

Exhibit 8

U.S. Patent No. 9,647,918 (“’918 Patent”)

Accused Devices: Samsung Galaxy phones and tablets, and all versions and variations thereof.

Claim 1

Issued Claim(s)	Public Documentation
A wireless end-user device, comprising:	Samsung Galaxy phones and tablets are each “a wireless end-user device.” For example, the Galaxy S22 is a “wireless end-user device.” 
a wireless modem configurable to connect to a wireless network;	Samsung Galaxy phones and tablets comprise “a wireless modem configurable to connect to a wireless network.” For example, the Galaxy S22 comprises wireless modems which communicate with mobile service provider base stations to access a wireless wide area network (such as a GSM-type network and a Wi-Fi network).

	Network & Connectivity 5G 5G Non-Standalone (NSA), Standalone (SA), Sub6 / mmWave LTE Enhanced 4x4 MIMO, Up to 7CA, LTE Cat.20 Up to 2.0Gbps Download / Up to 200Mbps Upload Wi-Fi Wi-Fi 802.11 a/b/g/n/ac/ax 2.4G+5GHz+6GHz, HE160, MIMO, 1024-QAM Up to 2.4Gbps Download / Up to 2.4Gbps Upload Bluetooth Bluetooth® v 5.2, USB type-C, NFC, Location(GPS, Galileo, Glonass, BeiDou) Ultra Wide Band *Requires optimal connection. Actual speed may vary depending on country, carrier and user environment. *The bandwidths supported by the device may vary depending on the region or service provider. *Download and upload speeds reaching up to 2.4Gbps only available with Wi-Fi 6E. Wi-Fi 6E only supported on Galaxy S22 Ultra and S22+. Galaxy S22 has Wi-Fi 6. *Galileo and BeiDou coverage may be limited. BeiDou may not be available for certain countries. https://www.samsung.com/us/smartphones/galaxy-s22/models/.
a network stack configurable to receive and transmit data via the wireless modem and the wireless network;	Samsung Galaxy phones and tablets comprise “a network stack configurable to receive and transmit data via the wireless modem and the wireless network,” as shown by the below exemplary citations. For example, the Galaxy S22 uses a forked version of the Android operating system.

	Network Stack  Send feedback Network Stack is an updatable Mainline module that ensures that Android can adapt to evolving network standards and allows for interoperability with new implementations. For example, updates to captive portal detection and login code allow Android to stay up to date with changing captive portal models, and updates to APF allow Android to save power on Wi-Fi as new types of packets become common. Included components The Network Stack module contains the following components. On devices using the module, these services were moved to a different process and are accessed through a stable AIDL interface. • IP services. <code>IpClient</code> (formerly <code>IpManager</code>) is a component in charge of IP layer provisioning and maintenance. In Android 9, it was already used cross-process by components such as Bluetooth, and in-process by components such as Wi-Fi. <code>DhcpClient</code> obtains IP addresses from DHCP servers so they can be assigned to interfaces. • NetworkMonitor. The <code>NetworkMonitor</code> component tests for internet reachability when connecting to a new network or upon network failures, when detecting captive portals, and when validating networks. • Captive portal login app. A preinstalled app in charge of managing login on captive portals. This has been a separate app since Android 5.0, but it interacts with <code>NetworkMonitor</code> to forward some user choices to the system. https://source.android.com/devices/architecture/modular-system/networking.
a first network stack Application Programming Interface (API), containing at least one first call accessible to each of a plurality of device applications, the first network stack API callable by each of the plurality of device applications to open and use data packet flows via the network stack, the wireless modem, and the at least one wireless network;	Samsung Galaxy phones and tablets comprise “a first network stack Application Programming Interface (API), containing at least one first call accessible to each of a plurality of device applications, the first network stack API callable by each of the plurality of device applications to open and use data packet flows via the network stack, the wireless modem, and the at least one wireless network.” For example, applications running on the Galaxy S22 use the ConnectivityManager API, which receives requests from applications to access the wireless modem for data traffic.

ConnectivityManager

Added in API level 1

Kotlin | Java

```
public class ConnectivityManager
    extends Object
```

[java.lang.Object](#)
↳ [android.net.ConnectivityManager](#)

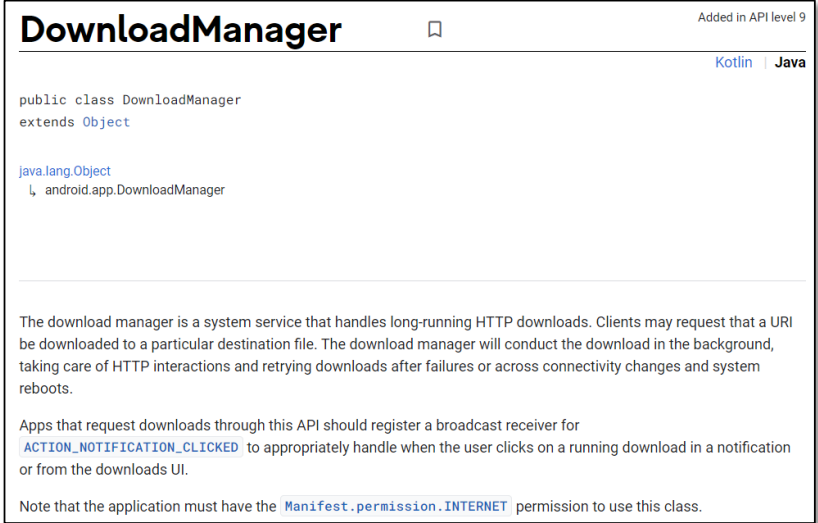
Class that answers queries about the state of network connectivity. It also notifies applications when network connectivity changes.

The primary responsibilities of this class are to:

1. Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
2. Send broadcast intents when network connectivity changes
3. Attempt to "fail over" to another network when connectivity to a network is lost
4. Provide an API that allows applications to query the coarse-grained or fine-grained state of the available networks
5. Provide an API that allows applications to request and select networks for their data traffic

<https://developer.android.com/reference/android/net/ConnectivityManager>;
<https://developer.android.com/develop/connectivity/network-ops/managing>;
<https://developer.android.com/develop/connectivity/network-ops/connecting>.

