2024 FCC Mobile Speed Test App Technical Description

Office of Engineering and Technology
April 2024

MOZARK

Table of Contents

List of Figures and Tables

I. Introduction

The FCC Mobile Speed Test App ("the App")[1] is a free mobile application available in the United States, designed to evaluate the performance of mobile broadband service. Data from crowdsourced speed tests and network performance challenges play a crucial role in supporting the FCC's Broadband Data Collection program and helps identify where broadband is and is not available across the country.[2]

This technical description document for the App provides detailed information on the metrics and methodologies used to conduct connected performance tests on mobile broadband networks. Specifically, this document reviews the measurement techniques; the test sequence and testing methods for download, upload, latency, packet loss, and jitter; environmental data collection for Android and iOS operating systems; data reporting features; and provides an overview of the backend test system architecture.

II. Mobile Broadband Performance Testing

This section covers the FCC Mobile Speed Test App's system architecture technical features, as well as other technical aspects of the methods used to measure mobile broadband network performance.

A. FCC Mobile Speed Test App Methodology Overview

The App's technologies and methodologies are developed collaboratively with Mozark PTE. Ltd., a technical solutions contractor. The contractor supports the FCC with the App's development and maintenance.

The FCC Mobile Speed Test App consists of two separate mobile applications, due to the variations between the iOS and Android operating systems, security features, and hardware. The iOS App is written in the Swift programming language, while the Android version is written using the Java and Kotlin programming languages.

[1] The FCC Mobile Speed Test App is available for Android devices from Google Play, at https://play.google.com/store/apps/details?id=com.agence3pp.fcc&pli=1 and for iOS devices from the Apple App Store, at https://apps.apple.com/us/app/fcc-mobile-speed-test/id6470025404

[2] See generally https://www.fcc.gov/BroadbandData.

The Android operating system is supported on hardware devices from many different vendors such as Google, Samsung, Motorola, and Nokia. The iOS operating system is supported on hardware devices from Apple. A list of radio parameters collected for each App is available in section A.2. Android devices support the radio parameters that the FCC Mobile Speed Test App collects so that the BDC system and FCC staff can verify the network environment. Both the Android and Apple devices collect speed test data for the challenge and crowdsource tests.

Due to limitations in the iOS operating system, the App running on iPhones, currently, cannot be configured to perform automated background testing.

1. Measurement Process

The measurements providing the underlying data depend on both the measurement client, which is responsible for initiating the testing process, and the measurement servers, which serve as the endpoints for the client's measurements. The measurement client is embodied in the publicly accessible FCC Mobile Speed Test App, which is available free of charge, on Android and iOS devices throughout the nation. The measurement client and measurement servers enable measurement tests to be conducted on the networks of any mobile broadband service provider to which consumers are subscribed while using the App.

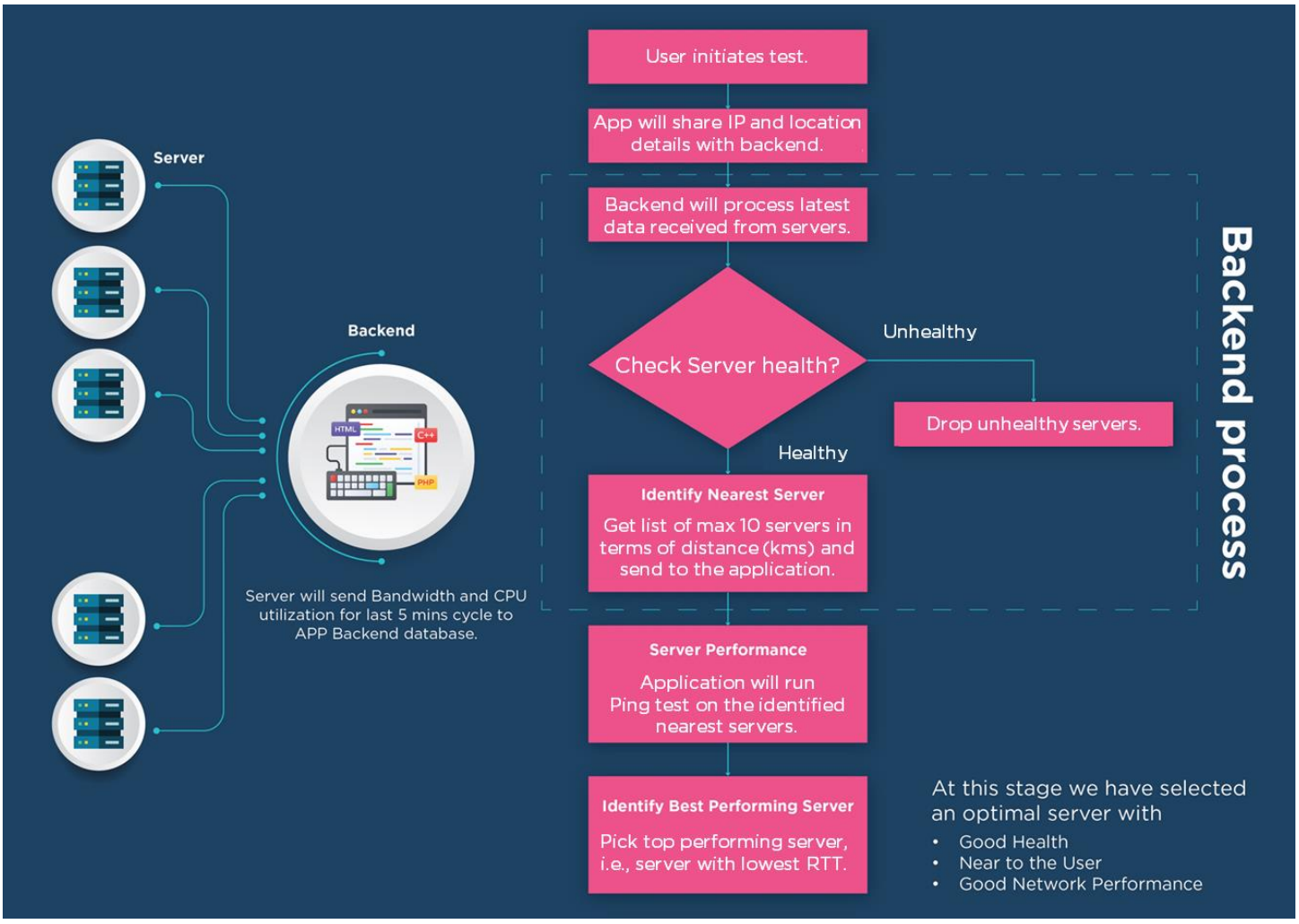
The measurement servers are hosted by carrier-neutral providers such as Amazon Web Services (AWS) and Google Cloud Platform (GCP). These servers are strategically positioned across the United States.

The App uses a speed testing methodology that focuses on the performance of the specific mobile broadband network being tested. When a test starts, the App will check if the connection is cellular or Wi-Fi. If the connection is over a cellular network, the App will further identify its connection technology type (i.e., 3G, 4G, or 5G). If the connection is over a Wi-Fi network, the App will test and show the performance of the Wi-Fi network connection through its connected internet service for informational purposes only.

Once the App selects a measurement server, all performance metrics are derived from traffic exchanged between the App and the selected measurement server. The measurement server chosen is the optimal server, determined by obtaining a list of 10 servers that are both in good health and in proximity to the App, based on IP address approximation. Subsequently, each server in the candidate list undergoes an individual ping using an Internet Control Message Protocol (ICMP) message, and the server providing the lowest round-trip latency in its response is designated as the chosen server. Please note that this selected server may not be the server that is geographically closest to the device running the App, but it is the optimal server based on current network conditions. As a result, the metrics measure performance along a specific path within each mobile broadband provider's network, through the point of interconnection.

