Integration of CommonAPI and MW COM

Building a Homogeneous Service Bus & Hybrid Binding Architecture

Agenda

- Introduction and Primary Objectives
- Misconceptions about CommonAPI
- Technical Challenges and Proposed Mitigations
- FRANCA IDL and CommonAPI Extensions
- Performance Benchmarks and Next Steps
- Future Roadmap

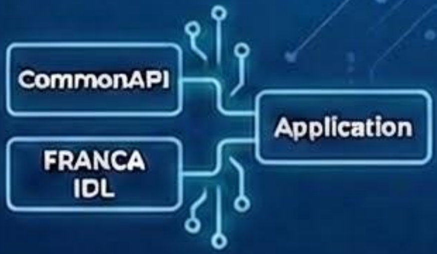
Introduction & Primary Objectives

Goal:

Provide a homogeneous service-bus supported by an efficient Interface Definition Language (IDL).

Abstracting Complexity:

The framework abstracts multiple Inter-Process Communication (IPC) implementations to handle diverse communication use-cases efficiently.



The PoC Scope:

Introduces extensions to CommonAPI and the FRANCA IDL to expose IPC-aware APIs directly to applications.

The Vision: A Homogeneous Service Bus

Logical Uniformity: Transforming a physically heterogeneous transport protocol into a logically uniform service bus.

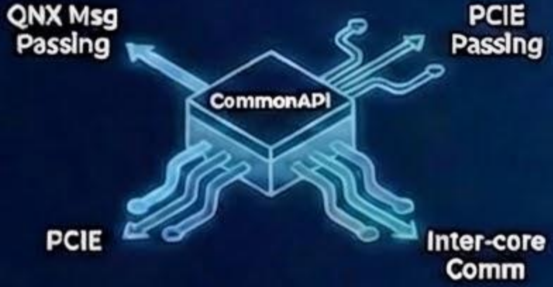
Interface Design
Implementation Details

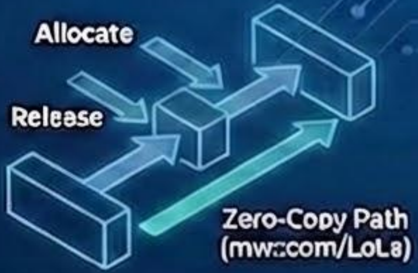
Decoupled Design: Completely separates interface design from underlying implementation details.

"Define Once, Deploy Anywhere": Services run agnostically, regardless of the transport layer (e.g., SOME/IP, D-BUS).

Enabler: Homogeneity is achieved via CommonAPI's powerful "Bindings" plugin mechanism.

Misconceptions about CommonAPI





Misconception: Latency Overhead

CommonAPI adds minimal latency overhead, the overhead usually comes from the underlying binder.

Misconception: Only for SomeIP/DBUS

Valeo Implemented a binder based on QNX Message Passing with as low as three kernel calls for the roundtrip.

Valeo is working on binders for PCIE and Inter core Communication.

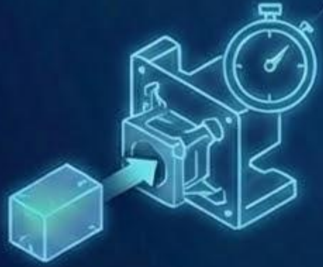
Misconception: No Zero-Copy Support

Valeo has already worked on a zero copy extension which would support buffer management through the typical "Allocate", "Release" APIs.

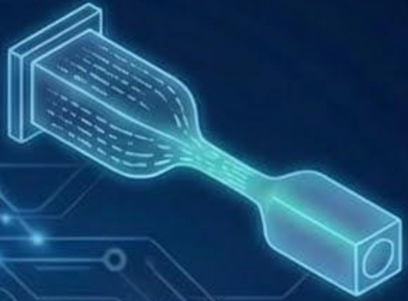
Valeo created a POC for using mw::com/LoLa as a binder under the new CommonAPI extension.

Identified Technical Challenges

The “One-Size-Fits-All” Flaw: Forcing all data through a single IPC approach creates massive performance regressions.