For further example, applications running on the Galaxy S22 use the DownloadManager API, which receives requests from applications to access the wireless modem for data traffic.

	 https://developer.android.com/reference/android/app/DownloadManager.
a second API containing at least one second call accessible to each of the plurality of device applications, the second API callable by each of the plurality of device applications to make a data transfer request for a media object associated with a network resource identifier supplied by the calling device application;	Samsung Galaxy phones and tablets comprise “a second API containing at least one second call accessible to each of the plurality of device applications, the second API callable by each of the plurality of device applications to make a data transfer request for a media object associated with a network resource identifier supplied by the calling device application,” as shown by the below citations.

	MediaPlayer overview  The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using <code>MediaPlayer</code> APIs. This document shows you how to use <code>MediaPlayer</code> to write a media-playing application that interacts with the user and the system in order to obtain good performance and a pleasant user experience. Alternatively, you might want to use <code>ExoPlayer</code>, which is a customizable open source library that supports high-performance features not available in <code>MediaPlayer</code>. Playing from a remote URL via HTTP streaming looks like this: <pre> Kotlin Java ----- val url = "http://....." // your URL here val mediaPlayer = MediaPlayer().apply { setAudioAttributes(AudioAttributes.Builder() .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC) .setUsage(AudioAttributes.USAGE_MEDIA) .build()) setDataSource(url) prepare() // might take long! (for buffering, etc) start() } </pre> https://developer.android.com/guide/topics/media/mediaplayer.html.
a media service manager prompted by the second call, to manage network data transfers for the media object by interfacing with the network stack to retrieve the media object associated with the network resource identifier via the	Samsung Galaxy phones and tablets comprise “a media service manager prompted by the second call, to manage network data transfers for the media object by interfacing with the network stack to retrieve the media object associated with the network resource identifier via the wireless modem and the wireless network,” as shown by the below citations.

wireless modem and the wireless network; and

MediaPlayer overview

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using `MediaPlayer` APIs.

This document shows you how to use `MediaPlayer` to write a media-playing application that interacts with the user and the system in order to obtain good performance and a pleasant user experience. Alternatively, you might want to use `ExoPlayer`, which is a customizable open source library that supports high-performance features not available in `MediaPlayer`.

Playing from a remote URL via HTTP streaming looks like this:

Kotlin Java

```
val url = "http://....." // your URL here
val mediaPlayer = MediaPlayer().apply {
    setAudioAttributes(
        AudioAttributes.Builder()
            .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
            .setUsage(AudioAttributes.USAGE_MEDIA)
            .build()
    )
    setDataSource(url)
    prepare() // might take long! (for buffering, etc)
    start()
}
```

<https://developer.android.com/guide/topics/media/mediaplayer.html>.

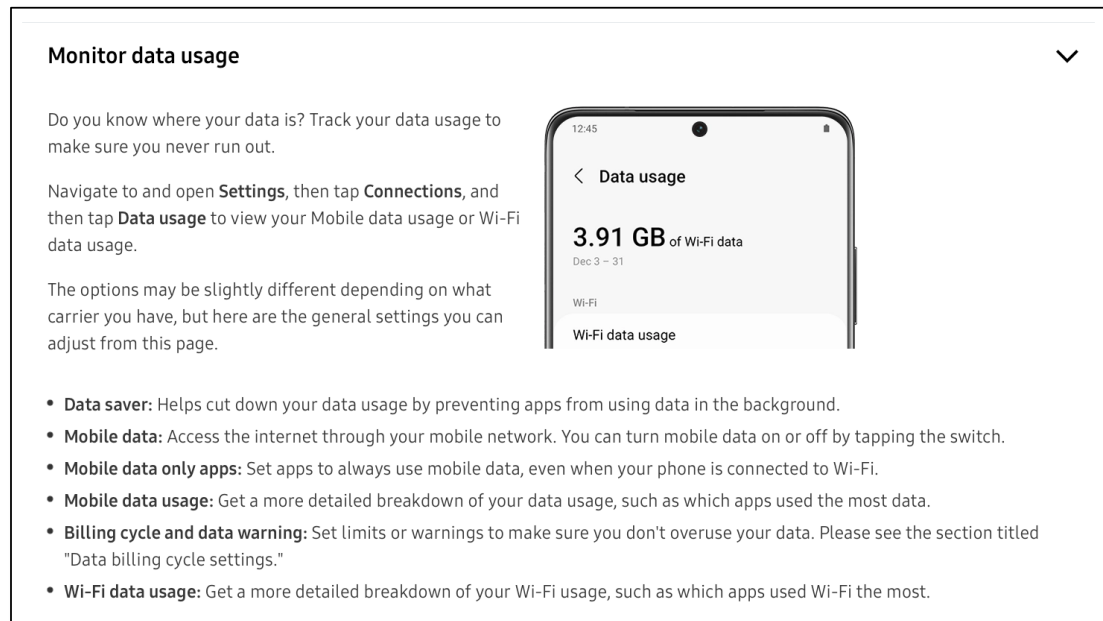
one or more service classification and measurement agents to associate wireless network data usage for the media object network data transfers with the device application that requests the data transfer for the media object, to associate wireless network data usage for respective data

Samsung Galaxy phones and tablets comprise “one or more service classification and measurement agents to associate wireless network data usage for the media object network data transfers with the device application that requests the data transfer for the media object, to associate wireless network data usage for respective data packet flows opened and used via the first network stack API with the device application opening such respective data packet flow, and to reconcile wireless network data usage for each of the plurality of device applications to

