Proxy-Wasm: a “standard” WebAssembly interface for the data plane

(April 2021 update)

Piotr Sikora

Proxy-Wasm recap

- Extend capabilities of the proxy and/or add business logic without modifying it.
- **Plugins portable between different proxies and infrastructure providers (“Write once, run anywhere”).**
- Ability to update plugins at runtime (e.g. “instant” security fixes).
- Ability to write plugins in higher-level and/or safer programming languages.
- Reliability (sandboxed plugins cannot crash host proxy).
- Security (sandboxed plugins cannot access outside world).
- Istio/Envoy-specific background at https://bit.ly/envoy-wasm-doc.

Proxy-Wasm features

- Core: logging, metrics.
- Network filters: accept/reject connection, get/set payload, access logs.
- HTTP filters: accept/reject request, get/set headers, get/set body, get/set trailers, access logs.
- HTTP and gRPC callouts.
- Shared key-value store, queue (non-durable).
- Background services and timers.
- Foreign functions and callbacks.

Proxy-Wasm status and adoption

Host implementations and libraries:

- Envoy and Istio Proxy (https://github.com/envoyproxy/envoy)
- MOSN (https://github.com/mosn/mosn)
    - Embeddable Go Host library (https://github.com/mosn/proxy-wasm-go-host)

SDKs:

- AssemblyScript (https://github.com/solo-io/proxy-runtime)
- C++ using Emscripten (https://github.com/proxy-wasm/proxy-wasm-cpp-sdk)
- Go using TinyGo (https://github.com/tetratelabs/proxy-wasm-go-sdk)
- Rust (https://github.com/proxy-wasm/proxy-wasm-rust-sdk)

ABI specification is lagging behind, so they're all using old and undocumented version.

Proxy-Wasm requirements

- Plugins executed in-process.
  - O(1µs) vs O(1ms).
- Minimal memory overhead:
  - Processing multiple streams at the same time (intermixed, not concurrently) inside a single persistent WasmVM instance.
  - Running multiple instances of the same plugin with different configurations.
- Handle as much as possible in heavily optimized host:
  - TLS processing done by the host, plugins see only decrypted data.
  - HTTP processing done by the host, plugins see only HTTP messages.

Proxy-Wasm design

- Event-driven streaming APIs using short-lived entrypoints:
  - `proxy_on_http_request_headers()`, etc.
- Minimize number of calls across the ABI boundary:
  - Size hints passed as entrypoint arguments.
  - Actions as return values.
- Transforming “pre-opened” client<>backend streams:
  - No `accept()`, no `connect()`.
  - Unmodified payload forwarded by default.
  - Zero-copy, and plugin bypass.

Proxy-Wasm design (example)

`proxy_on_http_request_body(context_id, body_size, end_of_stream) -> action`

Called for each chunk of HTTP request body for stream **`context_id`** received from downstream.
Request body (of size **`body_size`**) can be retrieved using **`proxy_get_buffer`**.
Return value (**`action`**) instructs the host environment what action to take:

- **`Forward`** means that this HTTP request body chunk should be forwarded upstream.
- **`EndStream`** means that this HTTP request body chunk should be forwarded upstream and that the underlying stream should be closed afterwards.
- **`Done`** means that this HTTP request body should be forwarded upstream and that this extension shouldn't be called again for this HTTP request.
- **`Pause`** means that further processing should be paused until **`proxy_http_resume_request`** is called.
- **`WaitForMoreData`** means that further processing should be paused and host should call this callback again when it has more data.
- **`WaitForEndOrFull`** means that further processing should be paused and host should call this callback again when complete HTTP request body is available or when its internal buffers are full.
- **`Close`** means that the stream should be closed immediately.

Changes (work in progress) since last update

- Memory allocations happen only in plugins (previously, allocations of a specific size were requested by host to avoid “get size” hostcalls).
- `context_id` (handle) passed as an argument in hostcalls (previously, host was kept track of the active context).
- Explicit `open/create` and `delete` hostcalls to support multiple instances of timers, queues, etc. (previously, there was only a single instance of each).
- Change to `iovec_array` for operations on byte buffers.
- Support for keys with multiple values in maps.

Proxy-Wasm / WASI components

- `wasi-context`,
- `wasi-common`
- `wasi-bytestream`,
- `wasi-http`,
- `wasi-grpc`,
- `wasi-log`,
- `wasi-kvstore`,
- `wasi-queue`,
- `wasi-timer`,
- `wasi-metrics`,
- `wasi-ffi`.

wasi-context (or wasi-plugin?)

```
on_plugin_start(plugin_id, configuration_size) -> is_success
on_plugin_shutdown(plugin_id) -> is_completed
plugin_shutdown(plugin_id)

on_context_create(context_id, plugin_id) -> new_context_id
on_context_finalize(context_id)
```

BIKESHEDDING:

1. `on_context_create()` vs `on_context_create(plugin_type)` vs `on_http_context_create()`
2. `on_context_create(context_id, …)` vs `on_context_create(request_id, response_id, …)`

QUESTION:

Do we need the ability to change `context_id` to `new_context_id` or not?

wasi-common

```
get_buffer(context_id, buffer_type, offset, max_size, return_content) -> status
set_buffer(context_id, buffer_type, offset, size, content) -> status

BIKESHEDDING: set_buffer(offset, size) vs {prepend,replace,append}_buffer

get_map_values(context_id, map_type, key_names, return_map) -> status
set_map_values(context_id, map_type, keys_to_remove, map_content) -> status

BIKESHEDDING: set_map_values(keys_to_remove) vs {set,add,remove}_map_values(key)

get_property(context_id, property_type, return_content) -> status
set_property(context_id, property_type, context) -> status

BIKESHEDDING: {get,set}_property(...) vs {get,set}_value(...)
```

wasi-bytestream (or wasi-bytestream-proxy?)

```
on_new_connection(context_id) -> action
on_downstream_data(context_id, data_size, end_of_stream) -> action
on_downstream_close(context_id, close_source)
on_upstream_data(context_id, data_size, end_of_stream) -> action
on_upstream_close(context_id, close_source)

{get,set}_buffer(context_id, buffer_type, …) from wasi-common

resume_downstream(context_id) -> status
resume_upstream(context_id) -> status
close_downstream(context_id) -> status
close_upstream(context_id) -> status
```

BIKESHEDDING:

1. `on_{downstream,upstream}_data(context_id)` vs `on_bytestream_data(downstream_id)`
2. `close_{downstream,upstream}(context_id)` vs `close_bytestream(downstream_id)`

wasi-http (or wasi-http-proxy?)

```
on_http_request_headers(context_id, num_headers, end_of_stream) -> action
on_http_request_body(context_id, body_size, end_of_stream) -> action
on_http_request_trailers(context_id, num_headers) -> action
on_http_request_metadata(context_id, num_elements) -> action

on_http_response_headers(context_id, num_headers, end_of_stream) -> action
on_http_response_body(context_id, body_size, end_of_stream) -> action
on_http_response_trailers(context_id, num_headers) -> action
on_http_response_metadata(context_id, num_elements) -> action
```

NOTE: `metadata` refers to HTTP/2 METADATA frames.

BIKESHEDDING:

`on_http_{request,response}_headers(context_id, …) vs on_http_headers(request_id, …)`

wasi-http (or wasi-http-proxy?)

```
{get,set}_map_values(context_id, map_type, ...) from wasi-common
{get,set}_buffer(context_id, buffer_type, ...) from wasi-common
```

NOTE: HTTP method, path and authority are available as part of the header map using HTTP/2 pseudo-header names (e.g. `:authority`).

```
send_http_response(context_id, status_code, headers, body, grpc_status) -> status

resume_http_request(context_id) -> status
resume_http_response(context_id) -> status
close_http_request(context_id) -> status
close_http_response(context_id) -> status
```

BIKESHEDDING:

```
resume_http_{request,response}(context_id) vs resume_http(request_id)
```

wasi-http (or wasi-http-client?)

NOTE: Side calls are sent directly to targets, not part of the main request/response stream.

```
dispatch_http_call(upstream_name, headers, body, trailers, timeout, return_id) -> status
on_http_call_response(callout_id, num_headers, body_size, num_trailers) -> action
close_http_call(callout_id) -> status

on_http_call_response_headers(callout_id, num_headers) -> action
on_http_call_response_body(callout_id, body_size, end_of_stream) -> action
on_http_call_response_trailers(callout_id, num_trailers) -> action
```

BIKESHEDDING:

```
on_http_call_response_{headers,body,trailers} vs on_http_response_{headers,body,trailers}
```

wasi-grpc (or wasi-grpc-client?)

NOTE: Side calls are sent directly to targets, not part of the main request/response stream.

```
dispatch_grpc_call(upstream_name, ...) -> status
on_grpc_call_response(callout_id, ...) -> action
close_grpc_call(callout_id) -> status

open_grpc_stream(upstream_name, ...) -> status
send_grpc_stream_message(callout_id, ...) -> status
close_grpc_stream(callout_id) -> status

on_grpc_stream_response_header_metadata(callout_id, ...) -> action
on_grpc_stream_response_message(callout_id, ...) -> action
on_grpc_stream_response_trailer_metadata(callout_id, ...) -> action
on_grpc_stream_close(callout_id, ...) -> action
```

wasi-log

```
log(log_level, message) -> status
get_log_level(return_log_level) -> status

on_log_level_update(log_level)
```

wasi-kvstore

```
open_shared_kvstore(name, create_if_not_exist, return_id) -> status
delete_shared_kvstore(kvstore_id) -> status

get_shared_kvstore_key_values(kvstore_id, key, return_values, return_cas) -> status
set_shared_kvstore_key_values(kvstore_id, key, values, cas) -> status
add_shared_kvstore_key_values(kvstore_id, key, value, cas) -> status
remove_shared_kvstore_key(kvstore_id, key, cas) -> status
```

QUESTIONS:

1. Do we want to support multiple values in the same key?
2. Do we want CAS (compare-and-swap)?

wasi-queue

```
open_shared_queue(queue_name, create_if_not_exist, return_id) -> status
delete_shared_queue(queue_id) -> status

dequeue_shared_queue_item(queue_id, return_payload) -> status
enqueue_shared_queue_item(queue_id, payload) -> status

on_shared_queue_ready(queue_id)
```

wasi-timer

```
create_timer(period, oneoff, return_id) -> status
delete_timer(timer_id) -> status

on_timer_ready(timer_id)
```

wasi-metrics

```
create_metric(metric_type, metric_name, return_id) -> status
get_metric_value(metric_id, return_value) -> status
set_metric_value(metric_id, value) -> status
increment_metric_value(metric_id) -> status
delete_metric(metric_id) -> status
```

TODO: Missing scopes (groups of metrics).

wasi-ffi

```
call_custom_function(custom_function_id, parameters, return_id) -> status
get_custom_function_results_sizes(return_id, return_count, return_size) -> status
get_custom_function_results(return_id, return_values) -> status

on_custom_callback(custon_callback_id, parameters, return_values) -> status
```

Proxy-Wasm next steps

Alignment with WASI:

- WITX,
- handles,
- `wasi-io`, `io-streams`, `io-arrays`,
- `wasi-sockets`.

New features:

- Load balancer (“next hop” selection).
- Cache control.
- Ability to follow HTTP redirects.
- STARTTLS.
- UDP datagrams.

Questions?