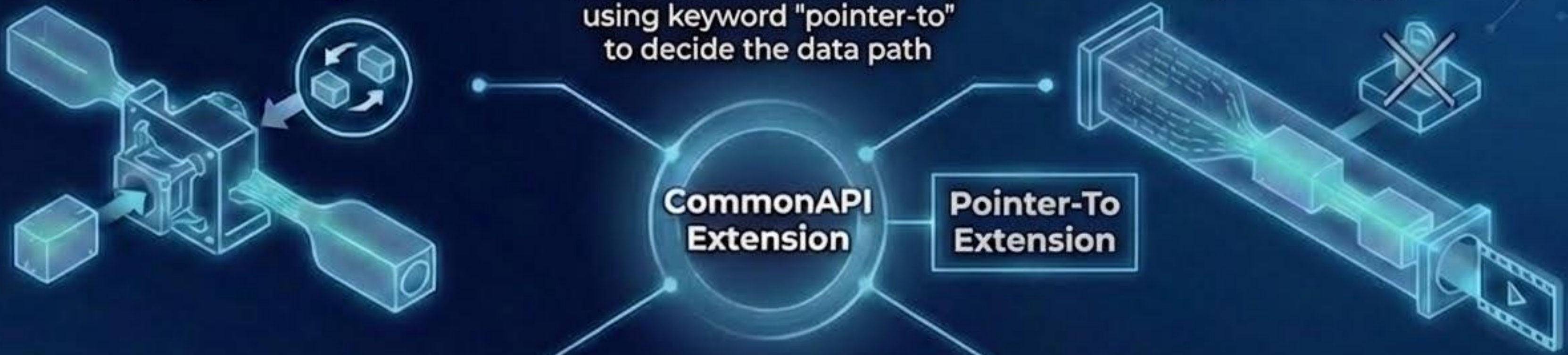
Small Data Overhead: For high-frequency, small payloads, "Zero-Copy" mechanisms introduce unnecessary overhead due to strict buffer synchronization.



Large Data Bottlenecks: For large datasets (e.g., camera frames), standard memory copy costs consume the CPU.

Hardware Constraints: Hardware-accelerated camera drivers require dedicated physical memory pools that standard middleware allocators simply cannot support.

Mitigation 1: Hybrid Binding Architecture



Path A: vOS IPC (Copy-Based)

A CommonAPI extension by using keyword "pointer-to" to decide the data path

Path B: MW::COM/LoLa IPC (Zero-Copy)

CommonAPI Extension

Pointer-To Extension

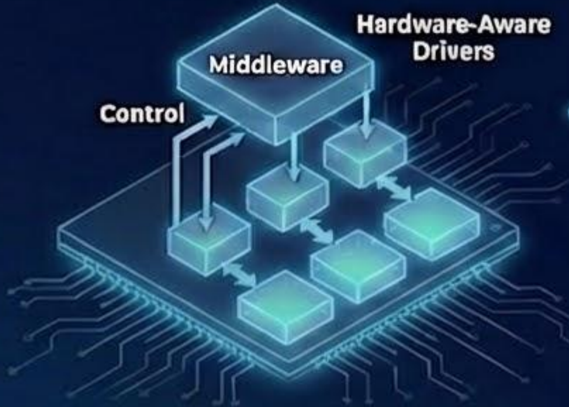
Handles small data and control signals. Reduces CPU load by avoiding complex shared memory management for tiny payloads.

A CommonAPI extension by using keyword “pointer-to” to decide the data path.

Handles high-bandwidth data (e.g., video feeds). Eliminates the massive CPU tax from multimegabyte copies, achieving 30FPS+ throughput.

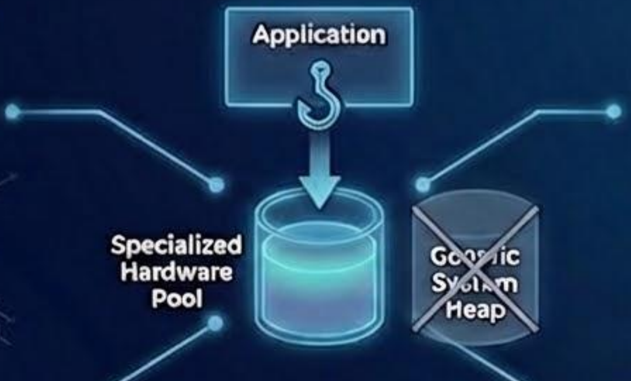
Mitigation 2: Custom Memory Management

Delegated Allocation



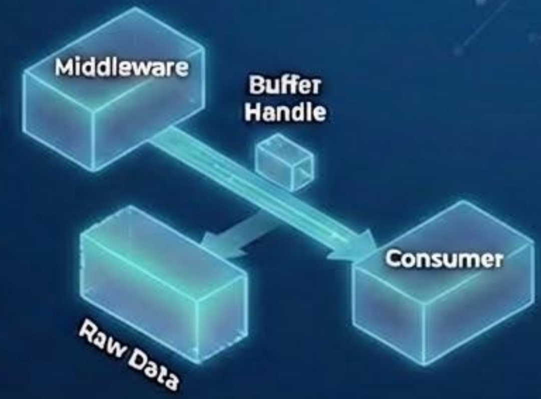
Middleware supports custom allocators at the service bus level; hardware-aware drivers handle the actual memory allocation.