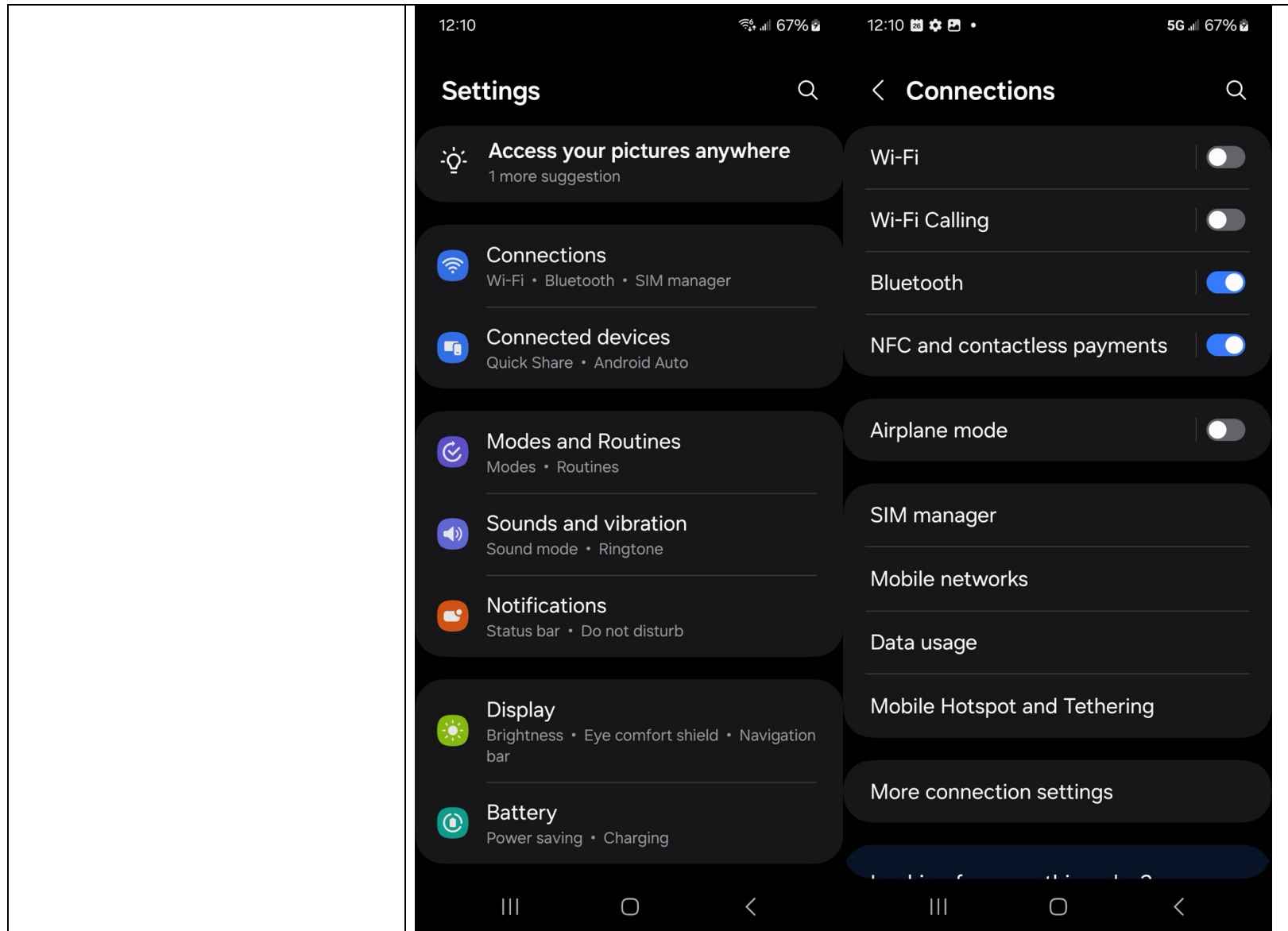
packet flows opened and used via the first network stack API with the device application opening such respective data packet flow, and to reconcile wireless network data usage for each of the plurality of device applications to track an aggregate wireless network data usage attributable to each of the plurality of device applications via both the first network stack API and the second API.

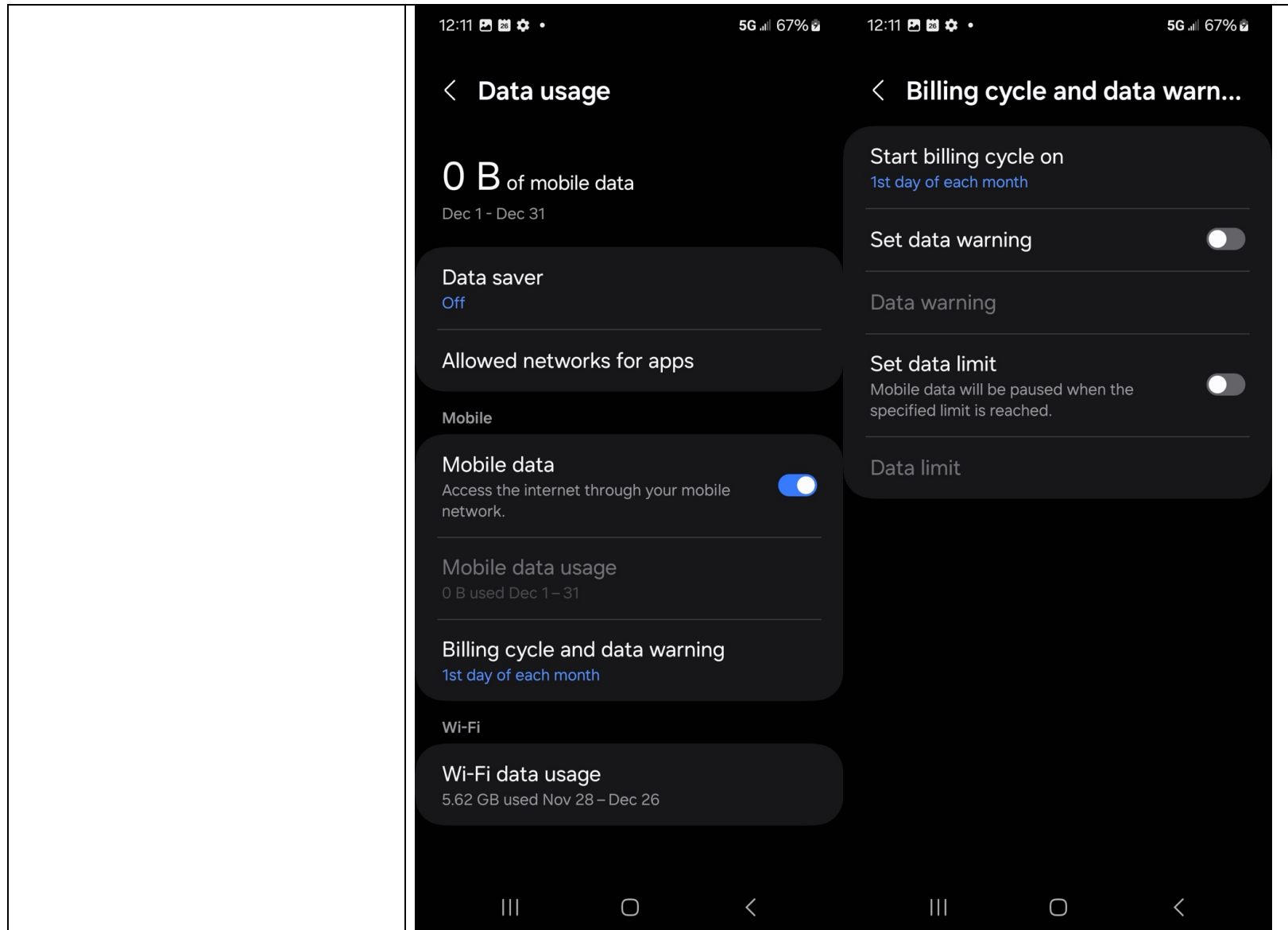
track an aggregate wireless network data usage attributable to each of the plurality of device applications via both the first network stack API and the second API.”\

