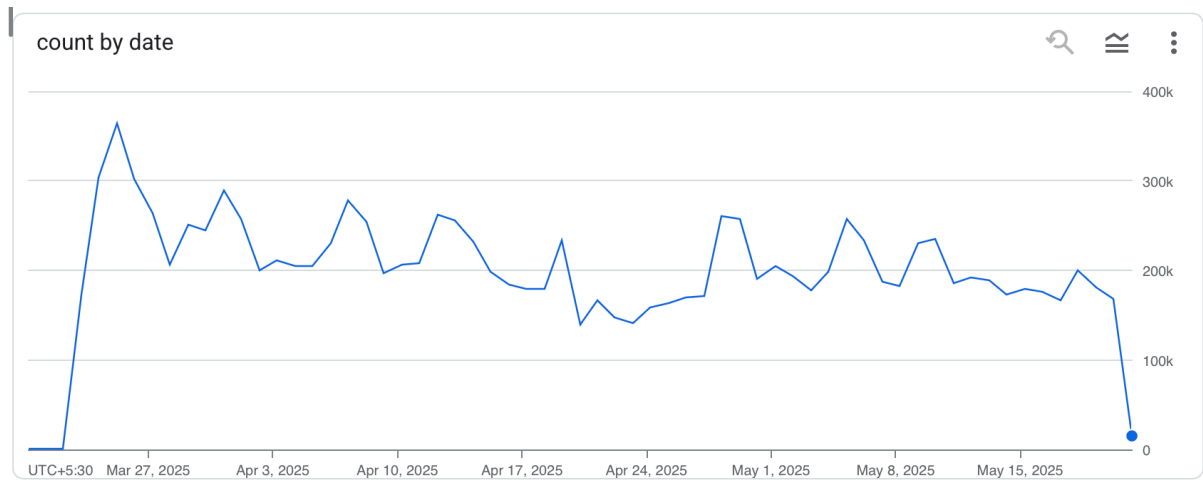
In BigQuery GA4 export data, [MISSING] in a field like radio_type means:

1. **The parameter was not set at all** in that event.
2. GA4/Firebase SDK didn't collect that data due to:
 - Platform limitations (e.g., iOS restrictions).
 - User permission settings (e.g., limited data collection).
 - App SDK not initialized properly at the time of event.
 - Offline events being batched and uploaded without network context.

Event Volume over time

```
SELECT
  EXTRACT(DATE FROM event_timestamp) AS `date`,
  COUNT(1) AS `count`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 1
```

Row	date ▼	count ▼
1	2025-03-20	4
2	2025-03-21	176
3	2025-03-22	1233
4	2025-03-23	172562
5	2025-03-24	303072
6	2025-03-25	364298
7	2025-03-26	302700
8	2025-03-27	264261
9	2025-03-28	206896
10	2025-03-29	251717

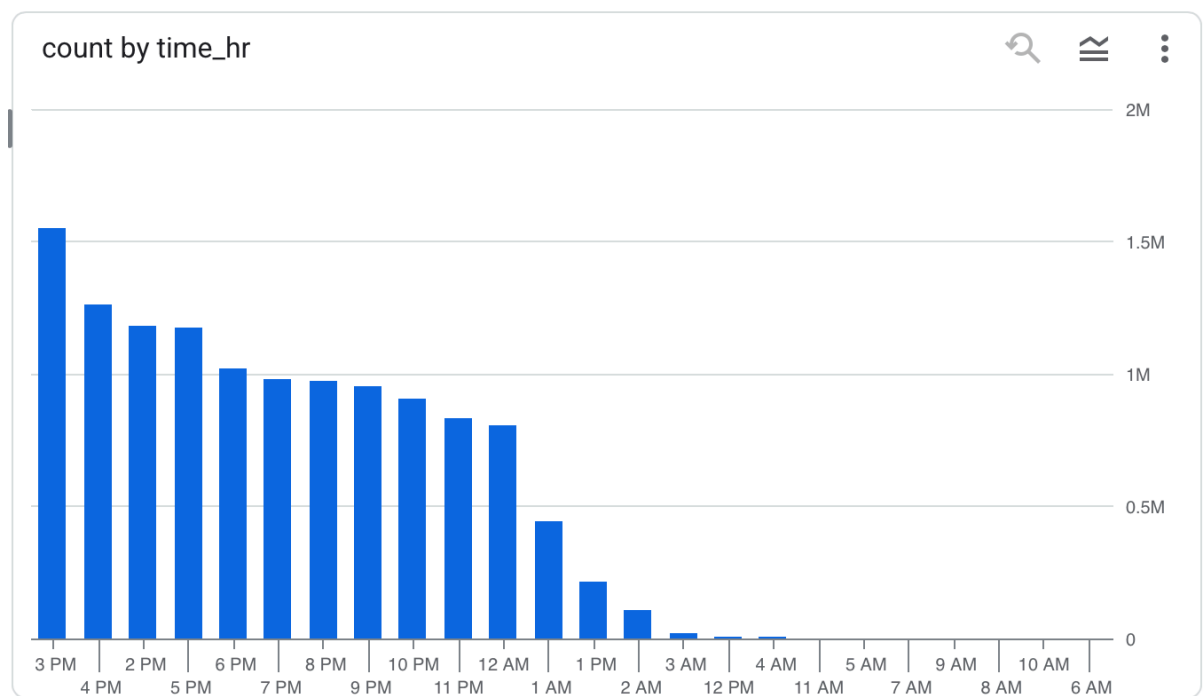


Peaked on March 25, 2025. Probably launch of a new store.

