Zenoh: Unifying Communication in Automotive

Unconstrained connectivity from the cloud to the microcontroller

Phani Gangula
Senior Solutions Architect
phani@zettascale.tech

Zenoh in V2X

Few months ago, we did a webinar to showcase how Zenoh can address the challenges of V2X and overcome the shortcoming of existing protocols.

We concluded the webinar by jumping on a Car and showing what Zenoh can do ☺

Source :

The Challenges

“The most dangerous phrase in the language is, ‘We’ve always done it this way.’” - Rear Admiral Grace Hopper

Fragmented Communication Landscape

Intra-node: Many IPC mechanisms, no common abstraction

Inter-node: CAN, LIN, FlexRay, SOME/IP, DDS

Cloud: MQTT/HTTPS, proprietary REST

Each layer has different semantics, discovery, QoS, and tooling

Results in high integration effort

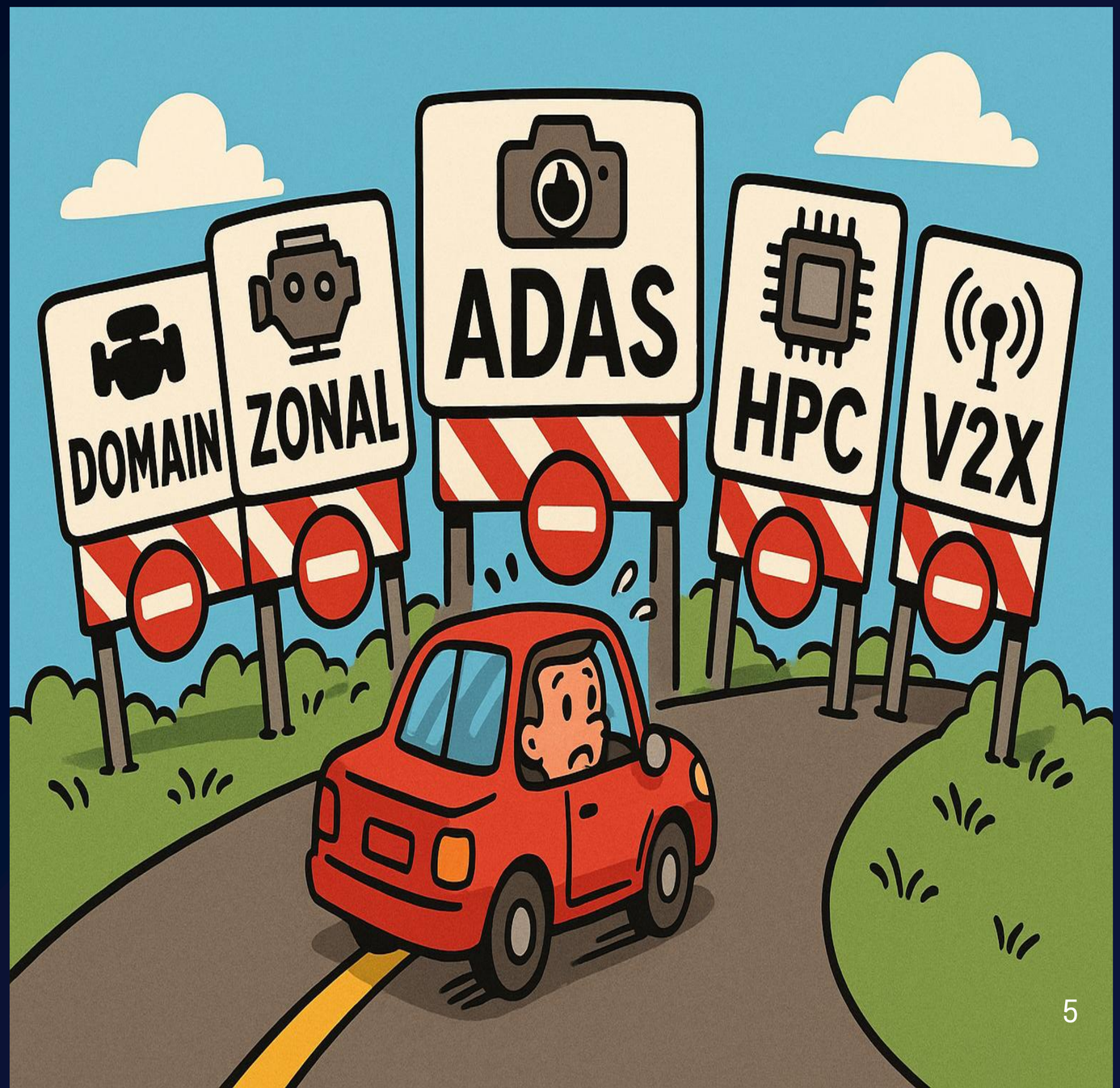
Segment-Bound Architectures

Communication stacks remain Domain-only, Zonal-only, ADAS-only, HPC-only, Telemetry-only, V2X-only

No unified way to move pub/sub data, stored data, or computation results across segments

Gateway chains multiply: HPC → domain → zone → cloud → V2X

Leads to latency, jitter, inconsistent behavior, and data silos



Scalability, Resources & Security

Discovery and service scaling break under large topic/service counts

QoS behavior changes when crossing protocol boundaries

Many MCUs lack processing / Memory to run heavy communication stacks

Security models differ by protocol → no end-to-end enforcement

Validation effort grows rapidly with every protocol interface

The Digital Frankenstein

Multiple technologies are stitched together only to make data flow end-to-end

Few more have to be packed-up to deal with data storage…

Not to mention computations

Simplicity does not precede complexity, but follows it – A. Perils

The Solution

"The best way to predict the future is to invent it." – Alan Kay

Meet our amazing Technology

Zenoh

Next generation protocol that unifies data at rest, data in movement and computations from the data-centre to the micro-controller.

Zenoh

Pub/Sub/Query protocol that Unifies data in motion, data at rest and computations from embedded microcontrollers up to the data centre

Provides location-transparent abstractions for high performance pub/sub and distributed queries across heterogeneous systems

Built-in support for Shared Memory/Zero-Copy as well as built-in support for distributed storage alignments.

Runs Everywhere

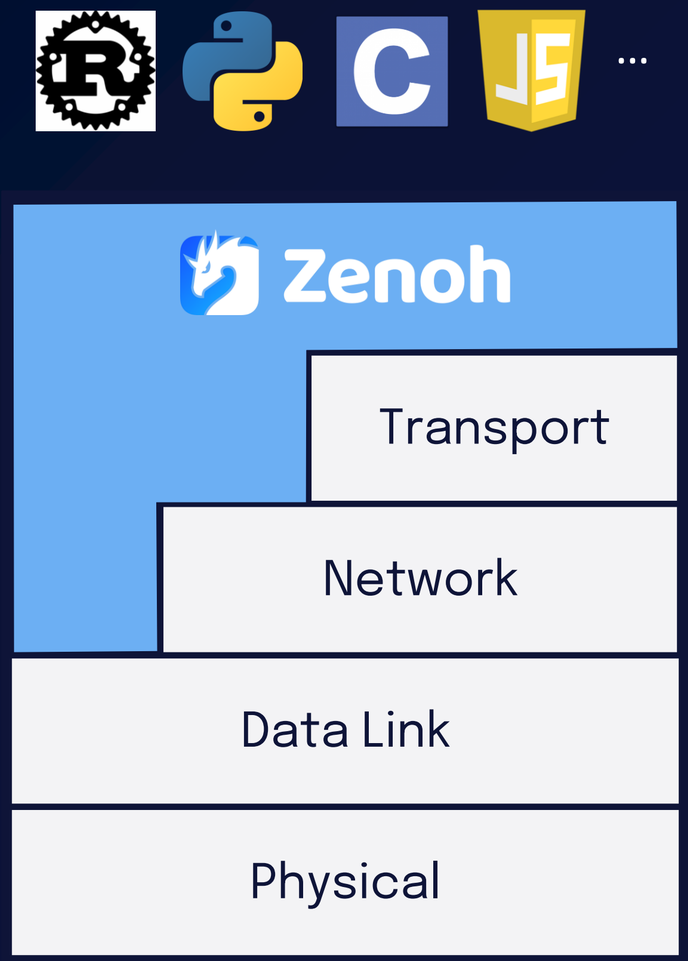
Written in Rust for security, safety and performance

Native libraries and API bindings for many programming languages, e.g., Rust, C/C++, Python, JS, REST, C#, Go and Kotlin

Built-in support Shared Memory and Zero Copy

Supports network technologies from transport layer down-to the data link. Currently runs on, TCP/IP, UDP/IP, QUIC, Serial, Bluetooth, OpenThread, Unix Sockets.