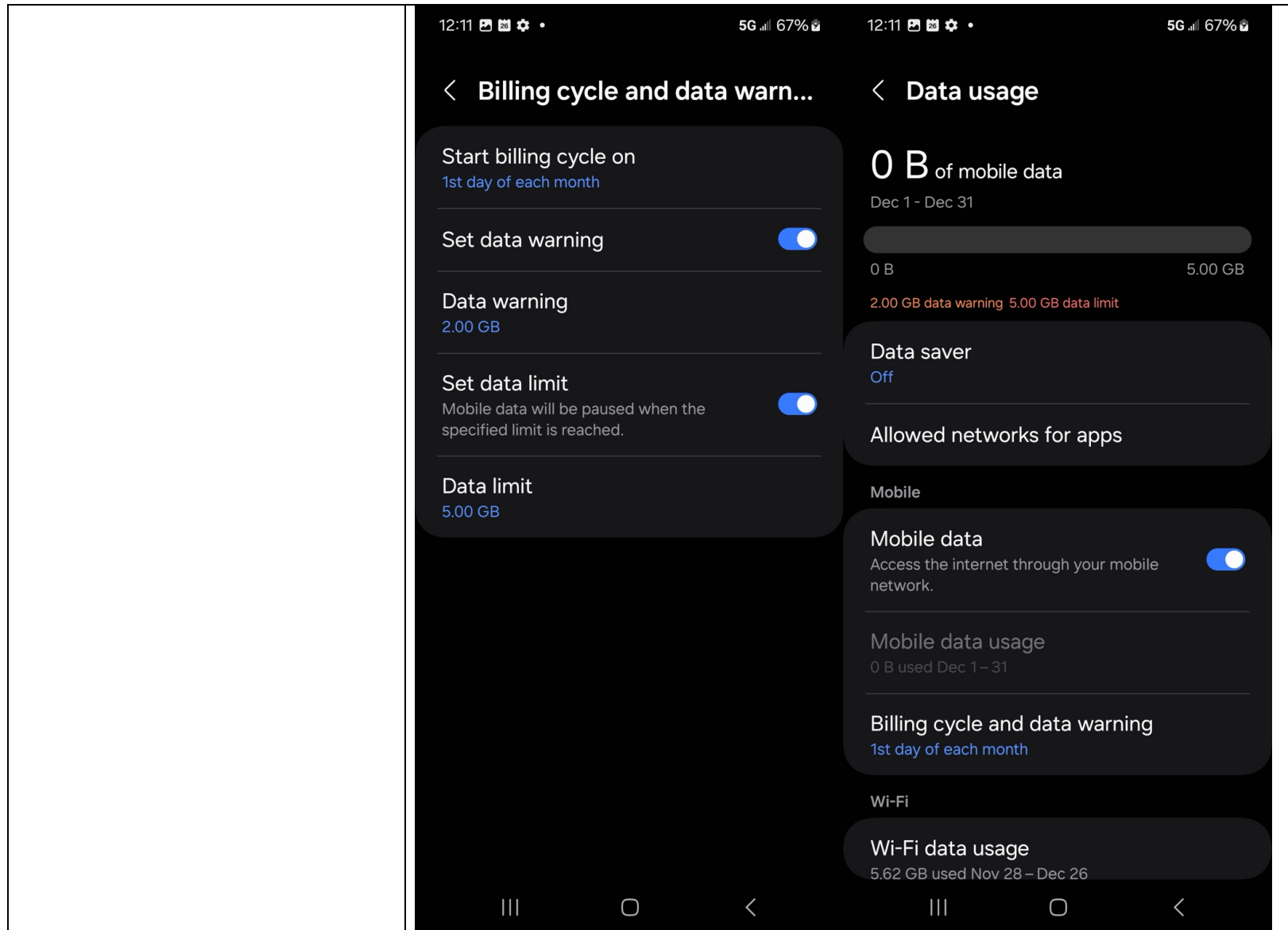
For example, Samsung devices classify and measure data usage on a per-application basis:

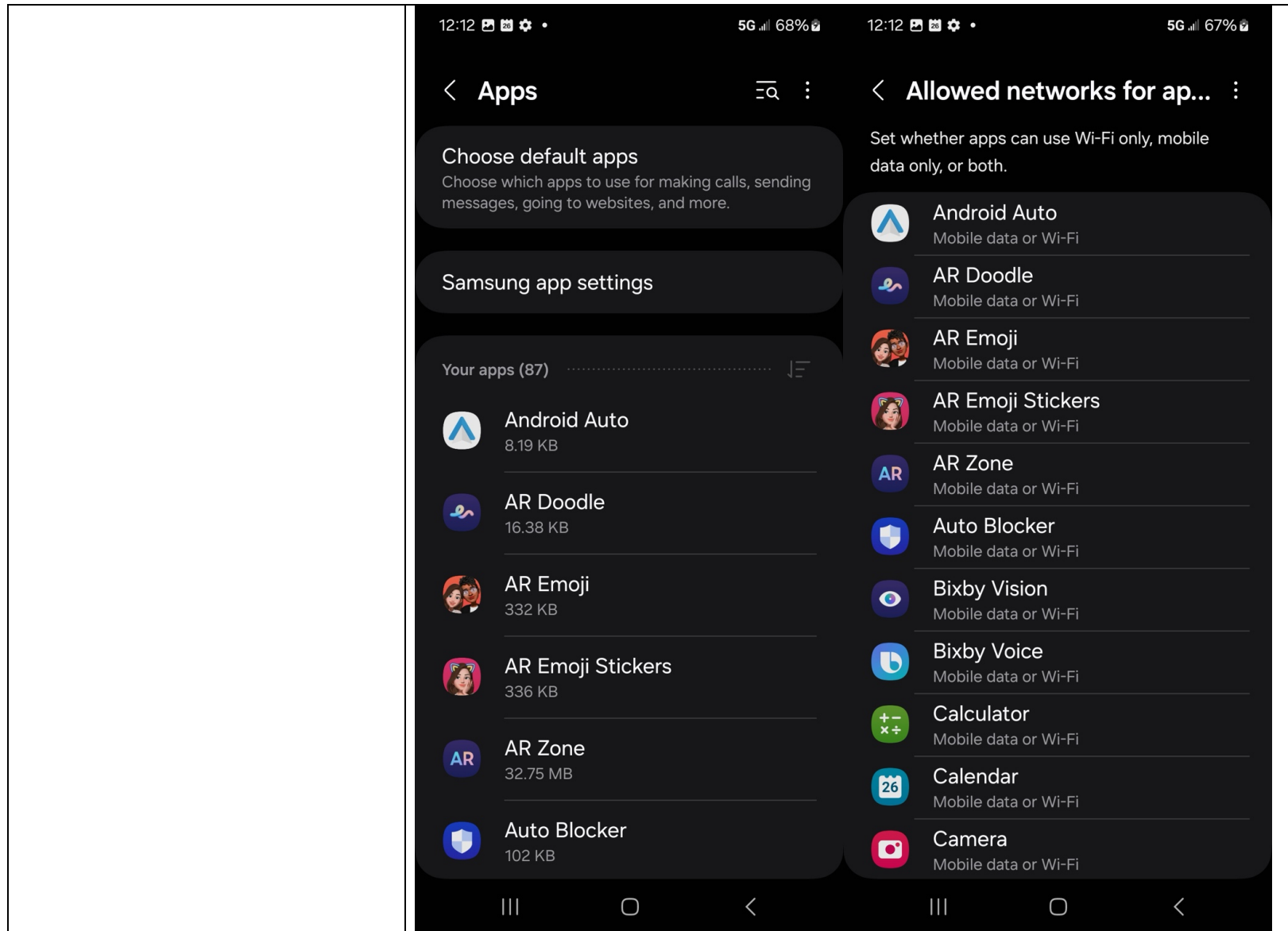


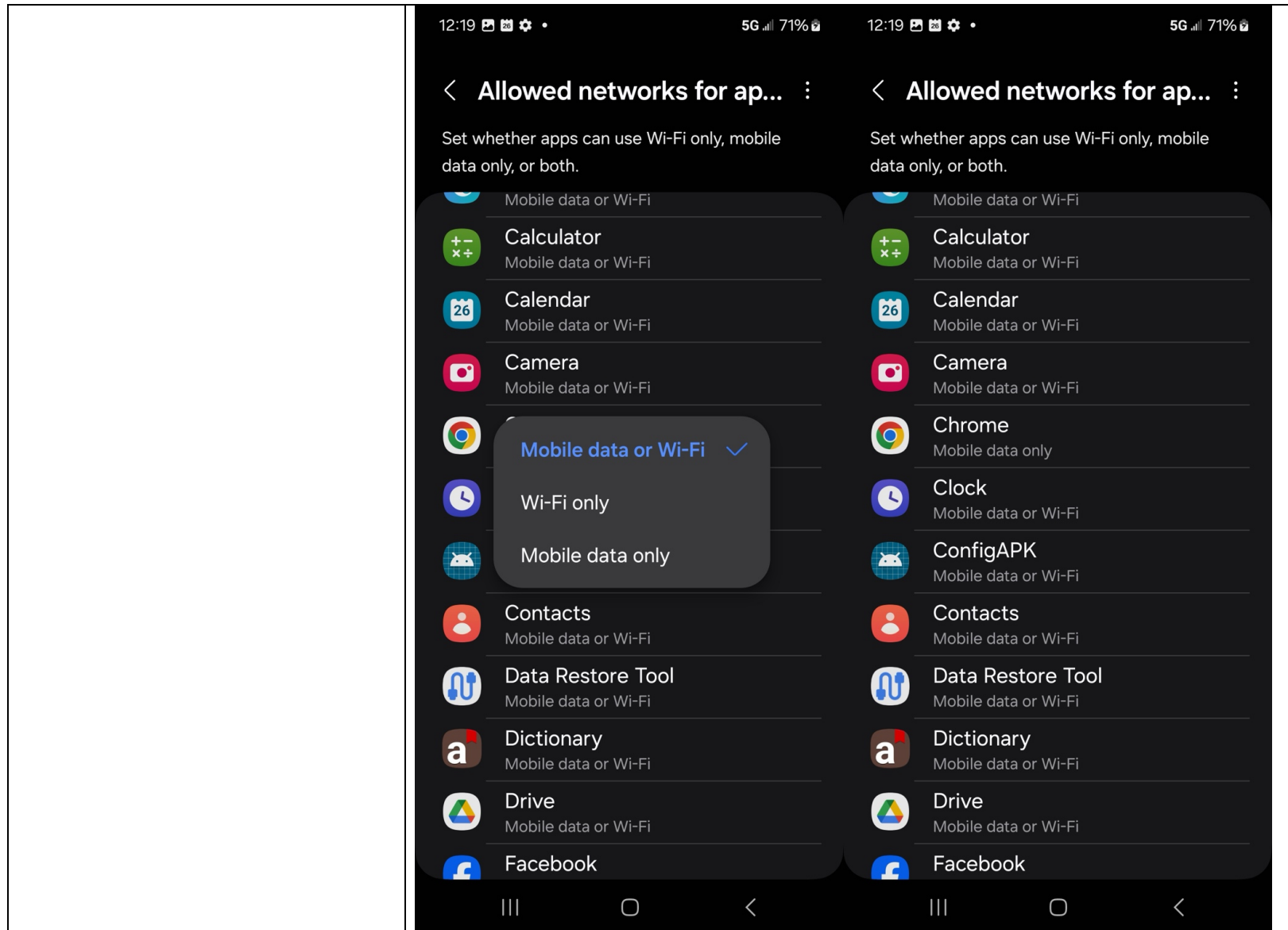
<https://www.samsung.com/us/support/answer/ANS00079018/>;