Specialized Pools



Applications use an allocation hook to request memory from specialized hardware pools instead of the generic system heap.

Handle Transport

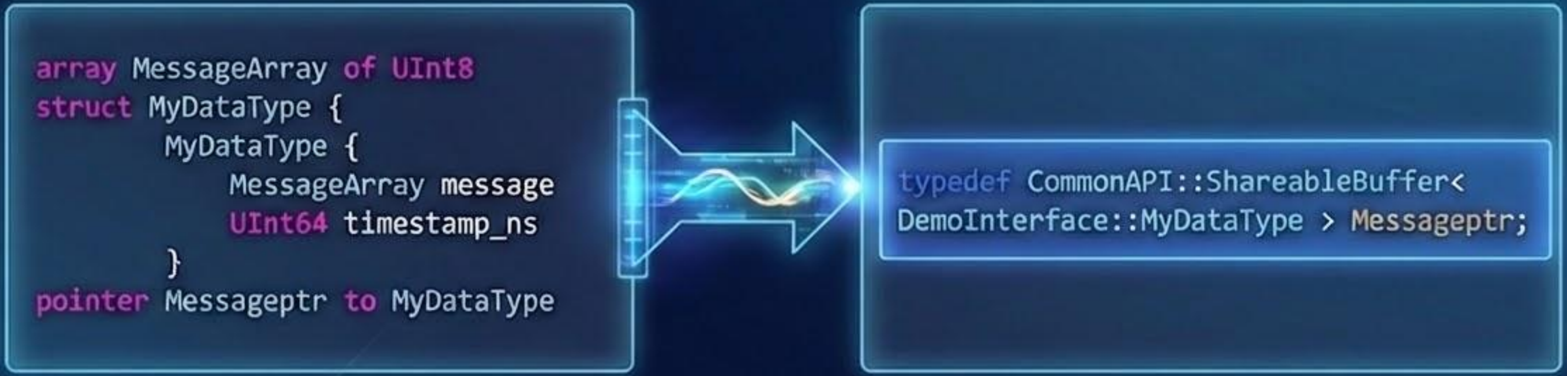


The middleware seamlessly transports just the buffer handle to the consumer, never the raw data.

The Result: A perfect, zero-overhead "Hardware-to-Middleware" handshake.

FRANCA IDL & CommonAPI Extensions

- **The `pointer-to` Keyword:**
  - Syntax extended in FRANCA IDL.
  - Automatically prepares a Zero-Copy path and generates a CommonAPI ShareableBuffer.
  - **FIDL:** **Generated:**

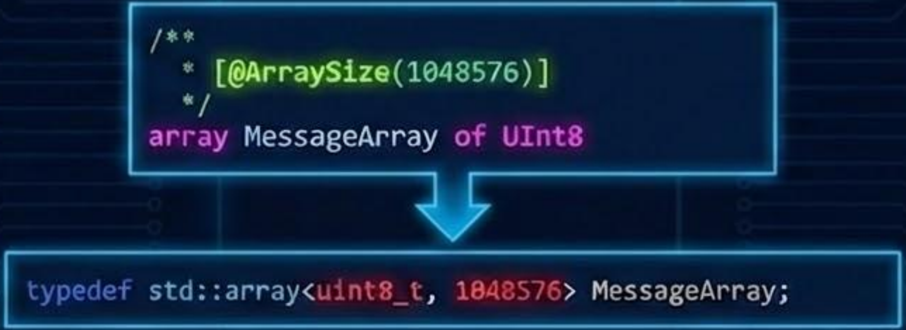


```
array MessageArray of UInt8
struct MyDataType {
        MyDataType {
            MessageArray message
            UInt64 timestamp_ns
        }
pointer Messageptr to MyDataType
```

```
typedef CommonAPI::ShareableBuffer<
DemoInterface::MyDataType > Messageptr;
```