Available on embedded and extremely constrained devices and networks – 5-7 bytes minimal overhead

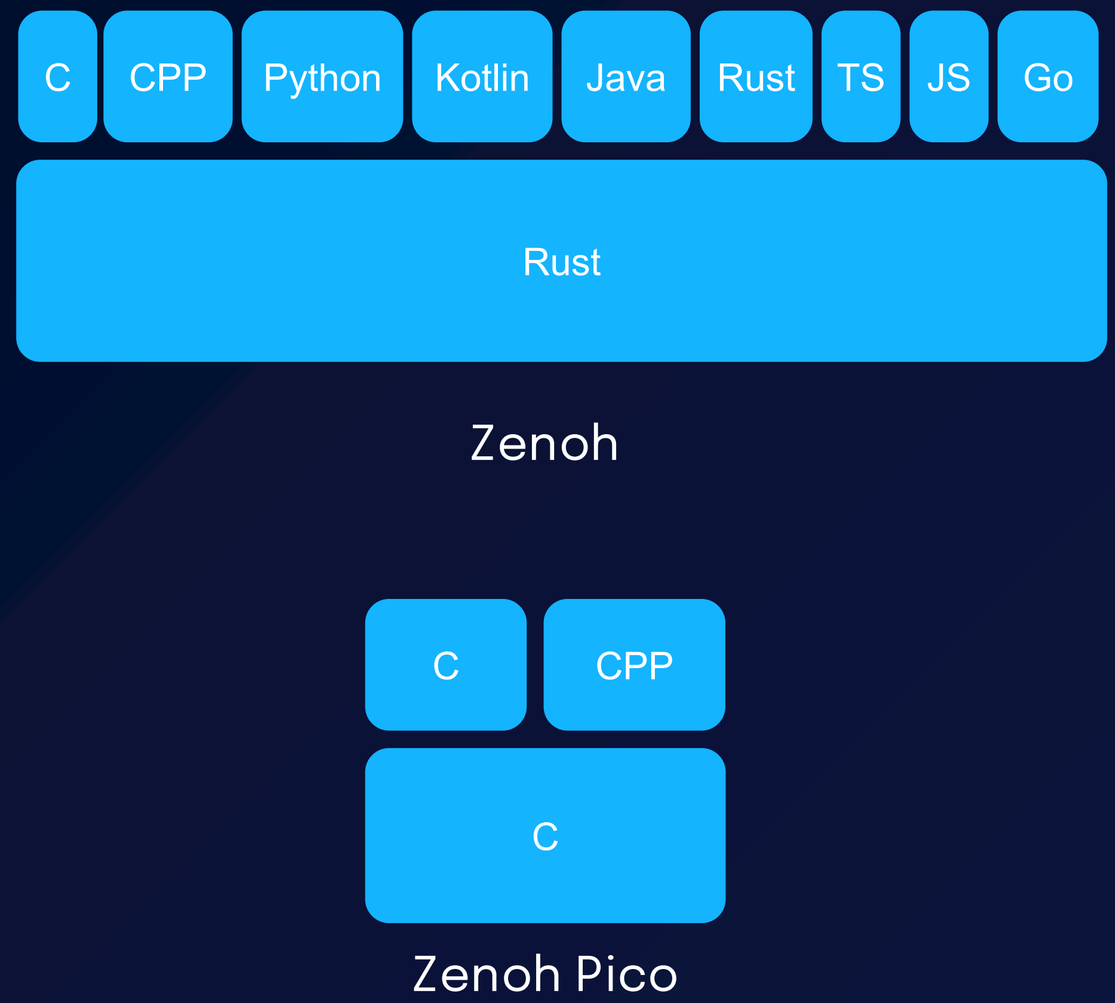


Zenoh Implementations

ZettaScale leads two implementations of the Zenoh protocol

Zenoh written in Rust and Zenoh Pico written in C and targeting micro-controllers

When using C/C++ APIs, applications can target either of these versions – it is a compile-time decision



NEW

Zenoh Rust Bare Metal

Pure Rust implementation of Zenoh targeting bare-metal and microcontroller environments in general

The implementation is no-std and no-alloc

Repository:

```rust
▶ Run | ⚙ Debug
#[esp_rtos::main]
async fn main(spawner: Spawner) {
    let scale_z: f64 = SCALE_Z.unwrap_or("1.0").parse().unwrap_or(1.0);
    let scale_y: f64 = SCALE_Y.unwrap_or("1.0").parse().unwrap_or(1.0);

    zenoh_nostd::info!("zenoh-nostd DEMO!");

    let (stack, i2c) = init_esp32(spawner).await;
    let mut mpu = MPU6050::new(i2c);
    mpu.write_field(TempDisable::Enable).await.unwrap();
    mpu.write_field(GyroFullScaleRange::FS500).await.unwrap();
    mpu.write_field(AccelFullScaleRange::FS2).await.unwrap();
    mpu.write_field(SleepMode::WakeUp).await.unwrap();

    let mut session = zenoh_nostd::open!(
        zenoh_nostd::zconfig!(
            PlatformEmbassy: (spawner, PlatformEmbassy { stack: stack }),
            TX: 2048,
            RX: 2048,
            SUBSCRIBERS: 2
        ),
        EndPoint::try_from(CONNECT.unwrap_or("udp/192.168.21.90:7447")).unwrap()
    )
```

zenoh

Rust
(no-std, no-alloc)

Microcontroller

zetta scale

Runs Everywhere

OS
Linux, MacOS, Windows, QNX

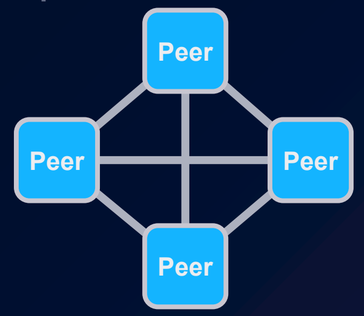
Embedded Targets
Arduino, ESP32, mbed, Zephyr

Automotive Targets
AUTOSAR Classic (microSAR)