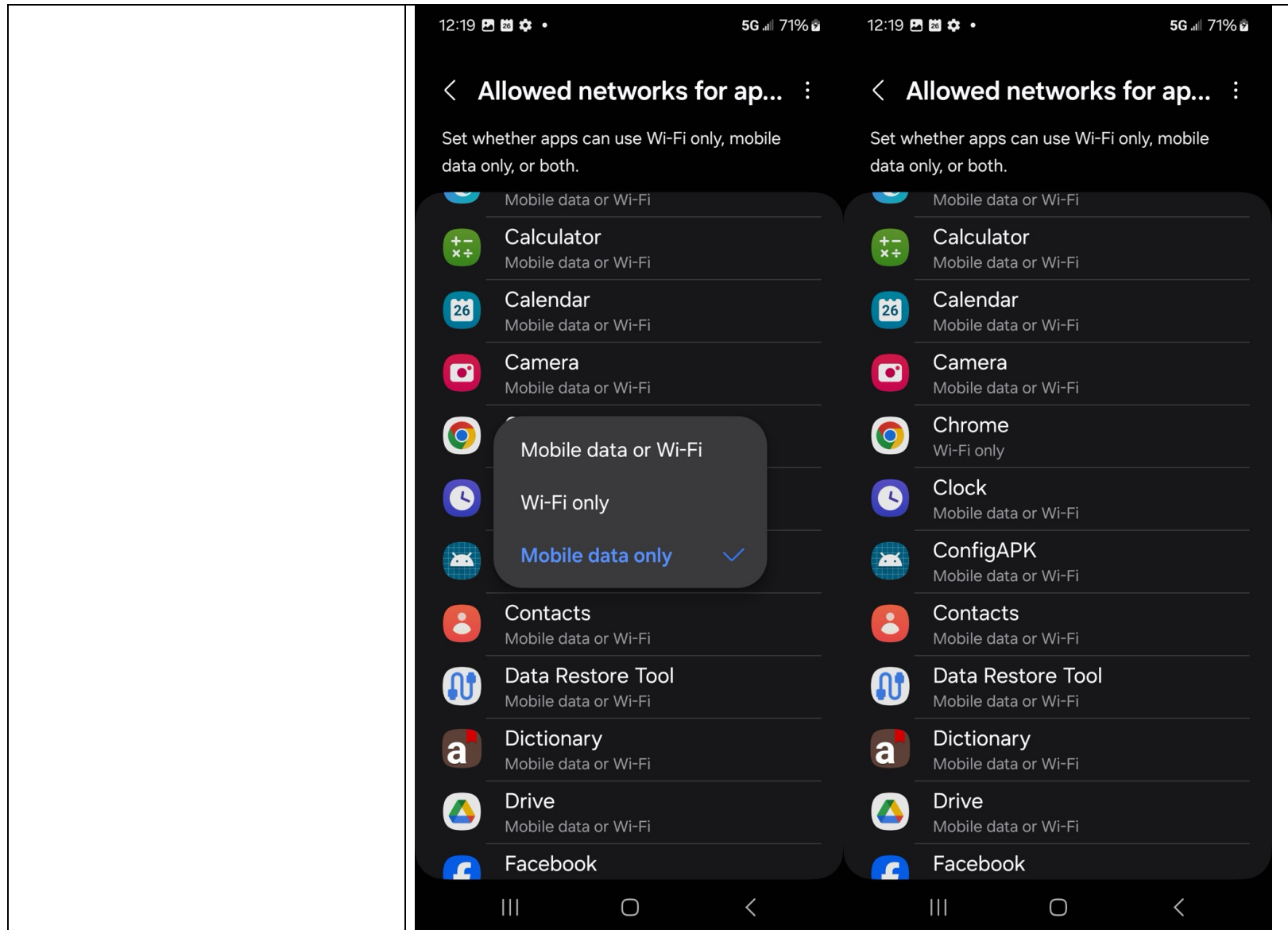


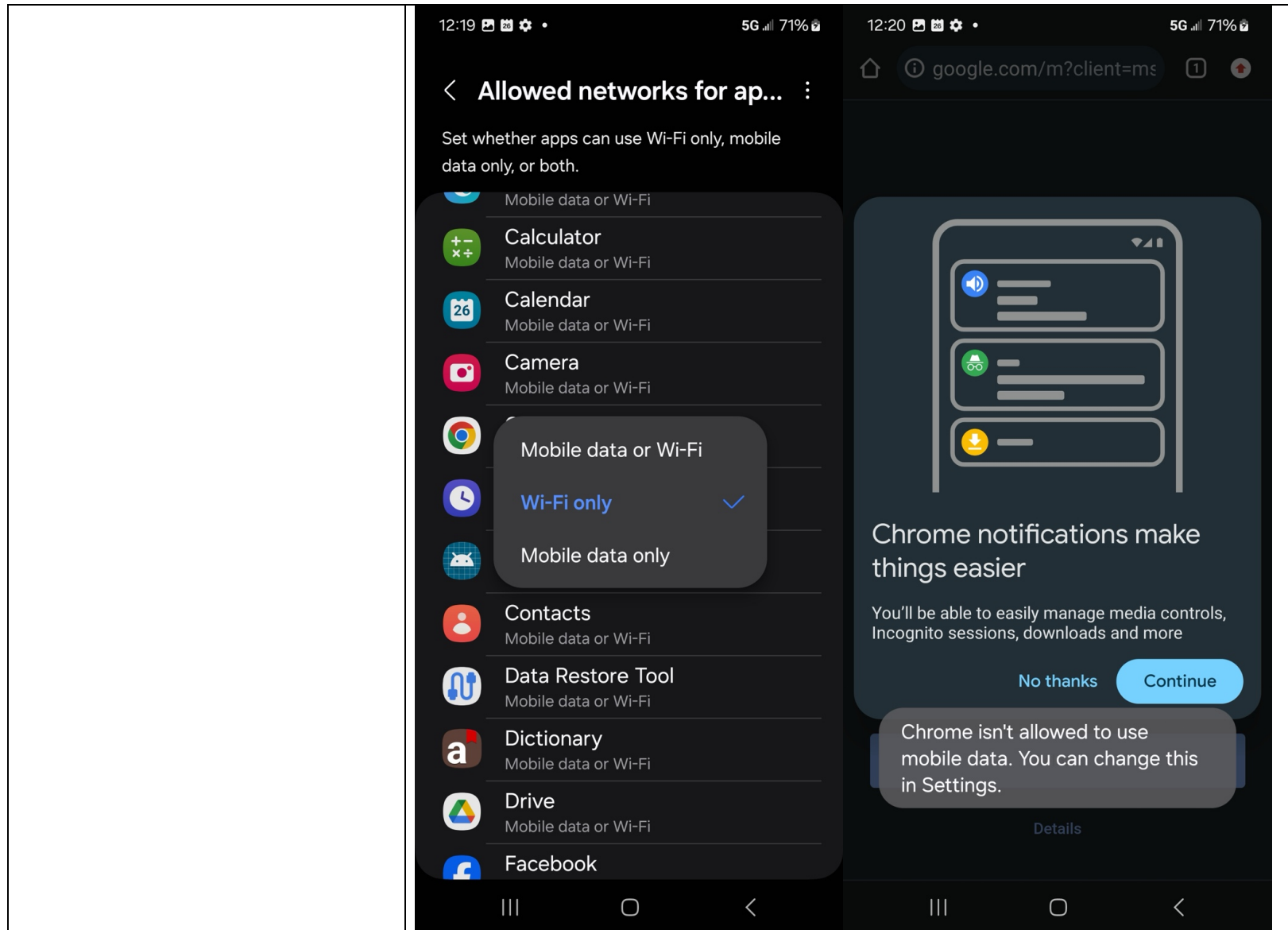


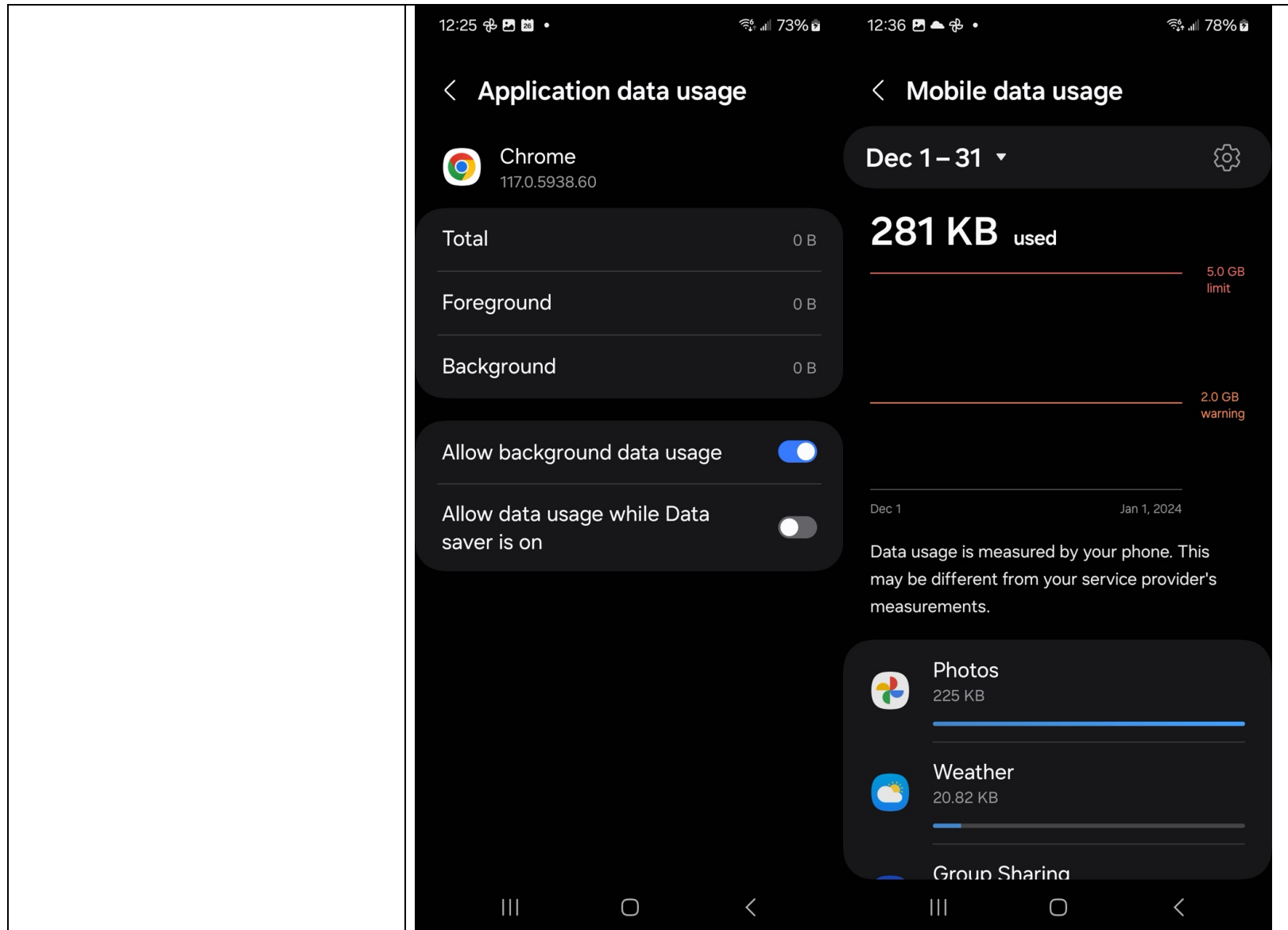












Android Developers > Develop > Reference

Was this helpful?

TrafficStats

Added in API level 8

[Kotlin](#) | [Java](#)

```
public class TrafficStats
    extends Object
```

[java.lang.Object](#)
↳ [android.net.TrafficStats](#)

Class that provides network traffic statistics. These statistics include bytes transmitted and received and network packets transmitted and received, over all interfaces, over the mobile interface, and on a per-UID basis.

These statistics may not be available on all platforms. If the statistics are not supported by this device, **UNSUPPORTED** will be returned.

Note that the statistics returned by this class reset and start from zero after every reboot. To access more robust historical network statistics data, use [NetworkStatsManager](#) instead.

<https://developer.android.com/reference/android/net/TrafficStats>;

Android Developers > Develop > Reference

Was this helpful?  

Added in API level 23

NetworkStats

[Kotlin](#) | **Java**

```
public final class NetworkStats
extends Object implements AutoCloseable
```

java.lang.Object

↳ android.app.usage.NetworkStats

Class providing enumeration over buckets of network usage statistics. `NetworkStats` objects are returned as results to various queries in `NetworkStatsManager`.

<https://developer.android.com/reference/android/app/usage/NetworkStats>;

Android Developers > Develop > Reference

Was this helpful?  

NetworkStatsManager

Added in API level 23

Kotlin | Java

```
public class NetworkStatsManager
    extends Object
```

[java.lang.Object](#)
↳ [android.app.usage.NetworkStatsManager](#)

Provides access to network usage history and statistics. Usage data is collected in discrete bins of time called 'Buckets'. See [NetworkStats.Bucket](#) for details.