FRANCA IDL & CommonAPI Extensions

- **The @ArraySize Annotation:**
  - Added to handle MW::COM's inability to process dynamic vector types.
  - Forces the core generator to produce a fixed-size array (std::array) instead of a dynamic vector.
  - **FIDL:**



```
/**
  * [@ArraySize(1048576)]
  */
array MessageArray of UInt8
```

  - **Generated:**

```
typedef std::array<uint8_t, 1048576> MessageArray;
```

ShareableBuffer Core Interfaces

What it is:

A highly-optimized, zero-copy wrapper acting like a custom std::unique_ptr (move-only to prevent accidental copying).

Abstraction:

A static registry completely hides the underlying middleware implementation from the application layer.

Writer-Side (Producer):

`Allocate(eventName)`: Requests an owning buffer.

Reader-Side (Consumer):

`Acquire(eventName)`: Obtains a pointer view of cached data without destroying it.

Performance Insights & Benchmarks

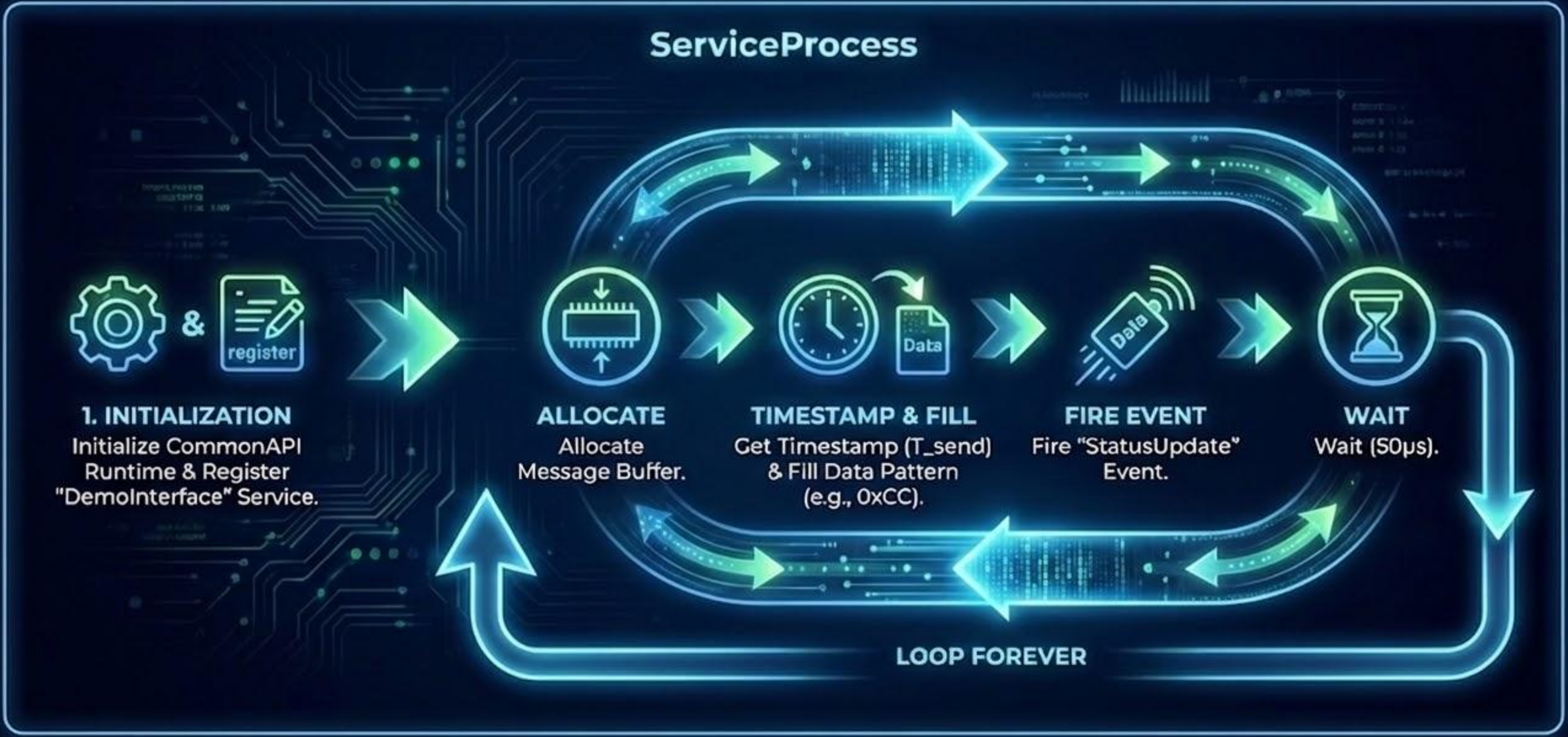
Test Environment:
One-way trip latency measured on a Raspberry PI 4 using QNX SDP 8.0 and Linux.

