BiDi in ChromeDriver design doc

This Document is Public

Short link: go/bidi-chromedriver
Authors: sadym@chromium.com, nechaev@chromium.com

Signed off by

Summary

Goal

How?

Plan for the first iteration (Q1)
- Assumptions
- No TLS support (Q1 or even whole 2022)
- Single connection per instance of ChromeDriver (2022)
- Single listener now and forever
- Single connection per session (2022)
- Sending the Mapper to Chrome
- Messaging between ChromeDriver and Chrome
- Acceptance Test (Q1)
- TODO
- Open Issues (need to be addressed before start)

Signed off by

Name	Write LGTM (or not) in this column
Mathias Bynens <mathias@chromium.org>	LGTM
Philip Jägenstedt <foolip@chromium.com>	Unclear whether changes to ChromeDriver threading model are needed to implement.

Summary

As a part of Chrome DevTools OKRs 2022 , ChromeDriver should support the WebDriver BiDi protocol:

“O1 Popularize cross-browser end-to-end testing” -> “KR4 Prototype Puppeteer and ChromeDriver to use BiDi [P2]”

Goal

Acceptance criteria:

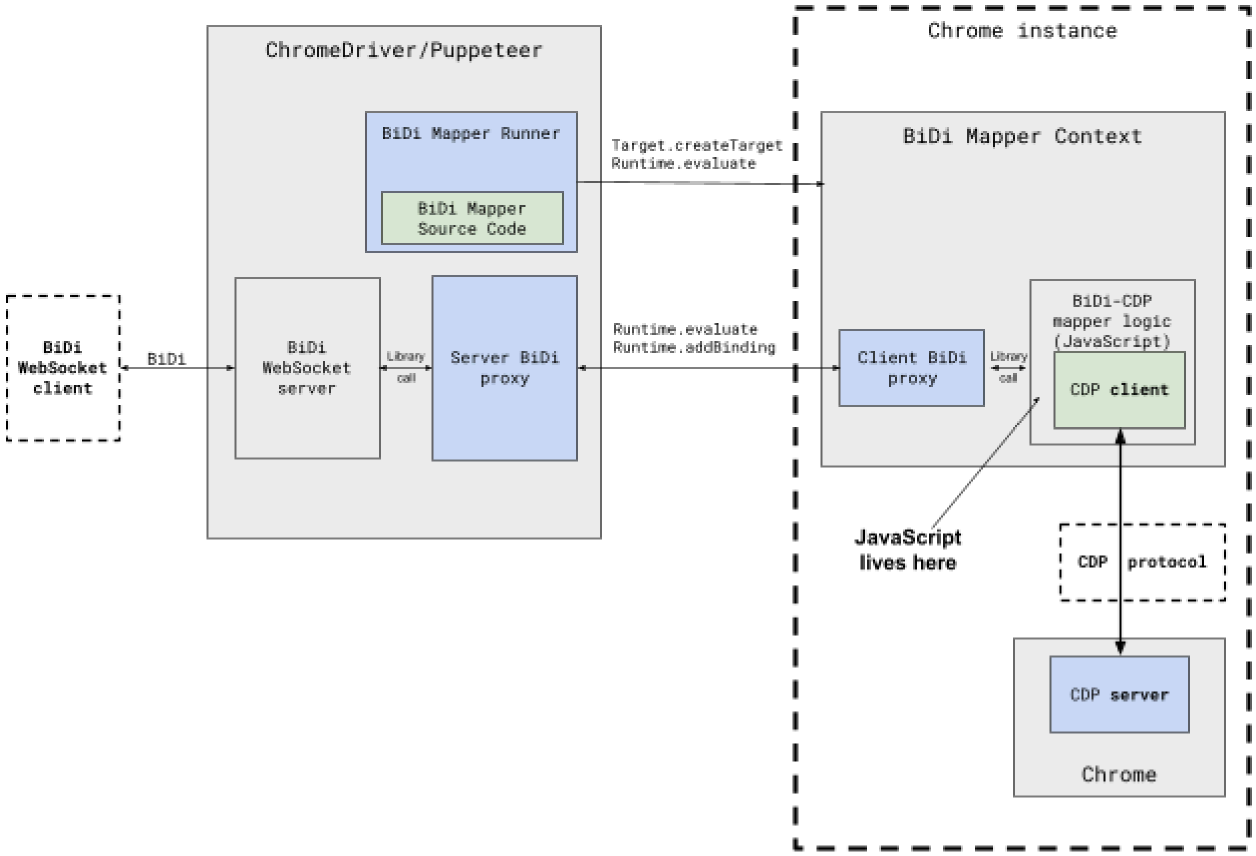
1. ChromeDriver accepts WebSocket connections.
2. ChromeDriver provides WebDriver BiDi protocol via established WebSocket connection.

How?

Alternatives considered in:

- WebDriver BiDi implementation design
- Async Command Processing for WebDriver in Chromium
- WebDriver BiDi C++ Backend Design.