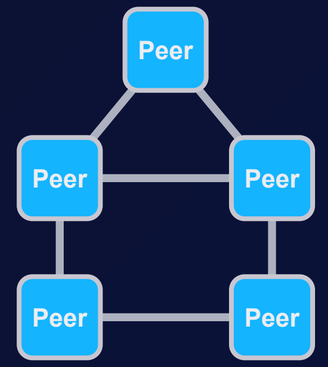
Any Topology

Peer-to-peer
Clique and mesh topologies

Clique



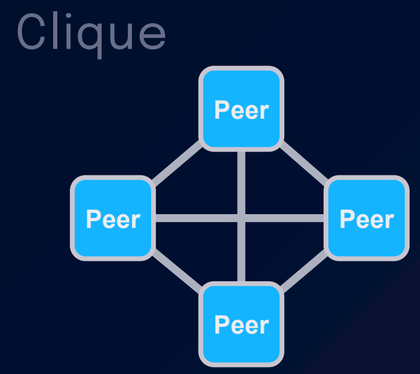
Mesh

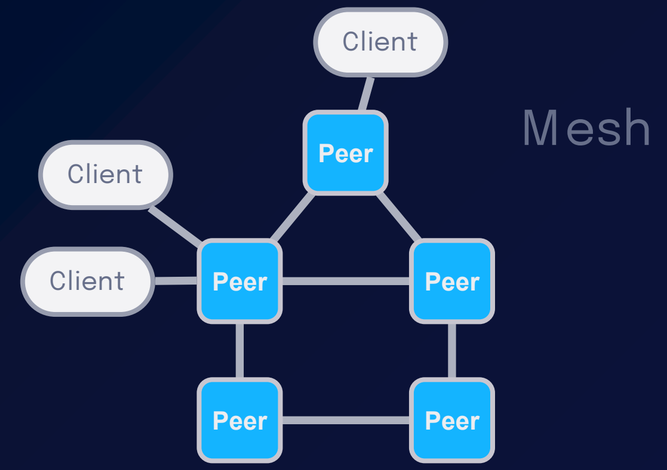


Any Topology

Peer-to-peer

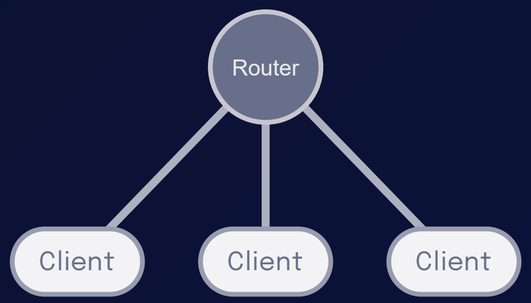
Clique and mesh topologies

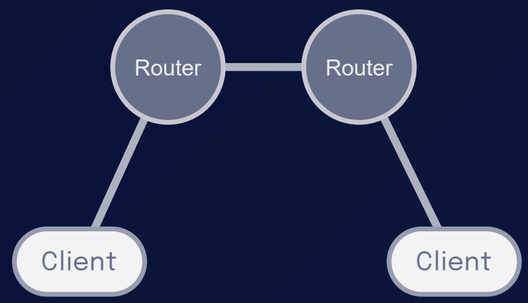




Brokered

Clients communicate through a router or a peer





Any Topology

Peer-to-peer

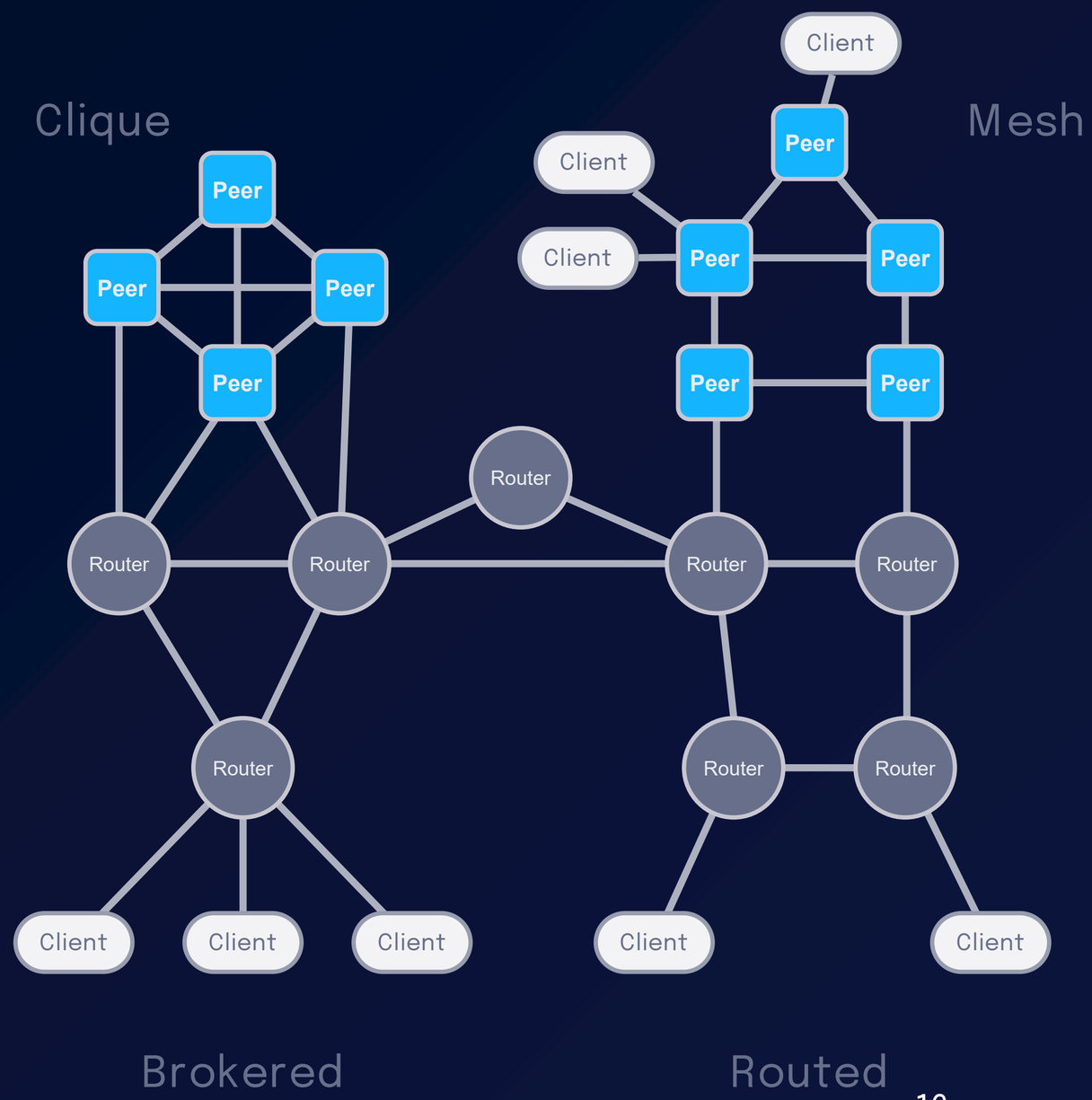
Clique and mesh topologies

Brokered

Clients communicate through a router or a peer

Routed

Routers forward data to and from peers and clients

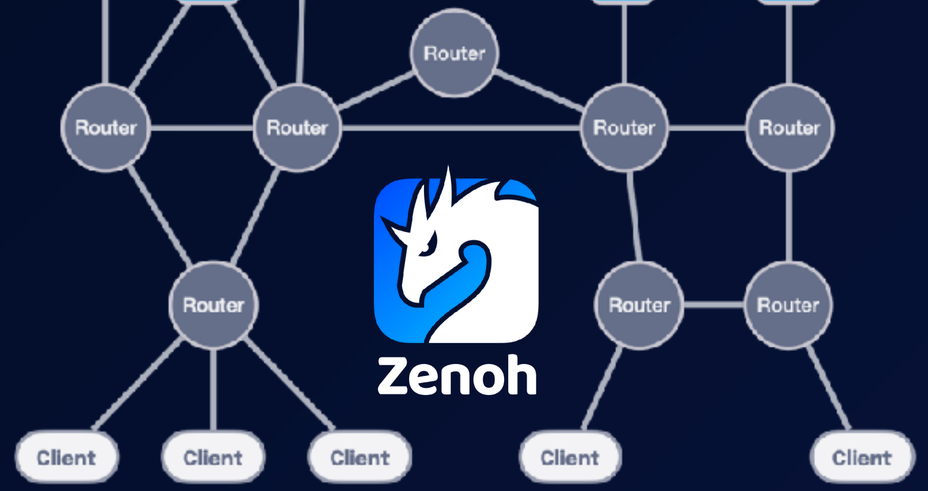


Topology in Perspective

DDS

Clique

Mesh



Brokered

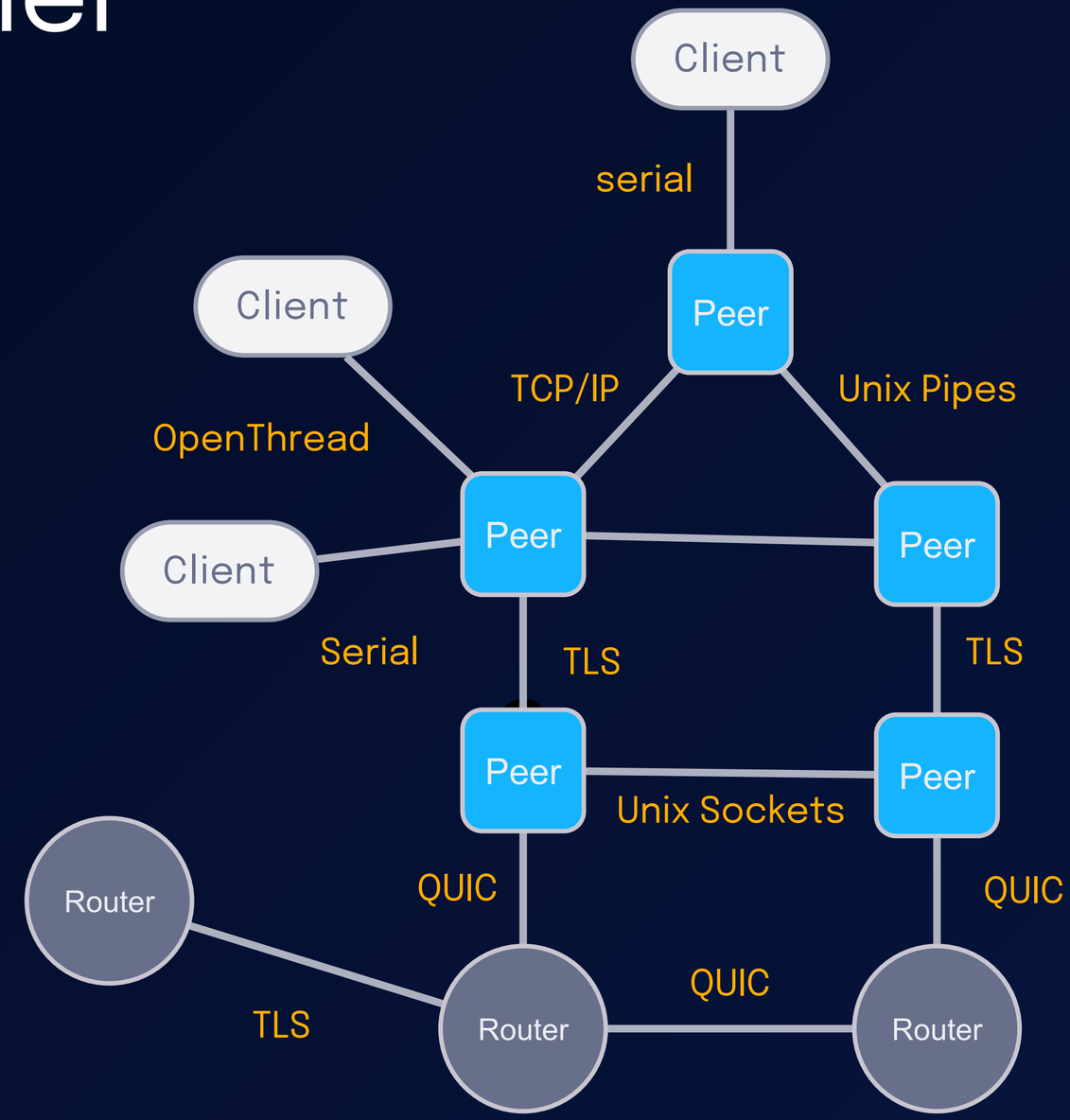
Routed

MQTT

Putting it all Together

Each Zenoh application can transparently run across multiple links at the same time