Standalone MW::COM:
Provides the lowest absolute latency (native communication path).

Standard CommonAPI:
Displays severe, size-dependent latency overhead and risks thread stack overflows for massive payloads.

ShareableBuffer PoC:
Successfully demonstrates stable, size-independent latency with only marginal overhead compared to raw MW::COM.

BENCHMARK SCENARIO: SERVICE PROCESS (PRODUCER)



ServiceProcess

1. INITIALIZATION
Initialize CommonAPI Runtime & Register "DemoInterface" Service.

ALLOCATE
Allocate Message Buffer.

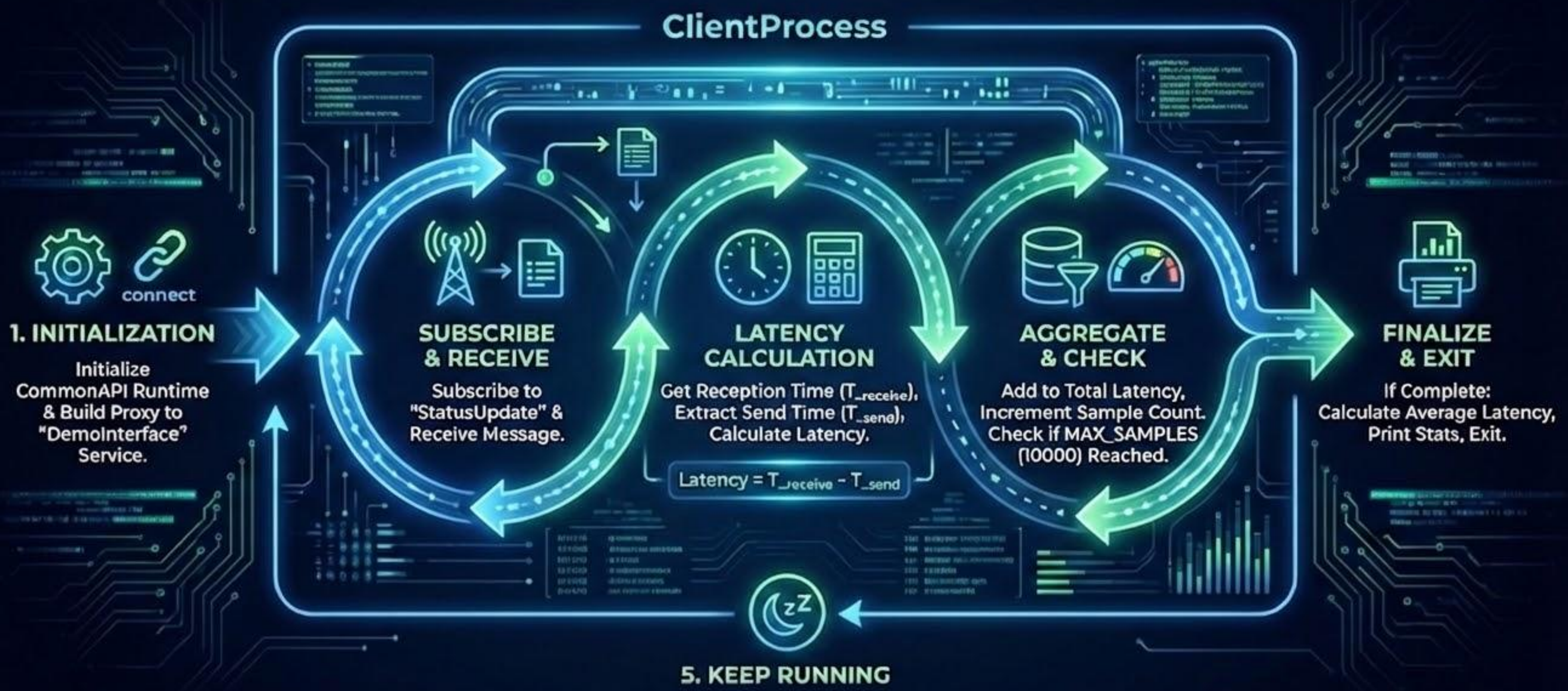
TIMESTAMP & FILL
Get Timestamp (T_send) & Fill Data Pattern (e.g., 0xCC).