As a part of WebDriver BiDi in Chrome Context implementation plan, the implementation should be like in the scheme. Specifically, ChromeDriver should run the CDP-BiDi mapper:



We propose that ChromeDriver does the following:

1. Provide Websocket server.

2. Get *BiDi Mapper Source Code* (go/bidi-mapper-source).
3. On each new WS (websocket) connection:
    a. Launch browser.
    b. (BiDi Mapper Runner) Run BiDi-CDP Mapper in the new browser context via CPD + provide callback bindings for the Mapper.
    c. (Server BiDi proxy) Forward all the WS messages to the Mapper via CDP command `Runtime.evaluate`.
    d. Forward all the Mapper messages to WS by listening to the binding events.

Plan for the first iteration (Q1)

Assumptions

- Amount of simultaneous connections to ChromeDriver is small. It does not exceed 10.
- Same session can be either classic, BiDi or both (as per the spec).
- TLS certification is not interesting to the user; only encryption is.
- Only a small share of users demands multiple connections per session.

No TLS support (Q1 or even whole 2022)

WebDriver-BiDi standard supports both secure and insecure connections.
Certification was probably not the cornerstone of TLS support.
End-to-end encryption however looks to be a more attractive benefit.
And here we have a problem that ChromeDriver talks to Chrome over an insecure channel.
In this situation claiming something about security can mislead the user.
Therefore it is proposed to support only insecure connections for Q1 or whole 2022.

Single connection per instance of ChromeDriver (2022)

Current ChromeDriver implementation abilities are limited by a single IO thread.
This thread listens for incoming connections, creates active sockets, reads user requests, forwards them to the session threads and sends back the results.
This works well with HTTP but with WebSocket this will limit ChromeDriver to handle a single WebSocket connection at a time.
Moreover while handling WebSocket traffic it will ignore (queue) the http traffic.
To solve this problem we need to implement another approach like either of:
- 1 listener thread + 1 thread per active socket.
- 1 thread for everything + asynchronous socket programming (polling)

These can be analyzed and one of them can be selected for the future implementation. In the future, a separate design doc will be dedicated to this investigation. However for now we will stick to the current implementation with its limitations because BiDi is currently in an active development phase and our main goal is to allow fast prototyping and testing of the protocol. One WebSocket connection per ChromeDriver instance seems to be a tolerable limitation for the current standard development phase.

Single listener now and forever

The BiDi standard allows having multiple listeners.

Does this multi-listener set up give us any benefits?
We do not expect high contention over the incoming connections therefore load balancing, that we might achieve exploiting multiple listeners on multiple threads, is not an attractive argument for this design.
On the other hand such architecture complicates the implementation.

Therefore we will have a single listener per ChromeDriver instance that will accept both HTTP and WebSocket connections.
The active sockets created by this passive listener for WebSocket traffic will be handled by dedicated threads (1 thread per websocket) or in the same thread if we use select or poll commands.

There is still an open question if this listener thread should read / write HTTP traffic.
The current approach (same thread for accept / read / write) seems to be ok as contention for connections is small.

Single connection per session (2022)

Multiple connections for a single session might be useful.
For instance, a client might execute some scenario via one connection and measure performance via another connection.
This advantage however seems to be rather exotic (not highly demanded by the users).

On the other hand this approach has the following disadvantages:

- Session lifetime management becomes more complicated and error prone (for some implementations).
- Multiple connections create additional races on the browser side. This will mostly concern the client code but can lead to increased bug reports if users suspect a bug on ChromeDriver side.
- Current implementation of Mapper closes Chrome when the CDP connection is closed. The time required to resolve this issue might be better invested into other topics at this time point.

The aforementioned problems will require significant maintenance efforts and can slow down the overall progress of BiDi implementation.

Therefore **we will postpone implementation of multiple connections per session** until the moment that we clearly see that this feature is really demanded by the users.

Sending the Mapper to Chrome

Steps to be executed by ChromeDriver:

1. Start an instance of Chrome.
2. Create a target (Target.create) and attach to it (Target.attach) obtaining the sessionId in return. This session id is used for communication with the mapper.
3. Inject and object providing CDP communication channel into the target's main frame (Target.exposeDevToolsProtocol).
4. Read the mapper code. Decision on how exactly will be described in CDP-BiDi Mapper source code lifecycle. Probably the code must be embedded into ChromeDriver code in the same way as we embed other JS code, i.e as a string literal.
5. Execute the mapper code via 'Runtime.evaluate' command

Messaging between ChromeDriver and Chrome

ChromeDriver forwards user requests to Chrome via method 'Runtime.evaluate' with argument 'expression' = 'onBidiMessage(' + bidiMessage + ')'.
Chrome sends messages to ChromeDriver as CDP events.

Acceptance Test (Q1)

We need to create one: something simple like create a websocket, connect to ChromeDriver, create a new session, navigate, execute some script, close the session.

There are 2 ways to implement tests, we can use both of them:

1. Add BiDi to the existing e2e tests (depends on the Python3 migration).
    - There are e2e tests in ChromeDriver BiDi NodeJS implementation, which can be used as a blueprint.
    - This approach allows to catch issues early during the development.
2. Run exising WPT tests.
    - This approach allows to keep inline with the WPT, which is a source-of-truth for the protocol implementation.

TODO

Elaborate on how ChromeDriver launches Mapper code in Chrome
Come up with some diagnostics for BiDi tab closure

Open Issues (need to be addressed before start)

1. Decision about CDP-BiDi Mapper source code lifecycle