Peak usage period by Hour

```
-- %l:%M %p
-- %I = zero-padded 12-hour (e.g., 07)
-- %l = non-padded 12-hour (e.g., 7)
-- %p = AM/PM
-- %M = minutes (always zero-padded)
SELECT
  FORMAT_DATETIME('%l %p', DATETIME(event_timestamp)) AS `time_hr`,
  COUNT(1) AS `count`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC
```

Row	time_hr	count
1	3 PM	1549472
2	4 PM	1259799
3	2 PM	1179913
4	5 PM	1171373
5	6 PM	1021527
6	7 PM	983205
7	8 PM	974498
8	9 PM	952811
9	10 PM	904641
10	11 PM	830843



Peak usage is at 3PM.

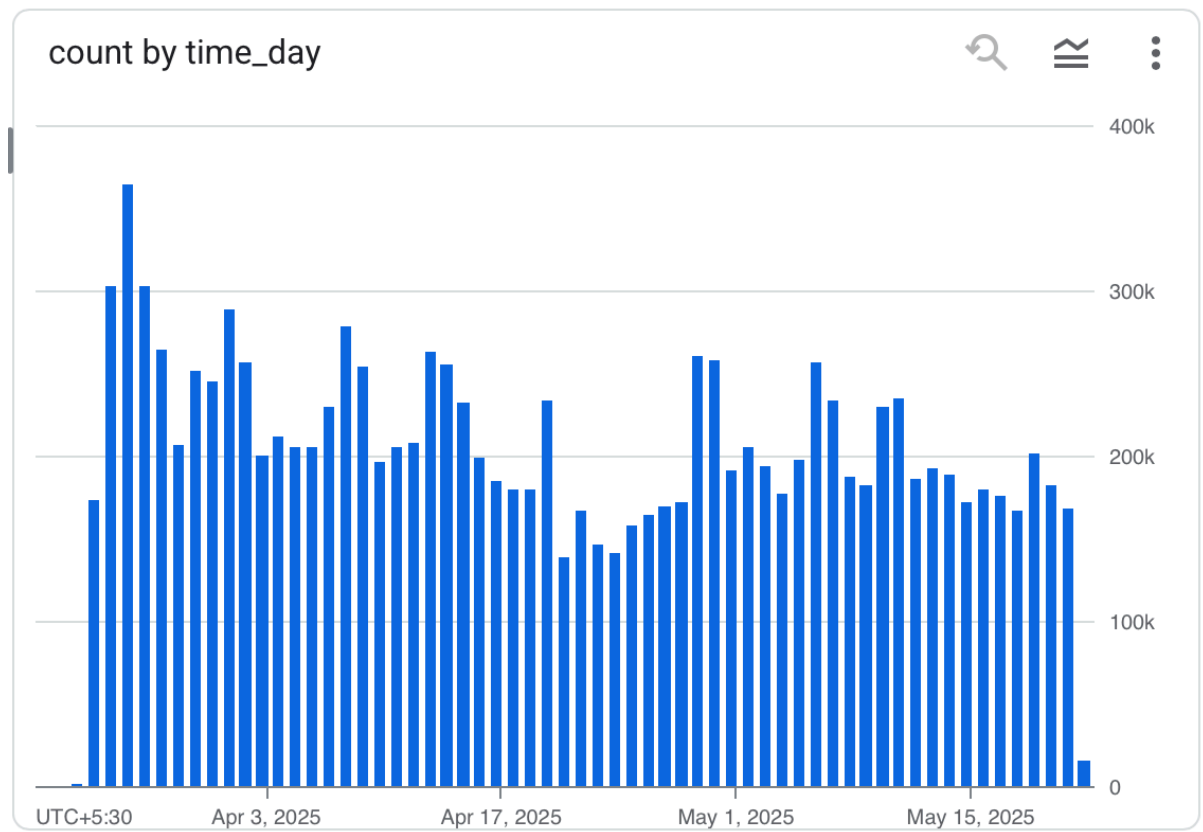
Peak usage period by Day

```

SELECT
  EXTRACT(DATE FROM event_timestamp) AS `time_day`,
  COUNT(EXTRACT(DATE FROM event_timestamp)) AS `count`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC

```

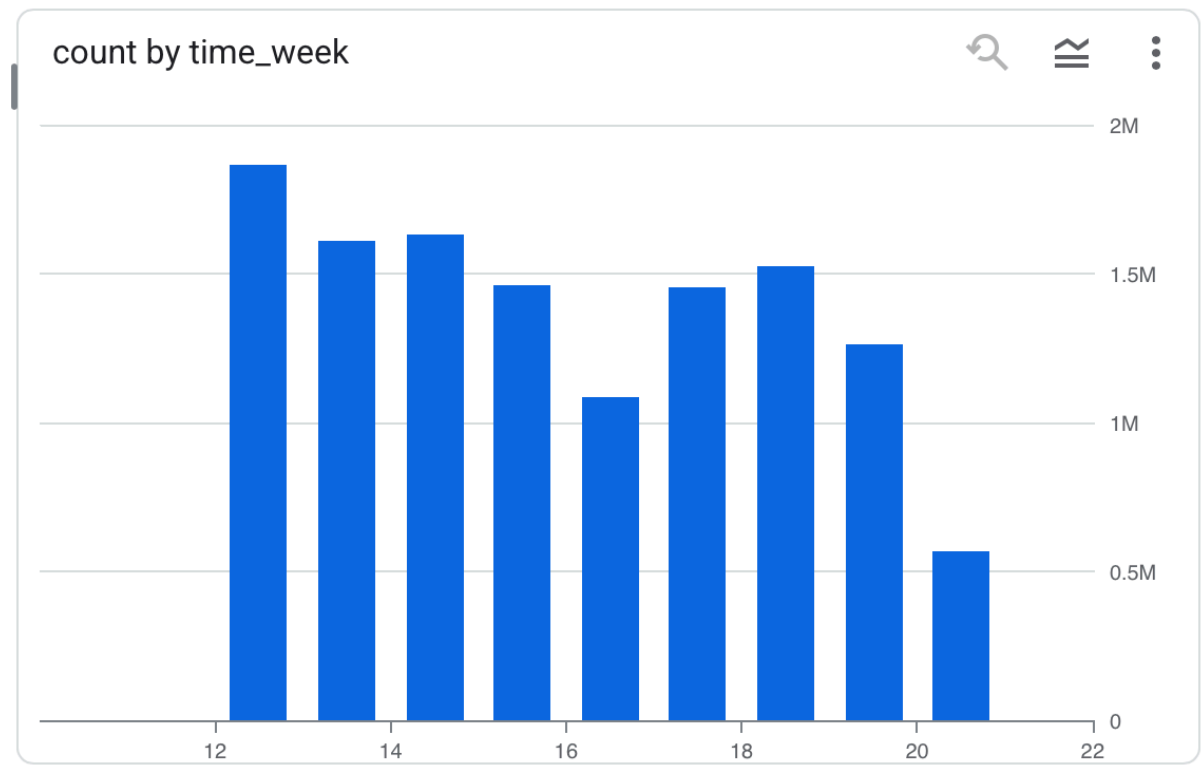
Row	time_day ▼	count ▼
1	2025-03-25	364298
2	2025-03-24	303072
3	2025-03-26	302700
4	2025-03-31	288919
5	2025-04-07	278526
6	2025-03-27	264261
7	2025-04-12	262387
8	2025-04-28	260369
9	2025-04-29	257743
10	2025-04-01	257022



Peak usage period by Week

```
SELECT
  EXTRACT(WEEK FROM event_timestamp) AS `time_week`,
  COUNT(EXTRACT(WEEK FROM event_timestamp)) AS `count`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC
```

Row	time_week	count
1	12	1865506
2	14	1634157
3	13	1612192
4	18	1523352
5	15	1461629
6	17	1456557
7	19	1260273
8	16	1084072
9	20	565318
10	11	1413



There are 52 weeks per year. It looks like it peak in the 12th week. Which is on March.

Event frequency

```
SELECT
  event_name,
  COUNT(1) AS `count`,
  ROUND(
    (COUNT(1) * 100) / (SELECT COUNT(*) AS `total` FROM `project---
prod.firebase_performance.com_project_ANDROID`), 2
  ) AS `percent`
FROM
  `project---prod.firebase_performance.com_project_ANDROID`
GROUP BY 1
ORDER BY 2 DESC
```

Row	event_name ▼	count ▼	percent ▼
1	orders	1928748	15.51
2	_app_in_foreground	1347324	10.83
3	_app_in_background	1218654	9.8
4	_st_MainActivity	1189735	9.57
5	/**	1164896	9.37
6	SESSION_DURATION	525978	4.23
7	DURATION	488546	3.93
8	_st_LaunchActivity	389187	3.13
9	_st_NavHostFragment	272755	2.19
10	io/**	230551	1.85

Interestingly 10% of the time App is in Background.