FIRE EVENT
Fire "StatusUpdate" Event.

WAIT
Wait (50μs).

LOOP FOREVER

BENCHMARK SCENARIO: CLIENT PROCESS (CONSUMER)



ClientProcess

1. INITIALIZATION

connect

Initialize CommonAPI Runtime & Build Proxy to "DemoInterface" Service.

SUBSCRIBE & RECEIVE

Subscribe to "StatusUpdate" & Receive Message.

LATENCY CALCULATION

Get Reception Time (T_receive), Extract Send Time (T_send), Calculate Latency.

Latency = T_receive - T_send

AGGREGATE & CHECK

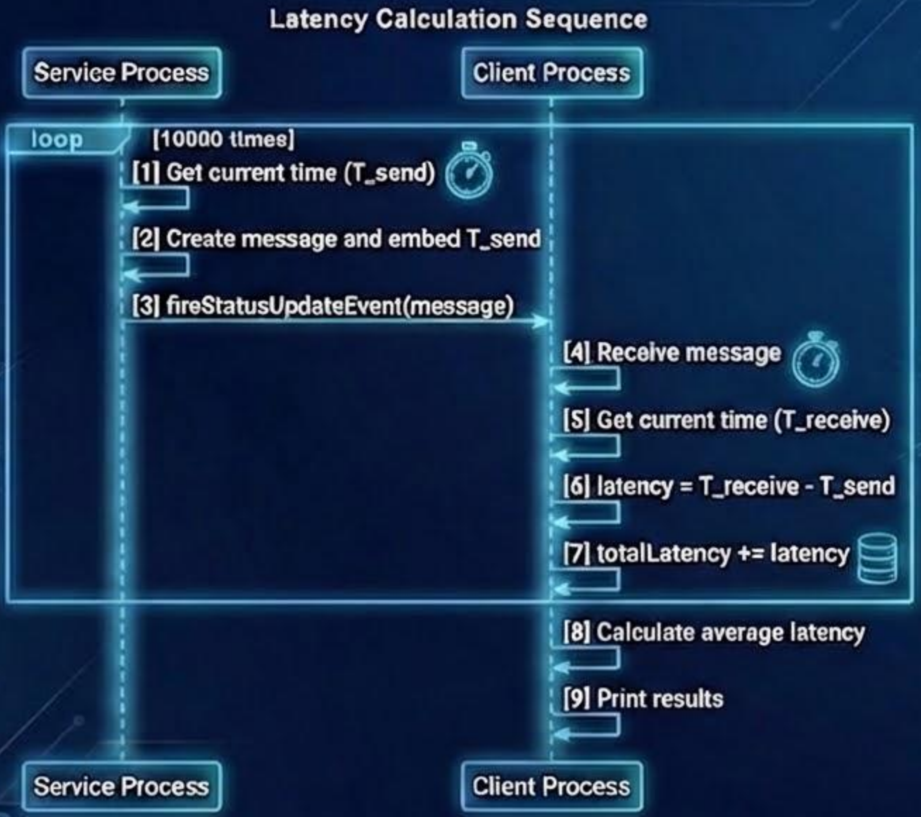
Add to Total Latency, Increment Sample Count. Check if MAX_SAMPLES (10000) Reached.

FINALIZE & EXIT

If Complete: Calculate Average Latency, Print Stats, Exit.