2. Performance Testing and Metrics

Figure 1: Test Server Selection Process



Once the server is selected, the App performs the following active tests of mobile broadband performance:

- **Download speed:** Measures the download speed of three independent connections in megabits per second over an 8-second time interval to transmit up to 1,000 megabytes (MB). The first 3-second period is considered a warmup period and is not included in the calculation (i.e., 3 seconds warmup and 5 seconds testing).

- **Upload speed:** Measures the upload speed of three independent connections in megabits per second over an 8-second time interval to transmit up to 1,000 megabytes (MB). The first 3-second period is considered a warmup period and is not included in the calculation (i.e., 3 seconds warmup and 5 seconds testing).

- **Latency:** Measures the average round-trip time in milliseconds of up to 200 UDP (User Datagram Protocol) data packets that are successfully acknowledged as received within 2 seconds. The packets are sent over a 3-second time interval with a 2-second acknowledgement time window.

- **Packet Loss:** The percentage equal to the number of the latency packets not acknowledged within 2 seconds divided by the number of total latency packets sent.

- **Jitter:** The variation in latency, measured in milliseconds, is calculated with the acknowledged latency test packets using the Packet Delay Variation (PDV) approach (see section II.D. Figure 6).

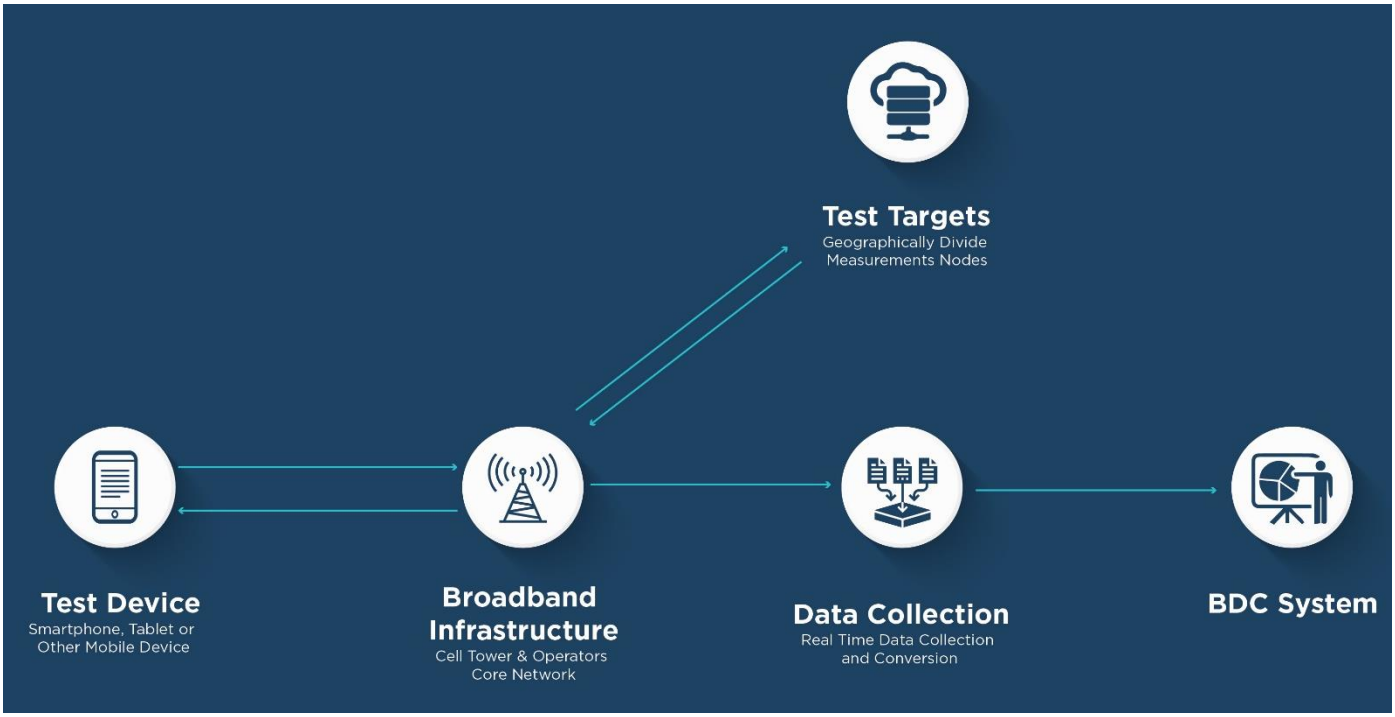
B. Test System Architecture

The test system architecture focuses on measuring the performance of mobile broadband networks. The carrier-neutral measurement servers are distributed geographically close to major internet exchange points (IXP) throughout the United States and its territories. This chosen measurement server architecture allows for unbiased network performance measurements.

As illustrated below in Figure 2, the App executes tests as part of the measurement system that comprises a distributed network of "Test Devices" (i.e., measurement clients) used to accurately measure the performance of mobile broadband connections. Upon completion of each test sequence, the App reports measurement results to the backend server ("Data Collection"). The App measures download and upload speeds, round-trip latency, packet loss, and jitter, by exchanging information with the selected measurement servers ("Test Targets"), which the App contacts when a Test Device starts a test sequence.

The maximum capacity of each measurement server is 10 Gigabits per second (Gbps). Measurement servers are located across the U.S. to enable a measurement client to select the measurement server with the least latency.

Figure 2: Mobile Broadband Test Architecture



C. Download Speed and Upload Speed

Two distinct tests are employed to measure download and upload speeds in megabits per second. These tests establish multiple TCP connections to execute HTTPS GET and POST requests directed to the selected test node. The primary goal of the test is to assess download or upload throughput by simultaneously downloading or uploading data through three concurrent byte streams.

The download test involves hosting a randomly generated, and uncompressible, file with a small amount of data on a web server located at the target test node. Throughout this test, the server measures the number of bytes transferred by each thread and the elapsed time from server to device using chunks of data downloaded from the selected measurement server. The actual test time utilized in calculations is a minimum of 5 seconds and a warmup period of about 3 seconds. Each speed test session concludes after 8 seconds (including warmup period).

The upload test entails transferring a randomly generated, and uncompressible, file from the device to the web server. The number of bytes transferred by each thread and the elapsed time are closely monitored during this test. The data is monitored and measured in the form of chunks uploaded to the selected measurement server. Like the download test, the actual test time used in calculations is a minimum of 5 seconds and a warmup period of about 3 seconds. Each speed test session concludes after 8 seconds (including the warmup period).

To gather test traffic data, the measurement process will disregard any data chunks that were entirely received or transmitted before the end of the initial 3-second warmup period. It is important to note the test will not come to an abrupt stop at the precise 8-second mark, and any data chunks already initiated at that point will be allowed to gracefully terminate. Each payload's data is both transmitted and received in multiple chunks, and comprehensive records are maintained for the bytes and times associated with each individual chunk.

In this context, each connection keeps track of the number of payload bytes transferred between two specific points in time and calculates the speed of each thread by dividing the number of bytes transferred by the number of seconds within the active test window. These individual thread speeds are then aggregated to calculate the total speed. Each individual thread is confirmed to ensure the 3-second warmup period is completed before the 5-second test time begins.