All of this transparently to the application

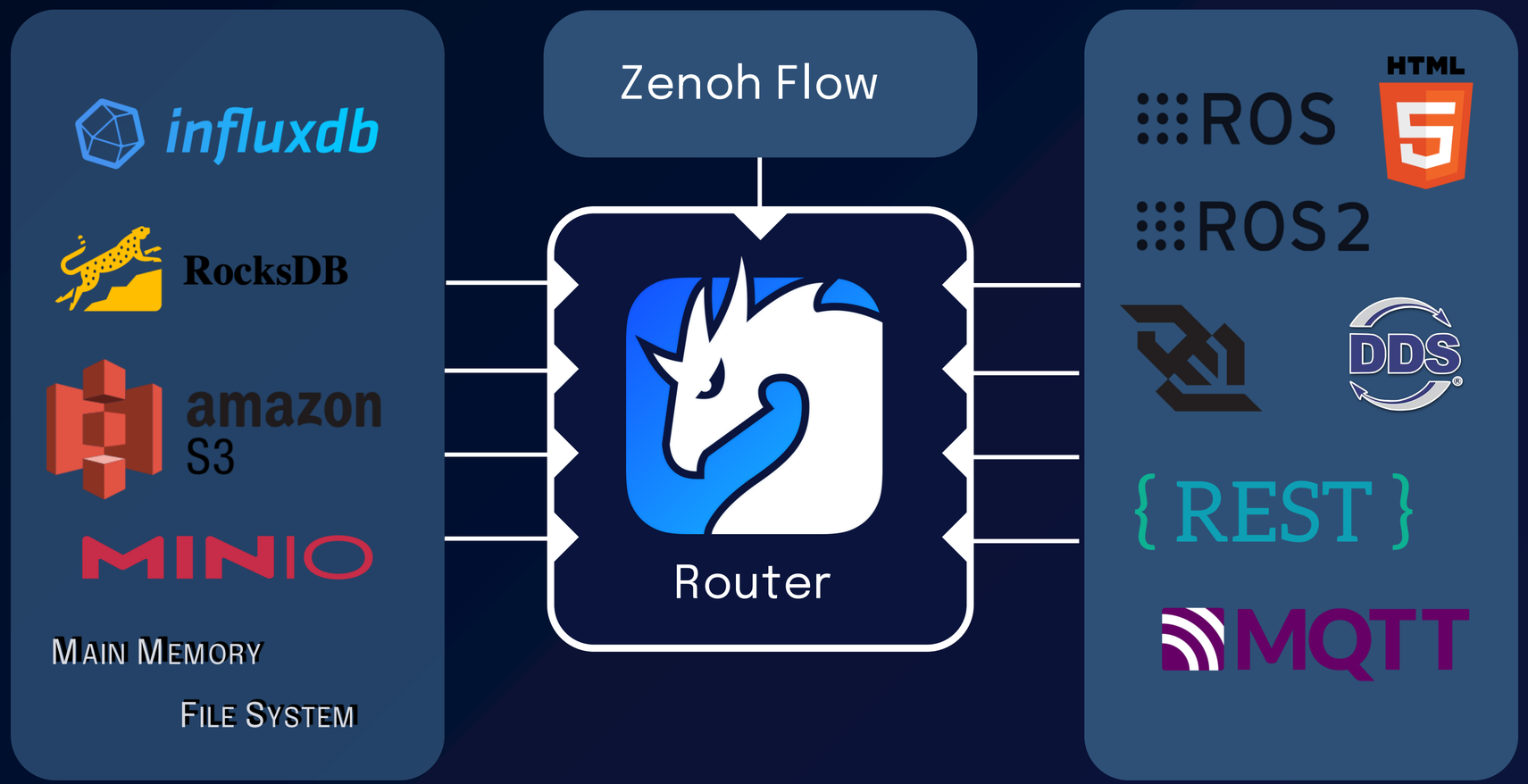


Plug-Ins

Storage Plugins

Runtime Plugins

Protocol Plugins

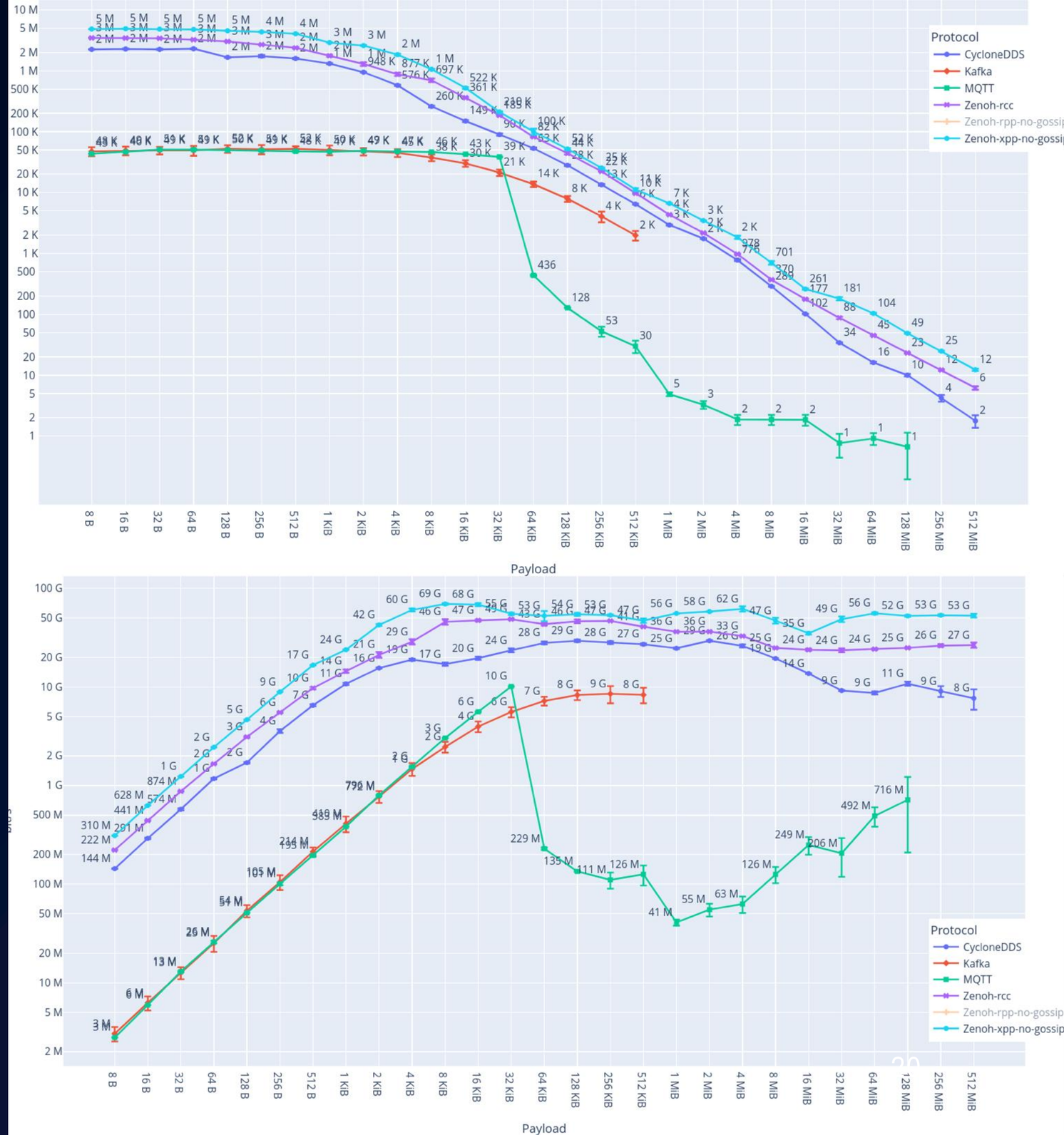


Zenoh vs DDS, MQTT & Kafka

Zenoh can deliver at peak performance of ~70Gbps at 8Kb payload:

- 3.3x higher than DDS
- 23x higher than Kafka
- 35x than MQTT (higher for larger payload)

Zenoh's latency 7-10 us with ultra-low latency support
- 25us for MQTT
- 75us for Kafka
- 8us DDS



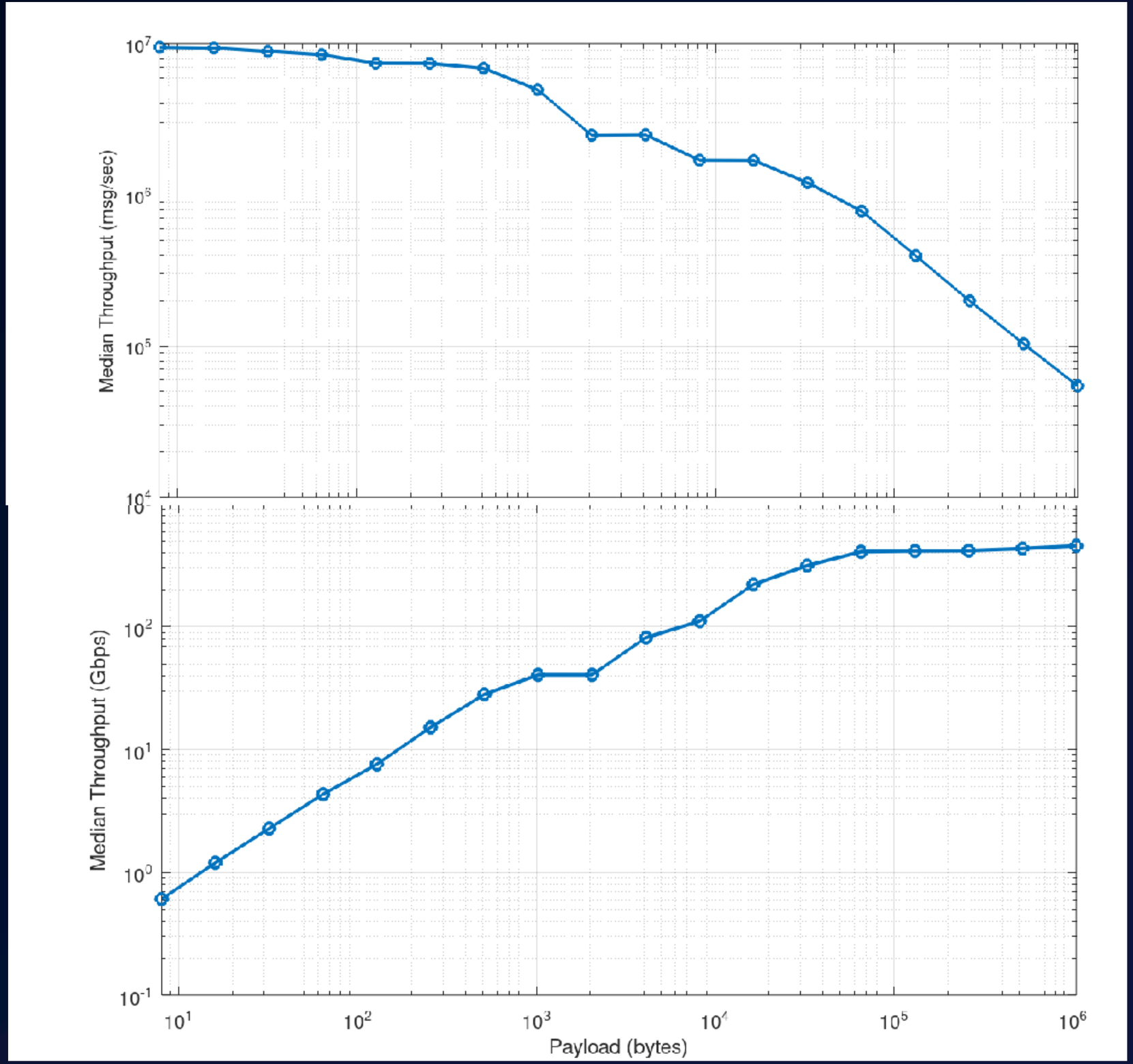
More Performance

Localhost

11M mgs/sec for 8 bytes payloads

1.2 Gbps for 16 bytes payload

410 Gbps for 64K payload



Adoption in OpenSource Frameworks

Eclipse SDV

Zenoh selected as one of the communication middleware in addition to MQTT

uProtocol

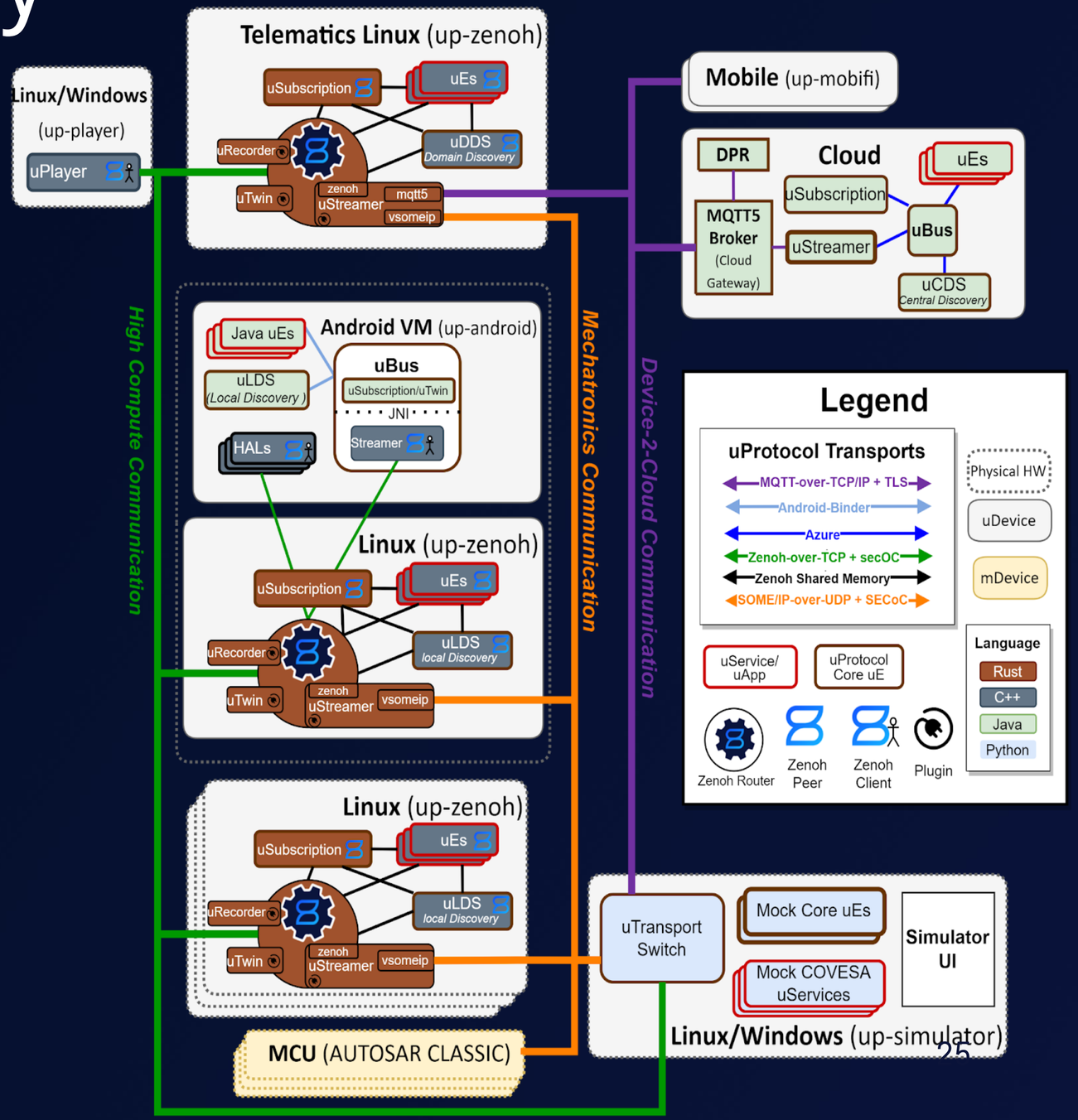
Eclipse uProtocol selected Zenoh as the first protocol to be integrated

uProtocol Topology

Device-to-device communication in Zenoh is covered by the green lines

Shared Memory communication in Zenoh is covered by the black lines

Some-IP to Zenoh mapping through plugin mechanism of Zenoh Router in uStreamer.



