PUBLICLY SHARED

2021 Platform Channel Performance Tuneup

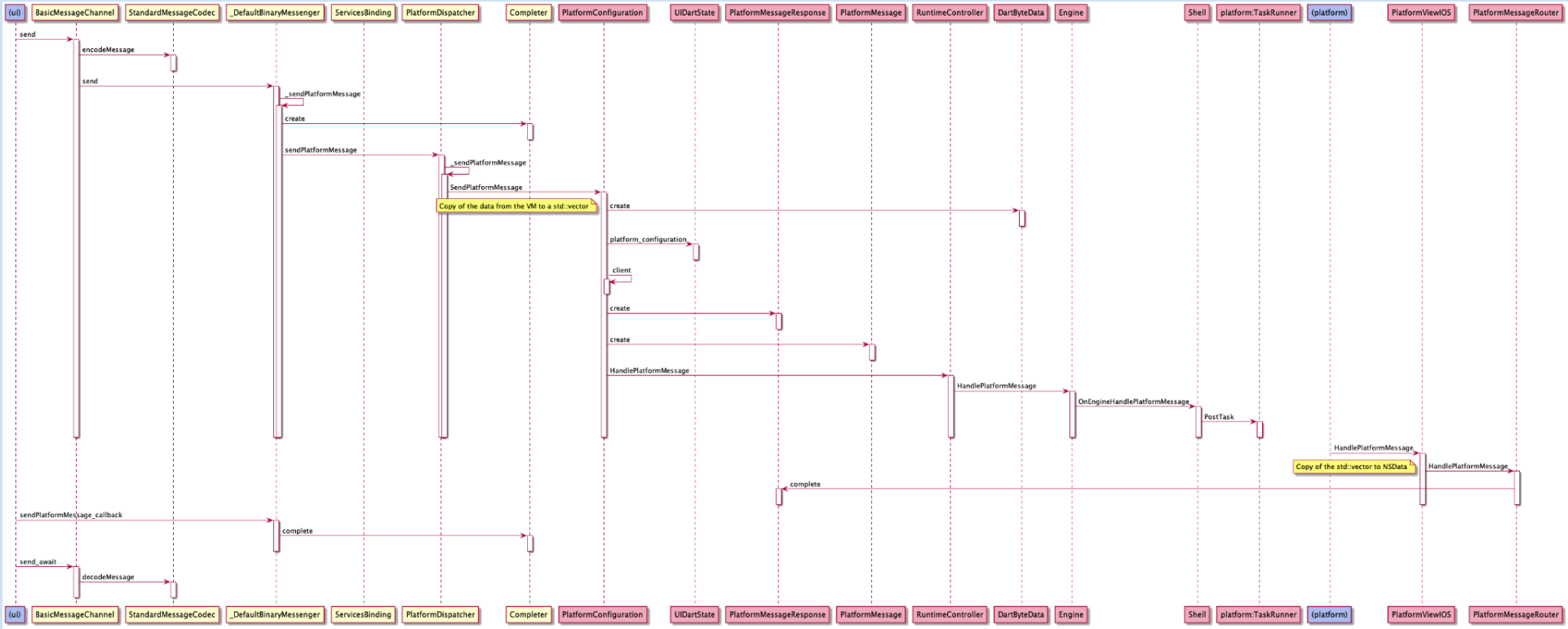
gaaclarke
authored: 4/22/21
edited: 5/21/2021

Description

Users are running into the current limitations of Flutter's platform channels when doing Add-to-app and Plugins. Platform Channels are overdue for a performance tuneup, this document outlines the fixes we can do to improve their performance.

Background

Here is a sequence diagram that shows sending a Platform Channel message from Flutter to iOS:



Here you can see that we are performing 2 different copies of the data. One that copies the data from the Dart VM to native memory, then another copy that copies from the native memory to the platform memory format (NSData in Objc or byte array for Java). The same process happens in reverse for returning responses, but is ellided in this diagram. Also, a similar flow is happening when sending messages from the host platform to Flutter.

Users are particularly concerned about the performance of large payloads (#79617) or building advanced features on top of Platform Channels (http://flutter.dev/go/data-sync). Furthermore, Platform Channels are used as part of typical Flutter operation inside the engine, even when not doing Add-to-app or plugins. So, any performance improvement for them will be a boon to all users.

Proposal

1. [Performance tests] Add performance benchmarks for platform channels. We are about to tweak the performance of the channels and make sure that we are making progress. I propose the following device lab performance tests for iOS and Android (I tried to get a representative sample without doing every possible test):

Direction	Channel	Codec	Payload	Response
Flutter->Host	BasicMessageChannel	StandardMessageCodec	Small, just null	Small, just null
Flutter->Host	BasicMessageChannel	StandardMessageCodec	Large, array of all supported data types ~2k	Large, array of all supported data types ~2k
Flutter->Host	BasicMessageChannel	BinaryCodec	Large, ~4k	Large, ~4k
Host->Flutter	BasicMessageChannel	StandardMessageCodec	Large, array of all supported data types ~2k	Large, array of all supported data types ~2k
Flutter->Host	MethodChannel	StandardMethodCodec	Small, null method	Small, null
Flutter->Host	MethodChannel	JSONMethodCodec	Small, null method	Small, null

2. [iOS] Eliminate copies between NSData and std::vector by switching the std::vector to a Mapping object that can transfer ownership of the raw buffer:
    a. Flutter -> Host Platform Messages:
        i. Eliminate the copy of the std::vector to the NSData (platform_message_router.mm#L25).
        ii. Eliminate the copy from the NSData to the std::vector for the response (platform_message_router.mm#L30:L30)

   b. Host -> Flutter Platform Messages:
      i. Eliminate the copy from std::vector to NSData (FlutterEngine.mm#L735)
      ii. Eliminate the copy from NSData to std::vector for the response (platform_message_response_darwin.mm#L20:L20)
3. [Android] Eliminate the copy from std::vector to a byte array by using direct buffers (platform_view_android_jni_impl.cc#L1111:L1111). We already wrap the byte array in a ByteBuffer so this shouldn't be a breaking change. The 4 same copies that happen on iOS may be able to be removed with the direct buffers on Android.
4. [Framework] Audit the performance of the codecs. I've already removed extra runtime checks in WriteBuffer in this PR: flutter/pull/80588.
5. TypeData - (I believe this can be used to remove the copy of data from the Dart VM to native memory. I don't know much about this.)
6. [Performance tests] (Optional) Add micro benchmarks for the codecs. These will be covered indirectly via the higher tests. It would be nice to have a bit finer grain view of the codec performance.

Results

5/21/2021: I was able to eliminate the copies on iOS that happen when going from C++ to Objective-C and back. This resulted in a 42% savings when using the BinaryCodec with a payload size of 1MB: skiaperf.

I eliminated the copy on Android but the savings were insignificant because the cost of scheduling on the UI thread from the Platform thread is so high (~13ms). See flutter/issues/81559.