The following is an example of the calculation performed for a multi-connection

download test using three concurrent TCP connections:

S = Speed (Bytes per second)
B = Bytes (Bytes transferred)
T = Time (Seconds) (between start time point and end time point)
S1 = B1 / T1 (speed for Thread 1 calculation)
S2 = B2 / T2 (speed for Thread 2 calculation)
S3 = B3 / T3 (speed for Thread 3 calculation)
Speed = S1 + S2 + S3
Example values from a 3 MB payload:
B1 = 3077360 T1 = 15.583963
B2 = 2426200 T2 = 15.535768
B3 = 2502120 T3 = 15.536826
S1 = B1/T1 = 197469.668017
S2 = B2/T2 = 156168.655454
S3 = B3/T3 = 161044.475879
S1 + S2 + S3 = Total Throughput of the line = 197469.668017 + 156168.655454 + 161044.475879 = 514682 (bps) * 0.000008 = 4.12 Mbps

Figure 3: Warmup and Testing cycle for 1 single thread for Download

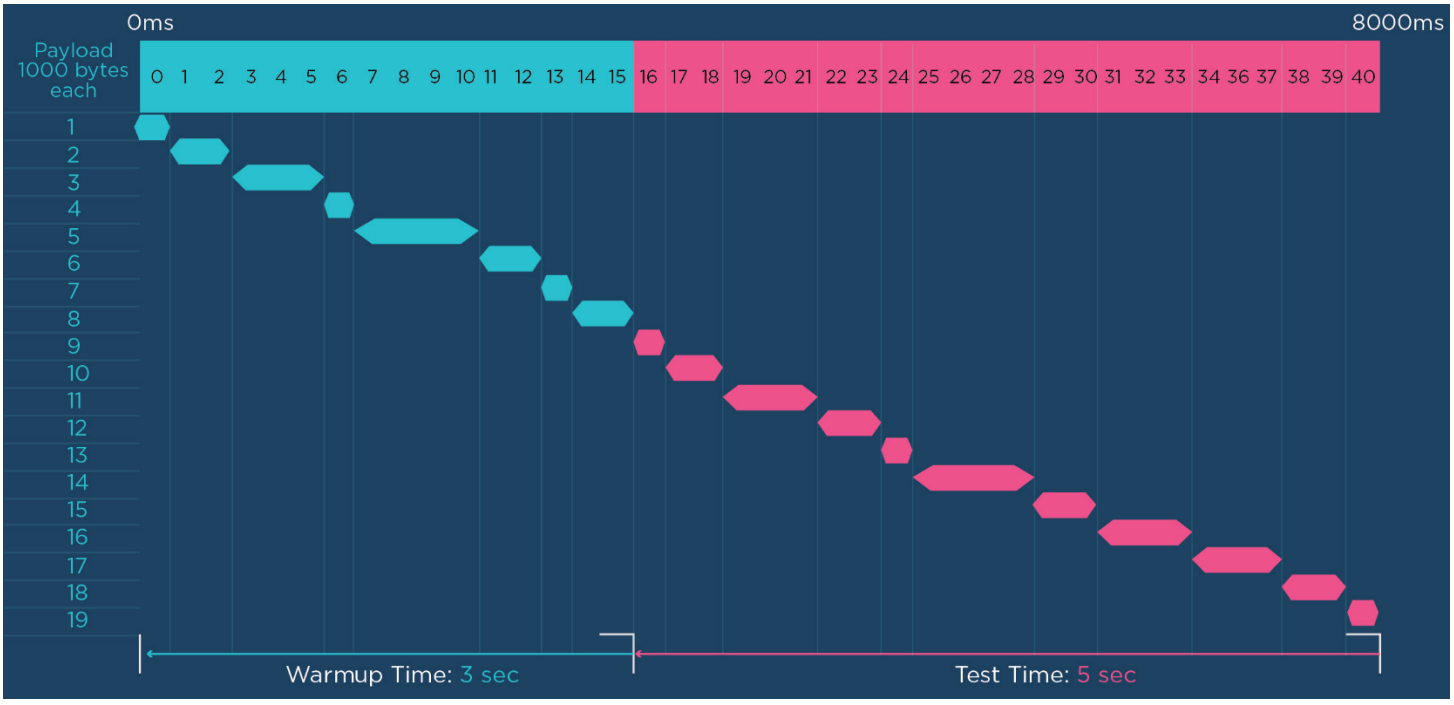
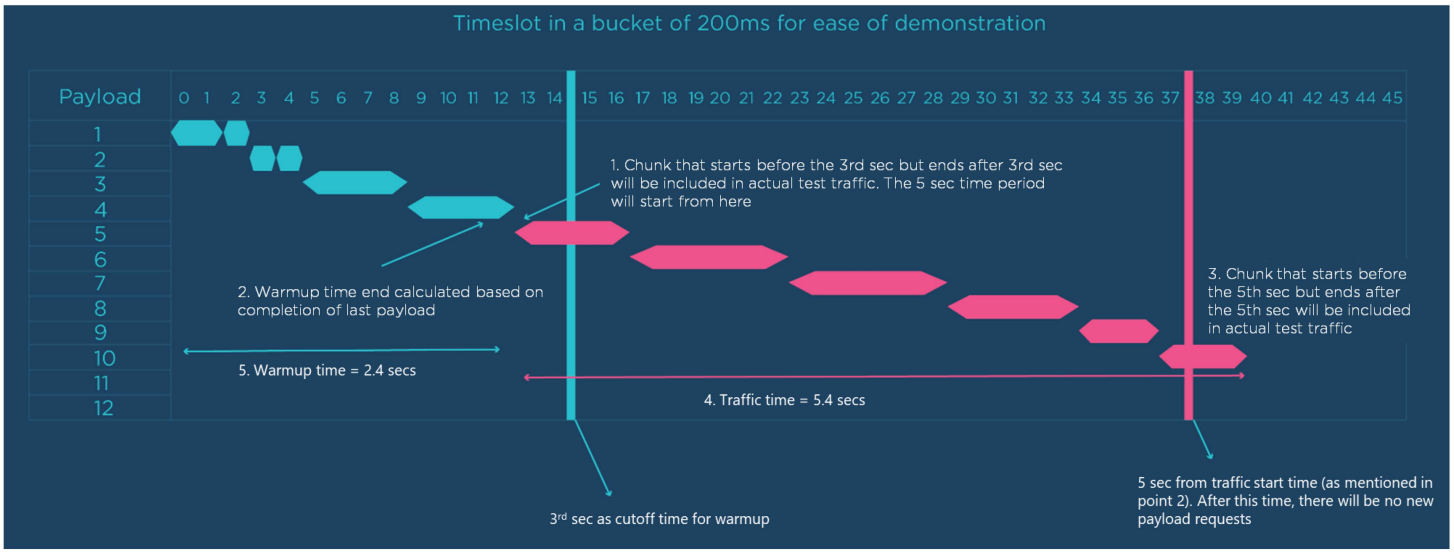


Figure 4: Warmup and Testing cycle for 1 single thread for Upload



The following pseudo-code describes the algorithm of the test in more detail:

Download test:

Get download test configuration from portal:
transferMaxTime
warmupMaxTime
testTimeout = transferMaxTime + warmupMaxTime
filename
nbThreads

