Of Blobs and Message Ports

From the CL description for https://codereview.chromium.org/2422793002:

> Finally, Blob registration (RegisterBlob) and coining a Blob URL (RegisterPublicURL) are made into synchronous IPCs. This is to avoid a race condition between Blob(URL) registration and passing a Blob(URL) over a Mojo channel, which is now a different FIFO. The previous code relied on the Chrome IPC channel between the renderer and browser being a single FIFO over which Blob(URL)s are registered and MessagePort IPCs are transmitted. Using a Mojo message pipe has the consequence of creating a new FIFO, which is good for performance (the FIFO is also direct between end-points rather than routing through the browser IO thread), but it comes at the cost of introducing potential race conditions. **A number of alternative designs were considered that come with much greater complexity (see https://goo.gl/bfdE64).** Instead, this CL adds UMA metrics to measure the cost of these synchronous IPCs. The cost should be acceptably low as these IPCs terminate on the browser IO thread.

This document elaborates on some of those alternative designs. Three alternative designs were considered:

1) Encode UUIDs w/ the process ID of the originator. If the browser receives a UUID it does not recognize, send an IPC to the corresponding process that just sends a simple reply IPC to the browser's IO thread over the same FIFO used to register UUIDs. This has the effect of flushing the queue of incoming registrations. Downsides: complexity and the mitigation only kicks in when there is a race, which means it may be a fragile code path. This also adds another asynchronous stage to the browser process. We'd also want to refactor the way UUIDs are both created and registered to use common code. Lot's more code change.

2) Delay sending messages on message ports when there is an outstanding (unacknowledged) UUID registration. This would have to be a blocking wait similar to the proposed synchronous IPCs but the blocking is done at the last possible moment. This is a lot of complexity. Again as in #1, it has the downside of being a code path that only lights up in the race condition. Hard to test, etc.

3) Add another special MessagePipeHandle to each PostMessage. Send the other end of the MessagePipeHandle to the browser IO thread over the same FIFO used to register UUIDs. Have the browser IO thread simply write a byte to the received MessagePipeHandle. Now, have the recipient of the sent message, wait on their end of the MessagePipeHandle to ensure that the browser IO thread has observed any possible UUID registrations. This unfortunately adds a lot of complexity on the receiving side of the MessagePort postMessage call. The recipient would need to deal with the

MessagePort being moved while waiting for the browser IO thread to signal that a received message is safe to use. This adds another asynchronous stage and complexity. We would need to move the pending message with the MessagePort if the MessagePort is moved to another process. That is very complex. We could also make the recipient do a blocking wait on the MessagePipeHandle, but that adds a behavior / code path that only lights up in the race condition. Hard to test, etc. This solution is also pretty expensive in terms of performance overhead w/ the extra message pipe needed for signaling.

Long term, I think we will be able to replace Blob uuids with MessagePipeHandles, and then we can just pass the Blob's MessagePipeHandle over a MessagePort when we wish to transfer it.