5. KEEP RUNNING

Benchmark scenario



Latency Calculation Sequence

Service Process

Client Process

loop [10000 times]

[1] Get current time (T_send)

[2] Create message and embed T_send

[3] fireStatusUpdateEvent(message)

[4] Receive message

[5] Get current time (T_receive)

[6] latency = T_receive - T_send

[7] totalLatency += latency

[8] Calculate average latency

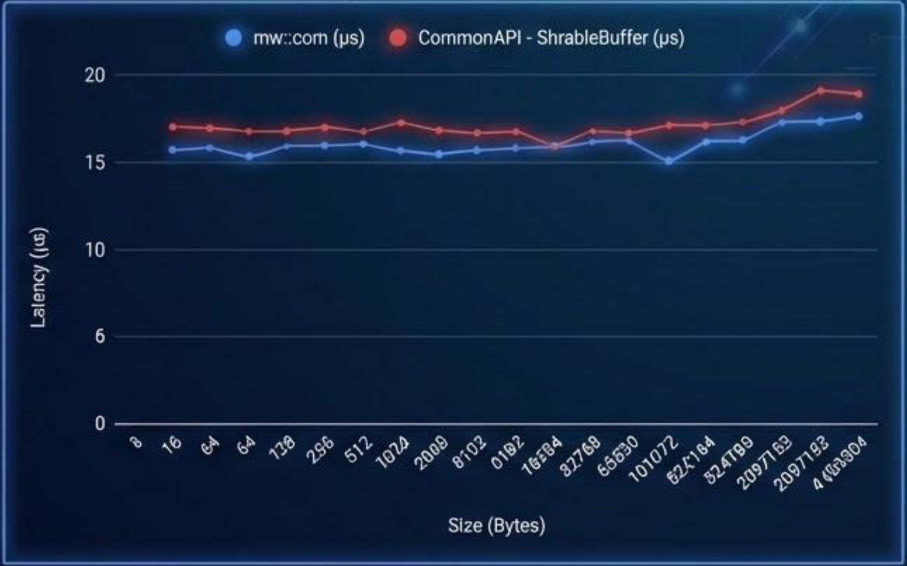
[9] Print results

Service Process

Client Process

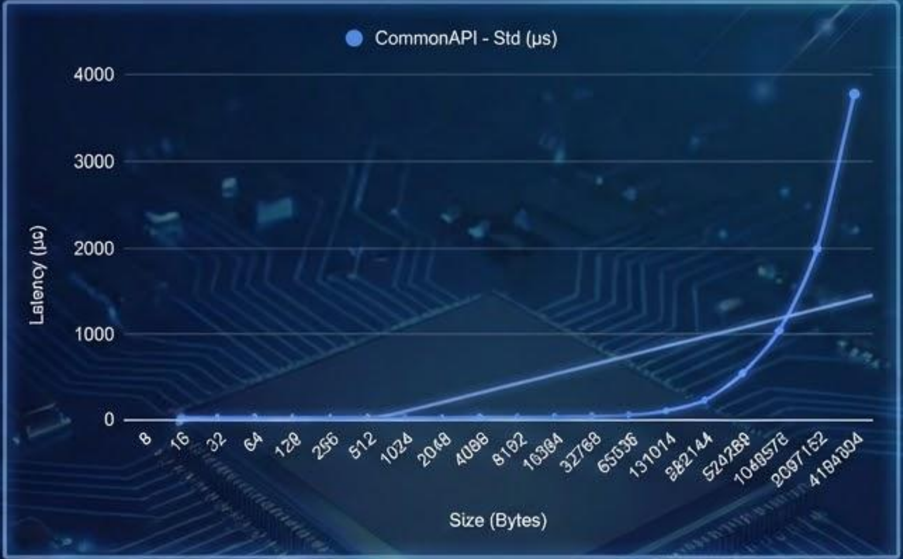
Performance Insights & Benchmarks

Size (Bytes)	mw::com (µs)	CommonAPI - Std (µs)	CommonAPI - ShrableBuffer (µs)
8			
16	15.65	23.3198	17.0263
32	15.774	23.8579	16.9493
64	15.326	23.5663	16.7933
128	15.871	23.7659	16.7804
256	18.688	23.7702	16.985
512	16.014	23.702	17.7227
1024	15.607	23.7854	16.7421
2048	15.478	25.0162	16.8178
4096	15.658	25.2988	16.7942
8192	15.764	26.18	16.7165
16384	15.863	30.0815	16.0545
33768	16.127	33.0489	16.7224
65536	16.146	S3.9821	17.7114
131072	16.12	110.579	17.1115
262144	16.167	222.489	17.1483
524288	16.326	540.099	17.3112
1048576	17.288	1048.41	18.047
2097152	17.383	1987.6	19.0679
4194304	17.655	3774.35	18.9091