Get download link from closest server: DownloadLink = closestServer.link +ressource (dl/)+ filename
Start download operations based on number of threads

```
for i in 0.. < nbThreads {
    customID = "Thread-\(i + 1)"
    startDownloadOperation(url: DownloadLink, customID: customID)
}
Func didDownloadDataCallback(nbOfBytesDownloaded)
{

// we create slot object and we store in it the last numberOfBytesDownloded + operationIdetifier +
currentElapsedTime
    slot = Slot(nbOfBytesDownloaded, OperationID, elapsedTime)
    slotArray.add(slot)
    totalBytesDownloaded = totalBytesDownloaded + nbOfBytesDownloaded
    If elapsedTime > timeout or totalBytesDownloaded <= 1000 MB {
        stopAllDowloadOperationInProgress()}
Func didFinishDownloadingFile(error, OperationID)
{
        If error = null {
            If elapsedTime > timeout or totalBytesDownloaded <= 1000 MB {
                stopAllDowloadOperationInProgress()
            }
            Else {
                // start new operation for the same threadID (which it finish downloading file)
                Op = operation.getOperation(operationID)
                startNewOperation(op.threadId)
            }
           }
        CalculateSpeed() {
            slotsPerThread = slotArray.filter(threadId)
```

```
    if elapsedTime <= testTimeout {
        trafficTime = transferMaxTime // 5 sec
    } else { // elapsedTime > 8
        trafficTime = elapsedTime - warmupMaxTime
    }
    for slot in slotsPerThread {
        // get bytes downloaded in the last 5 sec
        threadTime += taskTime
        if elapsedTime - slot.time <= trafficTime {
        threadTrafficBytes += slot.bytes
            if elapsedTime - prevSlot.time < trafficTime && prevSlot.time > WarmUpTime {
                threadTrafficBytes += slot.bytes
                timeToAdd = (WarmUpTime - slot.time)
            }
        }
        if threadTime <= trafficTime // (5 sec) {
            threadTrafficTime = threadTime
        }
        else {
            threadTrafficTime = (threadTime - WarmUpTime) + timeToAdd
        }
         threadWarmupBytes = allThreadBytes - threadTrafficBytes
        threadWarmupTime = allThreadTrafficTime - threadTrafficTime
        threadTrafficSpeed = threadTrafficBytes / threadTrafficTime
        threadWarmupSpeed = threadWarmupBytes / threadWarmupTime
        allThreadSpeed = allThreadSpeed + threadTrafficSpeed // (displayed in gauge in Mbps)
        totalWarmupSpeed = totalWarmupSpeed + threadWarmupSpeed
        trafficBytes = trafficBytes + threadTrafficBytes
        warmupBytes = warmupBytes + threadWarmupBytes
    }
    trafficTime = trafficBytes / allThreadSpeed       // ( in Bytes / sec)
    warmupTime = warmupBytes / totalWarmupSpeed
    // trafficBytes => bytesTransferred
    // warmupBytes => warmuptransferred
    // trafficTime => traffic duration
    // warmupTime => warmup duration
}
```

Upload test:

Get upload test configuration from portal:

transferMaxTime
warmupMaxTime
testTimeout = transferMaxTime + warmupMaxTime
filename
nbThreads

Get upload link from closest server: uploadLink = closestServer.link + ressource (upload.php)
Start upload operations based on number of threads

```
for i in 0.. < nbThreads {
    numberOfTask = numberOfTask + 1
    customID = "Thread-\(i + 1)"
    startUploadWithStreamOperation(url: uploadLink, customID: customID)
}
Func didUploadDataCallback(nbOfBytesUploaded)
 {
    // we create slot object and we store in it the last numberOfBytesUploaded + operationIdetifier +
currentElapsedTime
    slot = Slot(nbOfBytesUploaded, OperationID, elapsedTime)
    slotArray.add(slot)
    totalBytesUploaded = totalBytesUploaded + nbOfBytesUploaded
}
Func didFinishUploadingFile(error, OperationID)
{
        If error = null {
            numberOfTaskFinished = numberOfTaskFinished + 1
            If(elapsedTime > timeout || totalBytesUploaded <= 1000 MB) {
                If numberOfTaskFinished = numberOfTask
                { stopAllUploadedOperationInProgress()
                }
```

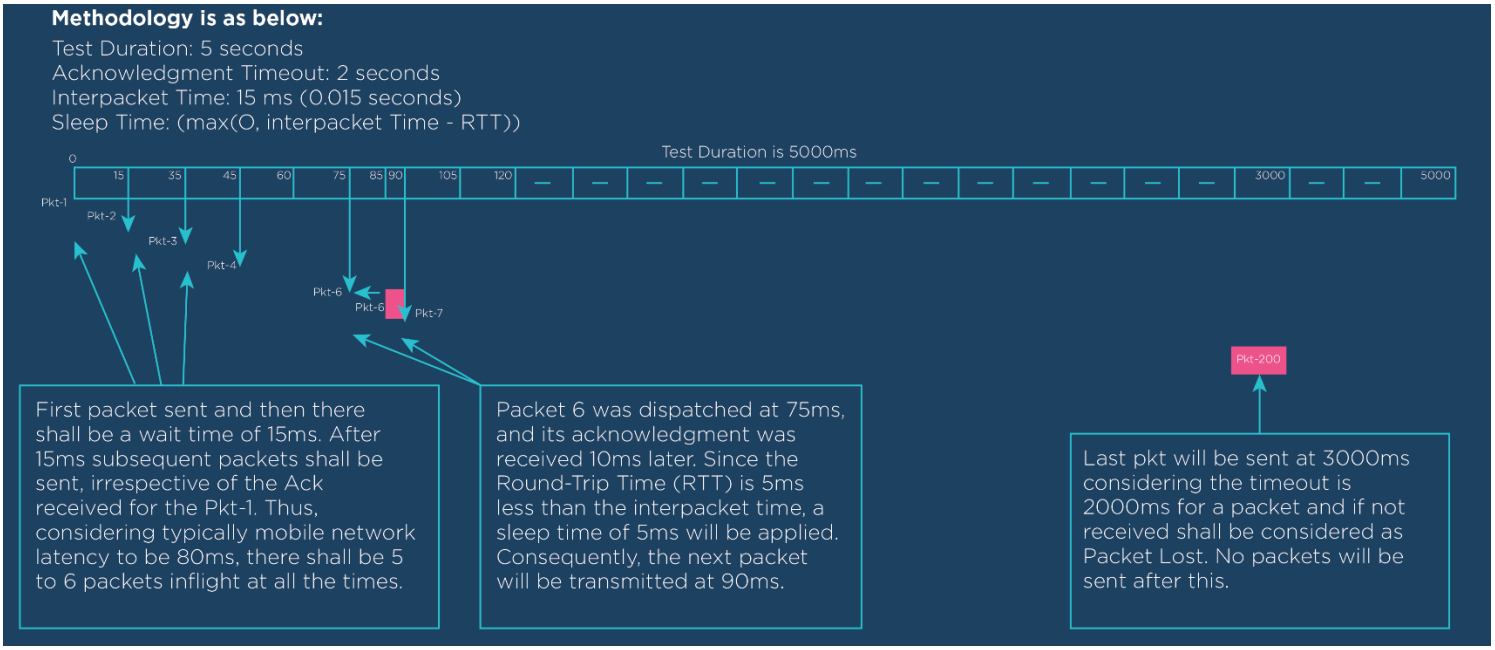
```
            // start new operation for the same threadID (which it finish uploading file)
            Op  =  operation.getOperation(operationID)
            startNewOperation(op.threadId)
        }


CalculateSpeed() {
    slotsPerThread = slotArray.filter(threadId)
    if elapsedTime <= testTimeout {
        trafficTime = transferMaxTime // 5 sec
    } else { // elapsedTime > 8
        trafficTime = elapsedTime - warmupMaxTime
    }
    for slot in slotsPerThread {
        // get bytes uploaded in the last 5 sec
        threadTime += taskTime
        if elapsedTime - slot.time <= trafficTime {
            threadTrafficBytes += slot.bytes
            if elapsedTime - prevSlot.time < trafficTime && prevSlot.time > WarmUpTime {
                threadTrafficBytes += slot.bytes
                timeToAdd = (WarmUpTime - slot.time)
            }
        }
        if threadTime <= trafficTime // (5 sec)
        {
        threadTrafficTime = threadTime
        }
        else {
            threadTrafficTime = (threadTime - WarmUpTime) + timeToAdd
        }
         threadWarmupBytes = allThreadBytes - threadTrafficBytes
        threadWarmupTime = allThreadTrafficTime - threadTrafficTime
        threadTrafficSpeed = threadTrafficBytes / threadTrafficTime
        threadWarmupSpeed = threadWarmupBytes / threadWarmupTime
        allThreadSpeed = allThreadSpeed + threadTrafficSpeed  // (displayed in gauge in Mbps)
        totalWarmupSpeed = totalWarmupSpeed + threadWarmupSpeed
        trafficBytes = trafficBytes + threadTrafficBytes
        warmupBytes = warmupBytes + threadWarmupBytes
    }
```

```
    trafficTime = trafficBytes / allThreadSpeed      // ( in Bytes / sec)
    warmupTime = warmupBytes / totalWarmupSpeed
    // trafficBytes => bytesTransferred
    // warmupBytes => warmuptransferred
    // trafficTime => traffic duration
    // warmupTime => warmup duration
}
```

D. Latency, Packet Loss and Jitter

The UDP Latency and Packet Loss tests measure the time needed for UDP packets to travel from the device to a designated test site and back within a set 5-second interval. The packets are transmitted only during the first 3 seconds, with the remaining 2 seconds dedicated to the acknowledgment of all the sent packets. Each packet size is 160 bytes. If an acknowledgment is not received within 2 seconds of sending a packet, it is considered lost. The test is programmed to transmit a maximum of 200 packets (datagrams) and maintains records on the number of packets sent, the average round-trip time for responses, and the total count of lost packets.