Queries can define a time interval in the form of start and end timestamps (Long.MIN_VALUE and Long.MAX_VALUE can be used to simulate open ended intervals). By default, apps can only obtain data about themselves. See the below note for special cases in which apps can obtain data about other applications.

<https://developer.android.com/reference/android/app/usage/NetworkStatsManager>;

In the case of applications that provide network data transfer as a service, such as the download manager, media streaming service, etc, it is possible to attribute the ownership of the network data transfer to the UID of the requesting application using the `TrafficStats.setThreadStatsUid()` function call. The caller must hold the "android.permission.MODIFY_NETWORK_ACCOUNTING" permission to re-assign the ownership of the network traffic.

<https://source.android.com/devices/tech/datausage/kernel-overview>;
<https://source.android.com/docs/core/data/tags-explained>;

AOSP > Docs > Core Topics

Was this helpful?  

eBPF traffic monitoring

The eBPF network traffic tool uses a combination of kernel and user space implementation to monitor network usage on the device since the last device boot. It provides additional functionality such as socket tagging, separating foreground/background traffic and per-UID firewall to block apps from network access depending on phone state. The statistics gathered from the tool are stored in a kernel data structure called `eBPF maps` and the result is used by services like `NetworkStatsService` to provide persistent traffic statistics since the last boot.

<https://source.android.com/docs/core/data/ebpf-traffic-monitor>;

AOSP > Docs > Core Topics

Was this helpful?  

Network interface statistics overview