Zenoh: The RMW Alternate

Zenoh has been selected as the first non-DDS protocol to be natively supported in ROS2

Intrinsic committed to have Zenoh RMW Tier-1 for the May 2025 Release!

Full Report:

2023-09 ROS 2 RMW alternate

Conclusion

Requirements from ROS 2 users were gathered, and middleware options that are available today were investigated. The research has concluded that Zenoh best meets the requirements, and will be chosen as an alternative middleware. Zenoh was also the most-recommended alternative by users. It can be viewed as a modern version of the TCPROS implementation, and meets most of the ROS 2 requirements. There are still a number of design decisions to be made regarding this implementation; those details will be discussed on https://discourse.ros.org as development begins.

Best Protocol

Zenoh ranked as the #1 for next generation robotics in a recent evaluation performed by Alphabet Intrinsic and the Open Robotics Foundation

Requirements		Middleware Options										
Item	Level	Cyphal	DDS	eCal	Kafka	LCM	MQTT	NNG	OPC-UA	TCPROS	Zenoh	ZeroMQ
Pub/Sub	Must	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes but requires a special broker	Yes	Yes	Yes
Security - Encryption	Must	No	Yes	No	Yes https://kafka.apache.org/documentation/#security	No	Yes, TLS	Yes, TLS	Yes	Via SROS	Yes, TLS	Not in the default, but CurveMQ builds on top to provide it
Security - Authentication	Must	No	Yes	No	Yes	No	Yes	Yes	Yes	Via SROS	Yes	Not in the default, but CurveMQ builds on top to provide it
Security - Access Control	Must	No	Yes	No	Yes	No	Yes	No	Yes	Via SROS	No, planned for later this year	No, but can be implemented on top
Gracefully handle disconnect/ reconnect of the network	Must	Yes	Borderline; in UDP mode, packets may be dropped. If the connection is reliable, data will be resent, but may be fragmented and subject to kernel limits. In best-effort mode, data will be dropped.	Borderline; in UDP mode (the default), packets will be dropped. In TCP mode, it will handle this better, but this is not the default	Yes. Handles failures well too https://kafka.apache.org/documentation/#basic_ops_restarting	Yes (connectivity will recover)	Yes	Yes	Yes	Yes	Yes	Yes
Tolerance to bandwidth changes	Must	Unclear	Borderline: without flow controllers, can struggle to send data if the link degrades.	Borderline; in UDP mode (the default), packets will be dropped. In TCP mode, it will handle this better, but this is not the default	Yes. Esp due to writing to disk	No built-in flow control	Nothing on top of what TCP offers	Nothing on top of what TCP offers	Nothing on top of what TCP offers	Nothing on top of what TCP offers	Yes	TCP
Configure which interface to use	Must	Yes (IP address)	Yes	Yes (via standard Linux configuration)	Yes	Yes	Yes, implementation dependant: See mosquitto	No, there doesn't seem to be an API to indicate interface	Yes	Indirectly through the routing table	Yes, configuration variable	Yes (both tcp and udp during zmq_bind())
Ability to send multi-megabyte messages	Must	Yes (with UDP transport options)	Borderline; in UDP mode, packets are heavily fragmented and sent. Without configuring kernel buffer sizes, very possible to lose some packets along the way and thus have to retransmit a bunch.	Yes, but with caveats depending on which transport layer is in use: - SHM: Yes - UDP: same issues as DDS - TCP: Yes	Yes	Yes	To the limit: 8mb - 10hz (from the Zenoh performance comparison at https://zenoh.io/blog/2023-03-21-zenoh-vs-mqtt-kafka-dds/)	Yes	Yes	Yes	Yes, relying on this analysis	Should be since TCP, see https://github.com/jeffbass/imagezmq#why-use-imagezmq
Ability to send fast small messages	Must	Yes (with UDP transport option)	Yes	Borderline; particularly with the SHM transport (default for localhost), if the publisher is running faster than the subscribers, data will be lost.	Yes. Apparently latency of 2ms is achievable	Yes	Yes	Yes	Maybe not	Yes	Yes, relying on this analysis	Yes
Restart discovery without restarting all nodes	Must	Yes since the channels are statically configured.	Yes	Yes (UDP multicast discovery)	Yes	Yes	Yes	N/A (discovery is not builtin)	N/A	No	Yes: Discovery happens when a zenoh session is created in each application. It looks for other peers/routers.	N/A
Cross platform support	Must	Linux, Windows, macOS	Implementation -specific	Linux, Windows, macOS	Java, so in theory runs anywhere a JVM does	Linux, Windows, macOS	Implementation -specific	Linux, Windows, macOS	Implementation -specific	Linux, Windows (third-party), macOS (third party)	Linux, Windows, macOS	Linux, Windows, macOS
OSI approved permissive license	Must	MIT	Implementation -specific	Apache 2.0	Apache 2.0	LGPL	Implementation -specific	MIT	Implementation -specific	BSD	Apache 2.0 and Eclipse Public License 2.0	MPL 2.0

May. 2025

Kilted Kaiju

Zenoh is now Tier-1 in Kilted Kaiju –it is the only non-DDS RMW to be Tier-1

This happens as the same time as ROS-1 goes end-of-life

rmw_zenoh in production

zetta scale

rmw_zenoh: Solving Autoware's Communication Challenges

Handles high-bandwidth sensor data (LiDAR, cameras) with ease

Scales effortlessly in complex, distributed Autoware systems

Unifies vehicle, edge, and cloud communication under one protocol

Stays reliable through mobility, handovers, and intermittent networks

THE AUTOWARE FOUNDATION

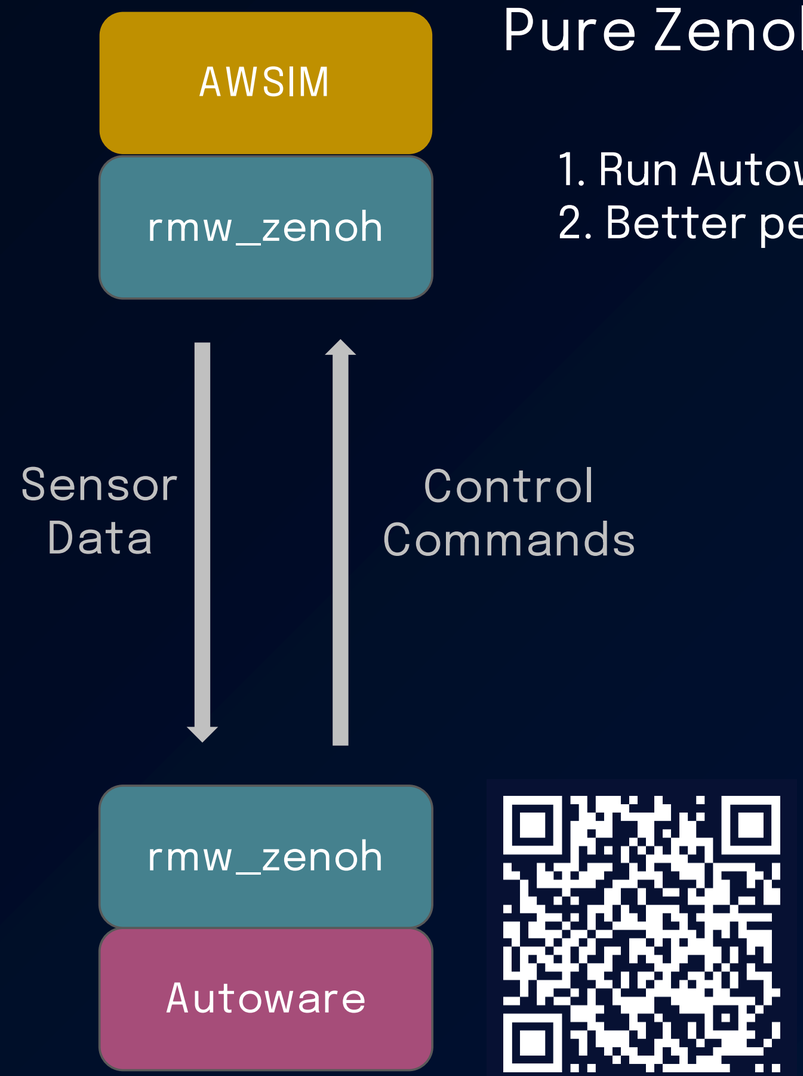
Demo

Two demo scenarios

zetta scale

Pure Zenoh (rmw_zenoh)