Figure 5: Example showing Latency Test Methodology



Latency test:

Get latency configuration from portal (interPacketTime + timeout + number of packets to send ...) and from optimal server: (ip + port)
Create packet with 160 bytes
if elapsedTime <= maxSendtime

```
{
      for i in 1...numberOfPacketSent {
         sendPacket()
         packetsSentArray.add(packet.identifier)
         sleep(15 ms)
      }
 }
 ReceivePacketsCallback(packet)
 {
            If packetsSentArray.contain(packet.identifier) {
               Rtt = currentTime - packet.sentTime
               If Rtt >= 2000 ms {
                  packetsLossArray.add(packet.identifier)
               }
               rttarray.add(rtt)
 }
```

5 -calculate avgArray base on rttArray and send the new value each 0.1 sec to the gauge in test screen
When we reach timeout, or we have 200 packets sent and 200 packets received we stop the test and we calculate

Jitter

Percentage of Number of packets loss = 1 – (# of packets lost/total # of Packets sent)

Bytes transferred = number of packet sent * 160 bytes

Traffic duration = elapsedTime

BytesSec = Bytes transferred / elapsedTime in sec

Jitter is determined by computing the average variation in Round-Trip Time (RTT) across successive packets. The image below provides how packet delay variation is calculated.

Table 1: Example showing Packet Delay Variation

Step 1: To get the variation subtract RTT value from previous RTT value

RTT Cycle	RTT	Variation= (RTT(n)-RTT(n+1))
CurrentRtt1	75.2641	
CurrentRtt2	73.95968	1.404416
CurrentRtt3	127.51	-53.550336
CurrentRtt4	110.4279	17.082112
CurrentRtt5	91.69306	18.734848
CurrentRtt6	72.64998	19.043072
CurrentRtt7	73.58797	-0.937984
CurrentRtt8	75.97773	-2.38976
CurrentRtt9	73.86112	2.116608
I		I
I		I
I		I
CurrentRtt197	115.1526	15.884032
CurrentRtt198	98.56205	16.590592
CurrentRtt199	79.75194	18.810112
CurrentRtt200	142.7799	-63.027968

Step 2: Change all negative values to positive

RTT Cycle	RTT	Variation= (RTT(n)-RTT(n+1))
CurrentRtt1	75.2641	
CurrentRtt2	73.95968	1.404416
CurrentRtt3	127.51	-53.550336
CurrentRtt4	110.4279	17.082112
CurrentRtt5	91.69306	18.734848
CurrentRtt6	72.64998	19.043072
CurrentRtt7	73.58797	-0.937984
CurrentRtt8	75.97773	-2.38976
CurrentRtt9	73.86112	2.116608
I		I
I		I
I		I
CurrentRtt197	115.1526	15.884032
CurrentRtt198	98.56205	16.590592
CurrentRtt199	79.75194	18.810112
CurrentRtt200	142.7799	63.027968

Step 3: Take average of variations to calculate jitter

Variation= (RTT(n)-RTT(n+1))
1.404416
53.550336
17.082112
18.734848
19.043072
0.937984
2.38976
2.116608
I
I
I
15.884032
16.590592
18.810112
63.027968

Jitter (Average of variations) = 14.03 ms

E. Background Testing

While the Android and iOS versions of the FCC Mobile Speed Test App allow users to initiate single and repeated tests in the foreground, the Android version of the App offers an additional feature, which automatically performs tests in the background at preset intervals. Users can turn this feature on or off from the App settings. iOS devices currently lack the capability for automated background testing. The Android automatic background testing approach is designed to minimize external factors that might impact the accuracy of mobile broadband performance measurements, thus providing valuable, high-quality data.

Even though the tests are scheduled to run at set intervals, the exact timing of the background test may not be precise. The framework may decide to delay tests, especially for higher frequencies or when the device is in power save mode, which is a low-power state during extended periods of inactivity.

The allowed frequencies are:

- Daily
- Weekly
- Monthly

F. Test Server and Data Collection

The App collects test results from active download, upload, latency, jitter, and packet loss tests. Additionally, the App collects date, time, location, connection, and device version data. All data are recorded on the device before it is reported to backend servers. Once the data is received by backend servers, it is entered into a database. Users can export data from the App in CSV format.[3]

All measurements are carried out using one of Mozark's test servers. Currently, these are:

- s1-fcc-test-lowa.mozark.ai
- s2-fcc-test-ohio.mozark.ai
- s1-oregon-us-west1-b-new.mozark.ai
- s1-mozark-nevada-fcc-new.mozark.ai
- s1-southcalifornia-us-east1-b.mozark.ai
- s1-oregon-us-west1-b-new.mozark.ai
- s1-fcc-georgia.mozark.ai
- s1-fcc-utah.mozark.ai
- s1-fcc-north-carolina.mozark.ai
- s1-fcc-nevada.mozark.ai
- s2-fcc-massachusetts.mozark.ai
- s2-fcc-illinois.mozark.ai
- s2-fcc-texas.mozark.ai
- s2-fcc-arizona.mozark.ai
- s2-fcc-washington.mozark.ai
- s2-fcc-colorado.mozark.ai
- s2-fcc-florida.mozark.ai
- s2-fcc-minnesota.mozark.ai
- s2-fcc-newyork.mozark.ai
- s2-fcc-pennsylvania.mozark.ai
- s1-fcc-california.mozark.ai
- s1-fcc-virginia.mozark.ai
- s1-fcc-texas.mozark.ai
- s1-fcc-ohio.mozark.ai

Test server selection:

This server selection is determined at the start of each test cycle. The optimal test server is determined for Android and iOS by measuring the latency using the ICMP ping. Multiple healthy servers, i.e., servers where utilization is under configured thresholds, are compared and the server with the lowest Round Trip Time [RTT] is used for measurements.

[3] The tests results are transferred depending on the available connectivity at the conclusion of the test and can be stored and forwarded when connectivity is not immediately available.

1. Android Test Flow and Environmental Data

After a user starts a test using an Android device, the following sequence of actions takes place:

- The App sends a current Geo Location to backend (Longitude and Latitude).
- Backend will share the best 10 servers after checking server health (CPU and Bandwidth Utilization) and determining the nearest location to user.
- App will perform the RTT test on each server as part of server selection process and choose the best server i.e., having the lowest RTT.
- The server selection process is performed against each of the targets in the list.
- The Radio Characteristics of network (e.g., signal strength, RSRP, RSSI, RSRQ, SINR, and CQI) are measured and reported at the beginning and end of every Latency, Download and Upload test.
- The UDP latency/Packet loss/jitter test is performed against the chosen test server.
- The Download test is performed against the chosen test server.
- The Upload test is performed against the chosen test server.
- The Radio Characteristics of network is measured and reported at the end of every Latency, Download and Upload test.
- The results are submitted to the backend Data Ingestion API and a resulting Test ID is retrieved.
- The results for the entire test sequence are paired with their respective Test ID and can be downloaded from the App.

At the end of a test sequence, the following information is displayed to the user:

- Download speed (Mbps)
- Upload speed (Mbps)
- Latency (ms)
- Packet loss (%)
- Jitter (ms)
- Test ID (Provided there was successful submission of test results, and this Test ID is generated)
- Connection Type (Indicates either the name of the wireless network that was tested if tests were purely conducted over cellular or indicates a Wi-Fi connection if tests were conducted over Wi-Fi at any point in the test sequence)
- Test Server (The location of the test server)

- Date (The date and time of measurement)

If any test fails to obtain a measurement for any reason (such as when there is no network connectivity at all), then “FAILED” is shown for the respective test result. If any other data was unavailable, due to a failure, then a "-" is shown in its place.

Table 2: Android Data Dictionary

The set of tables below define a collective list of all the data fields that are both collected during the tests and the fields that are generated server-side. A small subset of fields that only function as part of the submission from the Android App to the backend ingest are also listed for completeness.

The table below lists all possible fields that are reported by the Android App to the Mozark backend ingest.

Field Name	Description
submission_ category	The category of the data submission. This represents an enumerated value and must be one of the following: Consumer Crowdsource | Consumer Challenge.
name	The full name of the user.
email	The email address of the user. Value must match valid email address format e.g., user@domain.tld.
phone_number	The phone number of the device used for testing by the user. The phone number should be fixed to 10 digits. This value may be null if the submission_category is Consumer Crowdsource.
test_id	The unique Id for each test cycle, generated by backend.
device_timestamp	Timestamp of the time at which the test submission data were transmitted to the App's servers, measured by the device.
	- Value must match valid ISO-8601 format including seconds, e.g., YYYY-MM-DD[T]hh:mm:ss±hh:mm
server_timestamp	Timestamp of the time at which the test submission data were transmitted to the App's servers, measured by the server.
	- Value must match valid ISO-8601 format, e.g., YYYY-MM-DD[T]hh:mm:ss±hh:mm
server_source_ ip_address	Source IP address of the device submitting test submission data, measured by the server.
	- Value must be in valid IPv4 or IPv6 format if not null.
server_source_port	Source TCP port of the device submitting test submission data, measured by the server.
	-Value must correspond to transmission recorded in the server_timestamp and server_source_ip_address values as measured by the server, if not null.
device_type	Type of device.
	-Value must be either Android or iOS.
manufacturer	Name of the device manufacturer. e.g., Google.
model	Name of the device model. e.g., Pixel 6
operating_system	The OS version of Operating system e.g., 33
device_tac	The 8-digit Type Allocation Code (TAC) of the device during the Test e.g., 35142059 . The value may be null if the App does not have requisite permissions, or the device does not return a valid device TAC.
device_id	A unique device identifier generated by the App on the device, on an individual App-installation basis e.g., a255e318-df8d-46d1-a23b-9589e4d2e53e.

Field Name	Description
app_name	The name of the FCC Mobile Speed Test App. In the case of the Mozark App, this is currently “FCC Speed Test App - Android”.
app_version	The version of the FCC Mobile Speed Test App e.g., 1.0.
provider_name	The name of the mobile service provider e.g., T-Mobile.
sim_mobile_ country_code	The mobile country code of mobile service provider reported for the active SIM card in the device e.g., 310.
sim_mobile_ network_code	The mobile network code of mobile service provider reported for the active SIM card in the device e.g., 260.
beginning_cellular_net_mobile_country_code	The mobile country code of mobile service provider reported from the connected network at beginning of test e.g., 310.
end_cellular_net_mobile_country_code	The mobile country code of mobile service provider reported from the connected network at end of test e.g., 310.
beginning_cellular_net_mobile_network_code	The mobile network code of mobile service provider reported from the connected network at beginning of test e.g., 260.
end_cellular_net_mobile_network_code	The mobile network code of mobile service provider reported from the connected network at end of test e.g., 260.
in_vehicle_flag	Indicates whether the test was conducted while in-vehicle or outdoors.
scheduled_test_flag	Indicates whether the test was automated/scheduled or whether it was user- initiated manually. True if it is scheduled otherwise false.
external_antenna_flag	Boolean flag indicating whether an in-vehicle test was conducted using an antenna external to the vehicle. - The value may be false when in_vehicle_flag is false.
tests	The name of test e.g., Download/Upload/Latency.
download_time/upload_time/latency_time	Timestamp of the time at which the connection for the test metric was initialized (i.e., prior to any warmup period during which the connection stabilized). This is measured by the device and adheres to the ISO-8601 format e.g., YYYY-MM-DD[T]hh:mm:ss
warmup_ duration	Duration in microseconds that connection took to stabilize (e.g., TCP slow start) before the test metric commenced.
warmup_bytes_ transferred	Measured total amount of data in bytes that were transferred during the period the connection took to stabilize (e.g., TCP slow start) before the test metric commenced.
duration	Duration that the test metric took to complete in microseconds, excluding warmup time in case of download and upload.
bytes_transferred	Measured total amount of data in bytes that the test metric transferred, excluding warmup bytes transferred. Applies to only the Download and Upload tests.
bytes_sec	Measure number of bytes per second that the test metric transferred excluding warmup period. Applies to only the Download and Upload tests.
targets	List of hostname or IP address of target server(s) used for the test metric.
success_flag	Indicates status of test success (True) or failure (false). Latency: if total packet received array count = 0 status is False (Failure) else True (Success). Download: If totalBytesDownload= 0 after time out duration configured is False (Failure) else True (Success).

Field Name	Description
	Upload: If totalBytesUPload= 0 after time out duration configured is False (Failure) else True (Success).
beginning_carrier_aggregation_flag	Indicates whether the network used carrier aggregation during the test metric at beginning of test.
	- This value may be null for 3G tests. This value may also be null if the device does not return a valid value or a value of Unknown.
end_carrier_aggregation_flag	Indicates whether the network used carrier aggregation during the test metric at end of test.
	- This value may be null for 3G tests. This value may also be null if the device does not return a valid value or a value of Unknown.
network_ connected_flag	Indicates whether the network is connected.
network_ available_flag	Indicates whether the network is available.
network_ roaming_flag	Indicates whether the network is roaming.
round_trip_time	Round-trip latency in microseconds.
jitter	Round-trip jitter in microseconds.
packets_sent	Number of packets sent during the test.
packets_received	Number of packets received during the test.
beginning_location_time	Timestamp of the time at which the location was recorded at beginning of Test.
end_location_time	Timestamp of the time at which the location was recorded at end of Test.
beginning_latitude	Unprojected (WGS-84) geographic coordinate latitude in decimal degrees of the reported location at the start of test.
	Value must have minimum precision of 6 decimal digits.
end_latitude	Unprojected (WGS-84) geographic coordinate latitude in decimal degrees of the reported location at the end of test.
	Value must have minimum precision of 6 decimal digits.
beginning_longitude	Unprojected (WGS-84) geographic coordinate longitude in decimal degrees of the reported location at the start of test.
	Value must have minimum precision of 6 decimal digits.
end_longitude	Unprojected (WGS-84) geographic coordinate longitude in decimal degrees of the reported location at the end of test.
	Value must have minimum precision of 6 decimal digits.
beginning_horizontal_accuracy	Horizontal accuracy of the location, radial, in meters measured from the device at the end of test
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_horizontal_accuracy	Horizontal accuracy of the location, radial, in meters measured from the device at the beginning of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
avg_speed	Avg Speed in meters per second measured from the device.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
speed_accuracy	Speed accuracy in meters per second measured from the device.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_physical_cell_id	Measured Physical Cell Identity (PCI) of the cell at beginning of test
	- This value is null for 3G cells.