In Android 4.0, statistics reported by Linux network interfaces are recorded over time, and are used to enforce network quota limits, render user-visible charts, and more.

Each network device driver (Wi-Fi included) must follow the standard kernel device lifecycle, and return correct statistics through `dev_get_stats()`. In particular, statistics returned must remain strictly monotonic while the interface is active. Drivers may reset statistics only after successfully completing an `unregister_netdev()` or the equivalent that generates a `NETDEV_UNREGISTER` event for callbacks registered with `register_netdevice_notifier()` / `register_inetaddr_notifier()` / `register_inet6addr_notifier()`.

Mobile operators typically measure data usage at the Internet layer (IP). To match this approach in Android 4.0, we rely on the fact that for the kernel devices we care about the `rx_bytes` and `tx_bytes` values returned by `dev_get_stats()` return exactly the Internet layer (IP) bytes transferred. But we understand that for other devices it might not be the case. For now, the feature relies on this peculiarity. New drivers should have that property also, and the `dev_get_stats()` values must not include any encapsulation overhead of lower network layers (such as Ethernet headers), and should preferably not include other traffic (such as ARP) unless it is negligible.

The Android framework only collects statistics from network interfaces associated with a `NetworkStateTracker` in `ConnectivityService`. This enables the framework to concretely identify each network interface, including its type (such as `TYPE_MOBILE` or `TYPE_WIFI`) and subscriber identity (such as IMSI). All network interfaces used to route data should be represented by a `NetworkStateTracker` so that statistics can be accounted correctly.

<https://source.android.com/docs/core/data/iface-overview>;