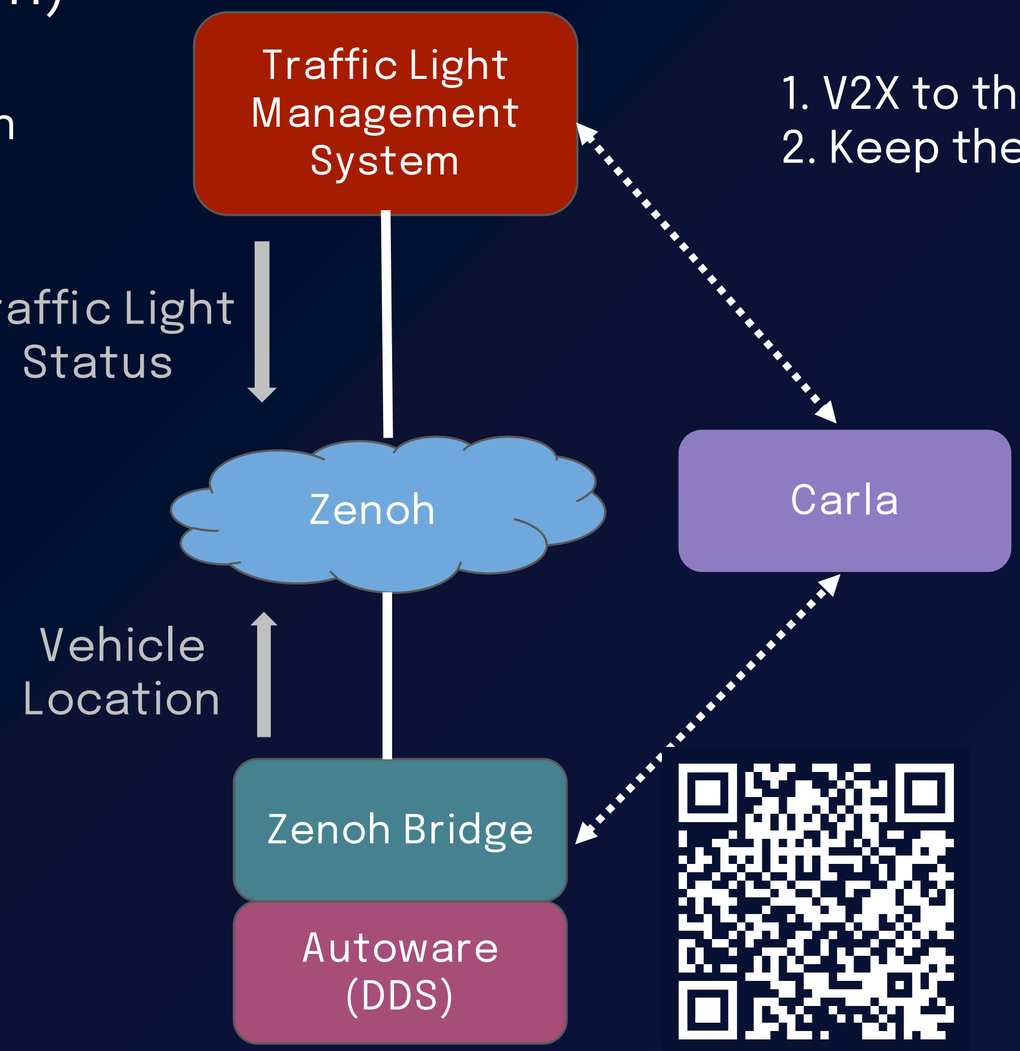
1. Run Autoware in simulation
2. Better performance



Scenario 1:

Zenoh Bridge

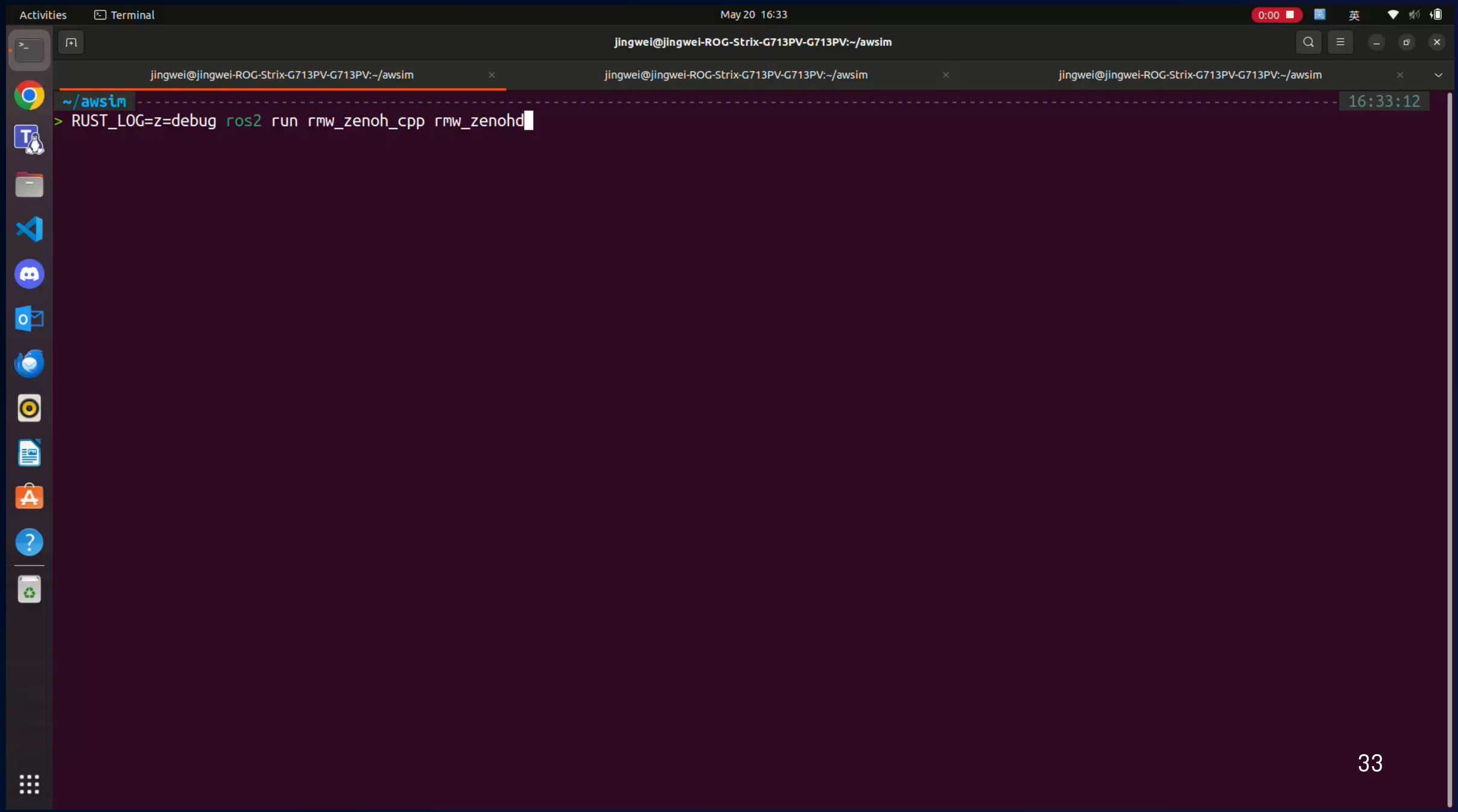
1. V2X to the traffic light
2. Keep the compatibility



Scenario 2:

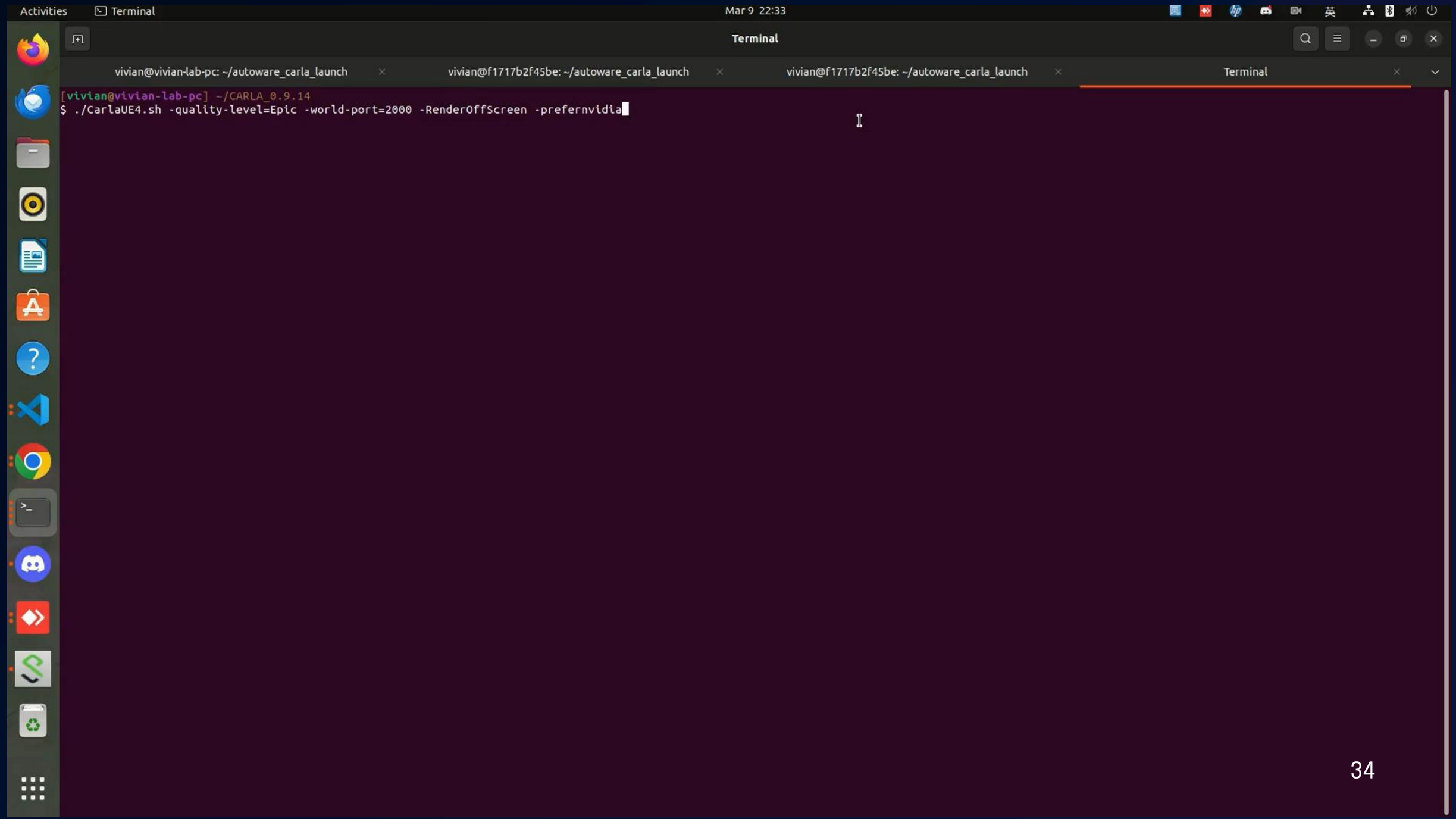
rmw_zenoh: Autoware in AWSIM

zetta scale



Zenoh Bridge: V2X scenario in Carla

zetta scale



Key Highlights

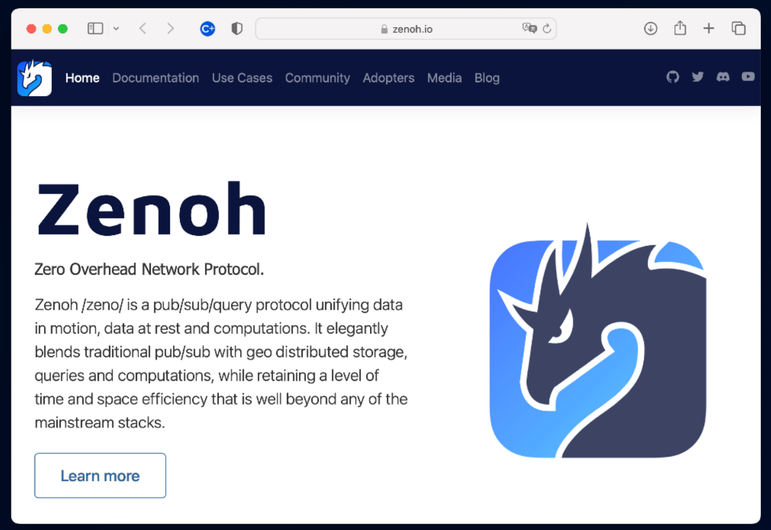
Zenoh is one of a kind protocol – it unifies data in motion, data at rest and computations

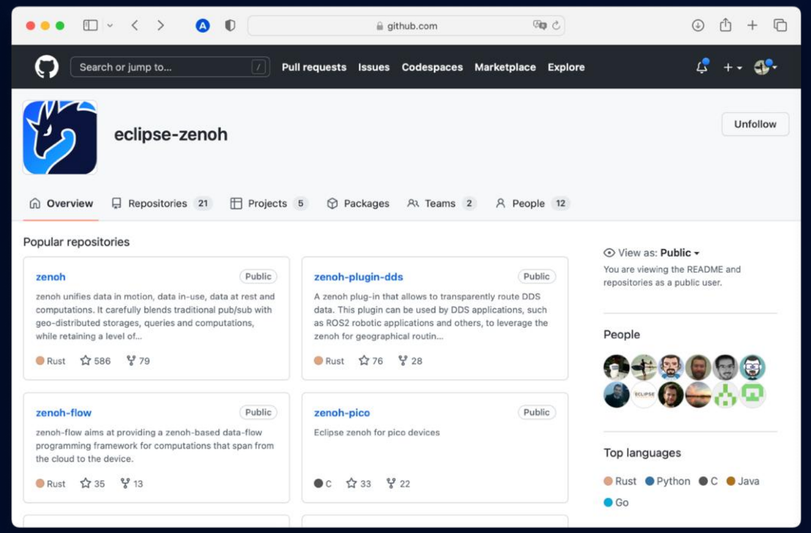
It is the only protocol able to run from the microcontroller up to the datacenter

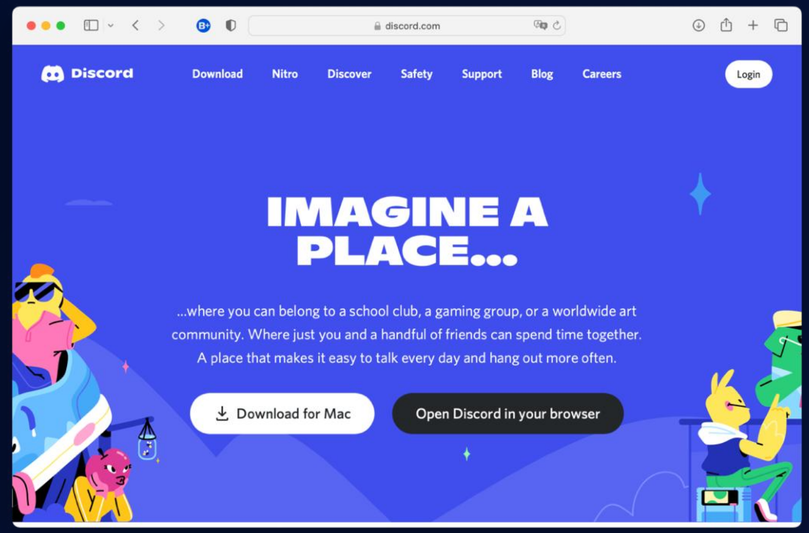
It has not topological constraints

It is extremely easy to use!

Follow-us


zenoh.io
Home Documentation Use Cases Community Adopters Media Blog
Zenoh
Zero Overhead Network Protocol.
Zenoh /zeno/ is a pub/sub/query protocol unifying data in motion, data at rest and computations. It elegantly blends traditional pub/sub with geo distributed storage, queries and computations, while retaining a level of time and space efficiency that is well beyond any of the mainstream stacks.
Learn more


github.com
Search or jump to...
Pull requests Issues Codespaces Marketplace Explore
eclipse-zenoh
Unfollow
Overview Repositories 21 Projects 5 Packages Teams 2 People 12
Popular repositories
zenoh Public
zenoh unifies data in motion, data in-use, data at rest and computations. It carefully blends traditional pub/sub with geo-distributed storages, queries and computations, while retaining a level of...
Rust 586 79
zenoh-plugin-dds Public
A zenoh plug-in that allows to transparently route DDS data. This plugin can be used by DDS applications, such as ROS2 robotic applications and others, to leverage the zenoh for geographical routin...
Rust 76 28
zenoh-flow Public
zenoh-flow aims at providing a zenoh-based data-flow programming framework for computations that span from the cloud to the device.
Rust 35 13
zenoh-pico Public
Eclipse zenoh for pico devices
C 33 22
View as: Public
You are viewing the README and repositories as a public user.
People
Top languages
Rust Python C Java
Go


discord.com
Discord
Download Nitro Discover Safety Support Blog Careers
Login
IMAGINE A PLACE...
...where you can belong to a school club, a gaming group, or a worldwide art community. Where just you and a handful of friends can spend time together. A place that makes it easy to talk every day and hang out more often.
Download for Mac
Open Discord in your browser

zetta scale

Thank You

Patience, persistence and perspiration make an unbeatable combination for success.