Field Name	Description
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_physical_cell_id	Measured Physical Cell Identity (PCI) of the cell at end of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_cell_connection	The connection status of the cell at beginning of test.
	The value is either 0, 1 or 2
	0 – Not Serving, 1 -Primary Serving, 2 – Secondary Serving.
	-This value may be null if the device does not return a valid value or returns a value of Unknown.
end_cell_connection	The connection status of the cell at end of test.
	The value is either 0, 1 or 2.
	0 – Not Serving, 1 -Primary Serving, 2 – Secondary Serving.
	-This value may be null if the device does not return a valid value or returns a value of Unknown.
beginning_network_generation	The network generation of the cell at beginning of test. The value is always one of the following; 2G |3G | 4G | 5G | unknown.
end_network_generation	The network generation of the cell at end of test. The value is always one of the following: 2G |3G | 4G | 5G | unknown.
beginning_network_subtype	The network subtype of the cell at beginning of test. The value is always one of the following: 1X | EVDO | WCDMA | GSM | HSPA | HSPA+ | LTE | NR_SA | NR_NSA.
end_network_subtype	The network subtype of the cell at end of test. The value is always one of the following: 1X | EVDO | WCDMA | GSM | HSPA | HSPA+ | LTE | NR_SA | NR_NSA.
beginning_rssi	The measured Received Signal Strength Indication (RSSI) of the cell at beginning of test.
	- This value may be null for 5G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_rssi	The measured Received Signal Strength Indication (RSSI) of the cell at end of test.
	- This value may be null for 5G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_rsrp	The measured Reference Signal Received Power (RSRP) of the cell at beginning of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_rsrp	The measured Reference Signal Received Power (RSRP) of the cell at end of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_rsrq	Measured Reference Signal Received Quality (RSRQ) in dB of the cell at beginning of test.
	- This value is null for 3G cells.

Field Name	Description
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_rsrq	Measured Reference Signal Received Quality (RSRQ) in dB of the cell at end of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_sinr	Measured Signal to Interference and Noise Ratio (SINR) in dB of the cell at beginning of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_sinr	Measured Signal to Interference and Noise Ratio (SINR) in dB of the cell at end of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_csi_rsrp	Measured 5G Channel State Information (CSI) RSRP in dBm of the cell at beginning of test.
	- This value is null for 3G and 4G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_csi_rsrp	Measured 5G Channel State Information (CSI) RSRP in dBm of the cell at end of test.
	- This value is null for 3G and 4G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_csi_rsrq	Measured 5G Channel State Information (CSI) RSRQ in dB of the cell at beginning of test.
	- This value is null for 3G and 4G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_csi_rsrq	Measured 5G Channel State Information (CSI) RSRQ in dB of the cell at end of test.
	- This value is null for 3G and 4G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_csi_sinr	Measured 5G Channel State Information (CSI) SINR in dB of the cell at beginning of test.
	- This value is null for 3G and 4G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_csi_sinr	Measured 5G Channel State Information (CSI) SINR in dB of the cell at end of test.
	- This value is null for 3G and 4G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_cqi	Measured Channel Quality Indicator (CQI) of the cell at beginning of test.
	- This value is null for 3G cells.

Field Name	Description
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_cqi	Measured Channel Quality Indicator (CQI) of the cell at end of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_spectrum_band	Spectrum band used by the cell at beginning of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_spectrum_band	Spectrum band used by the cell at end of test.
	- This value is null for 3G cells.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_spectrum_bandwidth	Total amount of spectral bandwidth used by the cell in MHz at beginning of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_spectrum_bandwidth	Total amount of spectral bandwidth used by the cell in MHz at end of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_arfcn	Measured physical RF channel number of the cell at beginning of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_arfcn	Measured physical RF channel number of the cell at end of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_location_provider	Name of location provider at the beginning of the test e.g., GPS.
end_location_provider	Name of location provider at the end of the test e.g., GPS.
beginning_cellular_cell_network_time	Timestamp indicating time when the test metric was initiated.
end_cellular_cell_network_time	Timestamp indicating time when the test metric was ended.
beginning_cellular_cid	Measured mobile broadcast cell identifier at beginning of test
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_cellular_cid	Measured mobile broadcast cell identifier at end of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
packet_loss	Percentage of packet lost during Latency test.
packet_size	Size of Packet sent during Latency test.
cycle_Date	Timestamp indicating time when the first test within the test cycle was initiated.
connection_type	Type of network connection e.g., Cellular or Wi-Fi.
location_type	Type of Location selected by user example Indoor, Ourdoor , In-Vehicle.
beginning_signal_strength	Measured signal strength in dBm of the cell at beginning of test.

Field Name	Description
	- Note: this value represents the Received Signal Strength Indication (RSSI) for 3G tests or the Reference Signal Received Power (RSRP) for 4G LTE or 5G-NR tests.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
	- Value is not available on iOS and may be null for these device types.
end_signal_strength	Measured signal strength in dBm of the cell at end of test.
	- Note: this value represents the Received Signal Strength Indication (RSSI) for 3G tests or the Reference Signal Received Power (RSRP) for 4G LTE or 5G-NR tests.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.

The following fields are generated server-side and are also present in the data transferred from the Mozark backend to the BDC backend. They are also present in the local device export data.

Field Name	Description
test_id	The unique identifier used by the App or entity to differentiate tests.
server_timestamp	The timestamp measured by the server of the moment at which the test submission data was received by the App's servers.
server_source_port	The source TCP port of the device submitting test submission data, measured by the server.
server_source_ip_address	The source IP address of the device submitting test submission data, measured by the server.

2. iOS Test Flow and Environmental Data

When a user starts a speed test using an iOS device, the following sequence of actions takes place:

- The App sends a Geo Location of the test to backend (Longitude and Latitude).
- Backend will share the maximum best 10 servers after checking server health (CPU and Bandwidth Utilization) and determining the nearest location to user.
- App will perform the RTT test on each of the above provided server list as part of server selection process and choose the server with lowest RTT.
- The UDP latency/Packet loss/jitter test is performed against the chosen test server.
- The Download test is performed against the chosen test server.
- The Upload test is performed against the chosen test server.
- The results are submitted to the backend Data Ingestion API and a resulting Test ID is retrieved.
- The results for the entire test sequence are paired with their respective Test ID and can be downloaded from the App.

At the end of a test sequence, the following information is displayed to the user:

- Download speed (Mbps)
- Upload speed (Mbps)
- Latency (ms)
- Packet loss (%)
- Jitter (ms)
- Test ID (provided there was successful submission of test results and this Test ID is generated)
- Connection Type (indicates either the name of the wireless network that was tested if tests were purely conducted over cellular or indicates a Wi-Fi connection if tests were conducted over Wi-Fi at any point in the test sequence)
- Test Server (the location of the test server)
- Date (the date and time of measurement)

If any test fails to obtain a measurement for any reason (such as when there is no network connectivity at all), then “FAILED” is shown for the respective test result. If any other data was unavailable, due to a failure, then a “-“ is shown in its place.

Table 3: iOS Data Dictionary

The set of tables below define a collective list of all the data fields that are both collected during the tests and the fields that are generated server-side. A small subset of fields that function only as part of the submission from the iOS App to the backend ingest are also listed for completeness.

The table below lists all possible fields that are reported by the iOS App to the Mozark backend ingest.

Field Name	Description
submission_category	The category of the data submission. This represents an enumerated value and must be one of the following: Consumer Crowdsource | Consumer Challenge.
name	The full name of the user.
email	The email address of the user. Value must match valid email address format e.g., user@domain.tld
phone_number	The phone number of the user. The phone no should be fixed to 10 digits. This value may be null of the submission category ; Consumer Crowdsource.
test_id	The unique Id for each test cycle, generated by backend.
device_timestamp	Timestamp of the time at which the test submission data were transmitted to the App's servers, measured by the device.
	- Value must match valid ISO-8601 format including seconds e.g., YYYY-MM-DD[T]hh:mm:ss±hh:mm
server_timestamp	Timestamp of the time at which the test submission data were transmitted to the App's servers, measured by the server.
	- Value must match valid ISO-8601 format e.g., YYYY-MM-DD[T]hh:mm:ss±hh:mm
server_source_ ip_address	Source IP address of the device submitting test submission data, measured by the server.
	- Value must be in valid Ipv4 or Ipv6 format if not null.
server_source_port	Source TCP port of the device submitting test submission data, measured by the server.
	-Value must correspond to transmission recorded in the server_timestamp and server_source_ip_address values as measured by the server, if not null.
device_type	Type of device: Value must be either Android or iOS.
manufacturer	Name of the device manufacturer. E.g., Apple.
model	Name of the device model e.g., "iPhone14,4" meaning iPhone 13 Mini.
operating_system	The OS version of Operating system e.g., 16.1.1
device_id	A unique device identifier generated by the App on the device, on an individual App-installation basis e.g., a255e318-df8d-46d1-a23b-9589e4d2e53e.

Field Name	Description
app_name	The name of the FCC Mobile Speed Test App. In the case of the FCC Mobile Speed Test App, this is always “FCC Speed Test App - iOS”.
app_version	The version of the FCC Mobile Speed Test App e.g., 1.0.
provider_name	The name of the mobile service provider e.g., T-Mobile.
sim_mobile_ country_code	The mobile country code of mobile service provider reported for the active SIM card in the device e.g., 310.
sim_mobile_ network_code	The mobile network code of mobile service provider reported for the active SIM card in the device e.g., 260.
in_vehicle_flag	Indicates whether the test was conducted while in-vehicle or outdoors.
external_antenna_flag	Boolean flag indicating whether an in-vehicle test was conducted using an antenna external to the vehicle.
	- The value may be false when in_vehicle_flag is false.
tests	The name of test e.g., Download/Upload/Latency.
download_time/upload_time/latency_time	Timestamp of the time at which the connection for the test metric was initialized (i.e., prior to any warmup period during which the connection stabilized). This is measured by the device and adheres to the ISO-8601 format e.g., YYYY-MM-DD[T]hh:mm:ss
warmup_ duration	Duration in microseconds that connection took to stabilize (e.g., TCP slow start) before the test metric commenced.
warmup_bytes_ transferred	Measured total amount of data in bytes that were transferred during the period the connection took to stabilize (e.g., TCP slow start) before the test metric commenced.
duration	Duration that the test metric took to complete in microseconds, excluding warmup time in case of download and upload.
bytes_transferred	Measured total amount of data in bytes that the test metric transferred, excluding warmup bytes transferred. Applies to only the Download and Upload tests.
bytes_sec	Measure number of bytes per second that the test metric transferred excluding warmup period. Applies to only the Download and Upload tests.
targets	List of hostname or IP address of target server(s) used for the test metric.
success_flag	Indicates whether the test completed successfully.
round_trip_time	Round-trip latency in microseconds.
jitter	Round-trip jitter in microseconds.
packets_sent	Number of packets sent during the test.
packets_received	Number of packets received during the test.
beginning_location_time	Timestamp of the time at which the location was recorded at beginning of test.
end_location_time	Timestamp of the time at which the location was recorded at end of test.
beginning_latitude	Unprojected (WGS-84) geographic coordinate latitude in decimal degrees of the reported location at the start of test.
	Value must have minimum precision of 6 decimal digits.

Field Name	Description
end_latitude	Unprojected (WGS-84) geographic coordinate latitude in decimal degrees of the reported location at the end of test.
	Value must have minimum precision of 6 decimal digits.
beginning_longitude	Unprojected (WGS-84) geographic coordinate longitude in decimal degrees of the reported location at the start of test.
	Value must have minimum precision of 6 decimal digits.
end_longitude	Unprojected (WGS-84) geographic coordinate longitude in decimal degrees of the reported location at the end of test.
	Value must have minimum precision of 6 decimal digits.
beginning_horizontal_accuracy	Horizontal accuracy of the location, radial, in meters measured from the device at the beginning of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
end_horizontal_accuracy	Horizontal accuracy of the location, radial, in meters measured from the device at the end of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
avg_speed	Avg speed in meters per second measured from the device at beginning of test.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
speed_accuracy	Speed accuracy in meters per second measured from the device.
	- Value may be null if the device does not return a valid value or else returns a value of unknown.
beginning_network_generation	The network generation of the cell at beginning of test. The value is always one of the following: 2G |3G | 4G | 5G | unknown.
end_network_generation	The network generation of the cell at end of test. The value is always one of the following: 2G |3G | 4G | 5G | unknown.
beginning_network_subtype	The network subtype of the cell at beginning of test. The value is always one of the following: 1X | EVDO | WCDMA | GSM | HSPA | HSPA+ | LTE | NR_SA | NR_NSA.
end_network_subtype	The network subtype of the cell at end of test. The value is always one of the following: 1X | EVDO | WCDMA | GSM | HSPA | HSPA+ | LTE | NR_SA | NR_NSA.
beginning_location_provider	Name of location provider at the beginning of the test e.g., GPS.
end_location_provider	Name of location provider at the end of the test e.g., GPS.
beginning_cellular_cell_network_time	Timestamp indicating time when the test metric was initiated.
end_cellular_cell_network_time	Timestamp indicating time when the test metric was ended.
packet_loss	Percentage of packet lost during Latency test.
packet_size	Size of Packet sent during Latency test.
cycle_Date	Timestamp indicating time when the first test within the test cycle was initiated.
connection_type	Type of network connection e.g., Cellular or Wi-Fi.
location_type	Type of Location selected by user example Indoor, Ourdoor , In-Vehicle.

G. App Functionality

The FCC Mobile Speed Test App runs tests to measure download and upload speed, latency, jitter, and packet loss. If users have a limited data plan, they can set a data limit for the App and which day of the month the limit resets. The App conducts crowdsource and challenge tests to provide network information. It also offers the option for users to execute repeated tests for both crowdsource and challenge tests.

Crowdsource Test: This type of test collects data from users to assess the performance of mobile broadband and internet services. Tests measure download and upload speeds, latency, packet loss, and other relevant metrics related to their internet and mobile broadband connections. The collected data is crowdsourced, meaning it comes from a broad and diverse range of users across the United States. This approach provides a comprehensive view of network performance. The FCC uses the data to analyze the state of broadband and wireless services in different regions.

Challenge Test: A Challenge Test is used to evaluate the accuracy of the broadband coverage maps provided by internet service providers. It is designed to encourage consumers to dispute the broadband coverage information reported by ISPs. Tests measure download and upload speeds, latency, packet loss, and other relevant metrics related to their internet and mobile broadband connections. Multiple challenge tests within a geographic area that meet certain criteria are used to challenge a provider's mobile broadband coverage in that area. Speed tests that generate a challenge are shared with the provider and help improve and update the FCC's National Broadband Map.

Repeated Crowdsource or Challenge Test: This feature allows users to perform back-to-back Challenge or Crowdsource tests without having to start a new test each time. Users can specify the number of tests, the time between each test, and the duration of the repeated tests. The maximum duration for these repeated tests is 4 hours. Users can interrupt repeated testing by tapping the "Stop Test" button. A repeated test session can be stopped after the completion of a test cycle.

Background Test (Android Only): This feature allows users to run periodic background tests. Users have the ability to select the preferred schedule of background tests: daily, weekly, or monthly. Once users enable the background test feature, tests will autonomously run based on the schedule selected, eliminating the need for manual initiation by the user. Users may enable or disable the background test feature.