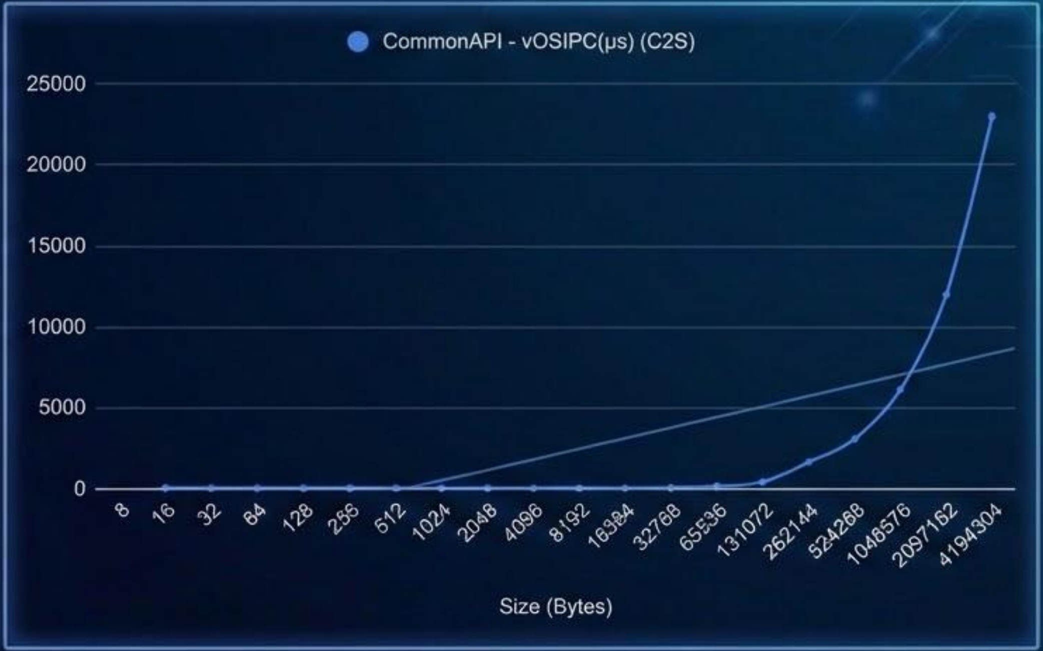
Performance Insights & Benchmarks

Size (Bytes)	mw::com (µs)	CommonAPI - Std (µs)	CommonAPI - ShrableBuffer (µs)
8			
16	15.65	23.3198	17.0263
32	15.774	23.8579	16.9493
64	15.326	23.5663	16.7933
128	15.871	23.7669	16.7804
256	18.888	23.6252	16.7327
512	16.607	23.702	17.2421
1024	15.478	25.0164	16.8178
2048	15.578	25.0162	16.8178
4096	15.658	25.2988	16.1642
8192	15.763	26.18	16.7166
16384	15.863	30.0615	16.0545
32768	16.127	36.0489	16.7294
65536	16.146	53.9825	17.1114
131072	16.12	110.571	17.1115
282144	16.167	232.489	17.1483
524288	16.326	540.059	17.3112
1048576	17.288	1048.41	18.047
2097152	17.383	1987.6	19.0679
4194304	17.655	3774.35	18.9091



Performance Insights & Benchmarks

Size (Bytes)	mw::com (µs)	CommonAPI - vOSIPC(µs) (C2S)
8		
16	15.65	11.8372
32	15.774	12.4394
64	15.326	12.26
128	15.871	12.9493
256	15.888	13.018
512	16.014	14.9713
1024	15.607	16.4933
2048	15.478	21.4796
4096	15.658	23.8879
8192	15.764	28.7723
16384	15.863	39.8903
32768	16.127	61.2448
65536	16.146	116.816
131072	15.12	382.617
262144	16.167	1630.61
524288	16.326	3071.99
1048576	17.288	6129.83
2097152	17.383	11969.6
4194304	17.655	23023.1



What’s Next?

- **Zero-Copy Polling:** Adding polling support for event data without copies at the client side.
- **Copy-Based Polling:** Adding polling support for copy-based event data with client-side caching.
- **Extended Support:** Adding support for fields and methods within the MW::COM binder.
- **Language Expansion:** Introducing support for the Rust programming language.

Zero-Copy Polling

Adding polling support for event data without copies at the client side.

Copy-Based Polling

Adding polling support for copy-based event data with client-side caching.

Extended Support

Adding support for fields and methods within the MW::COM binder.

Language Expansion

Introducing support for the Rust programming language.

Valeo

Driving Change Together

Follow us on: