rmw_zenoh

An alternative open-source middleware for ROS 2

Yadunund Vijay
Julien Enoch
ROSCon 2024, Odense

ROS™

Yadunund Vijay

Senior Software Engineer @ Intrinsic

GitHub @yadunund

LinkedIn @yadunundvijay

Julien Enoch

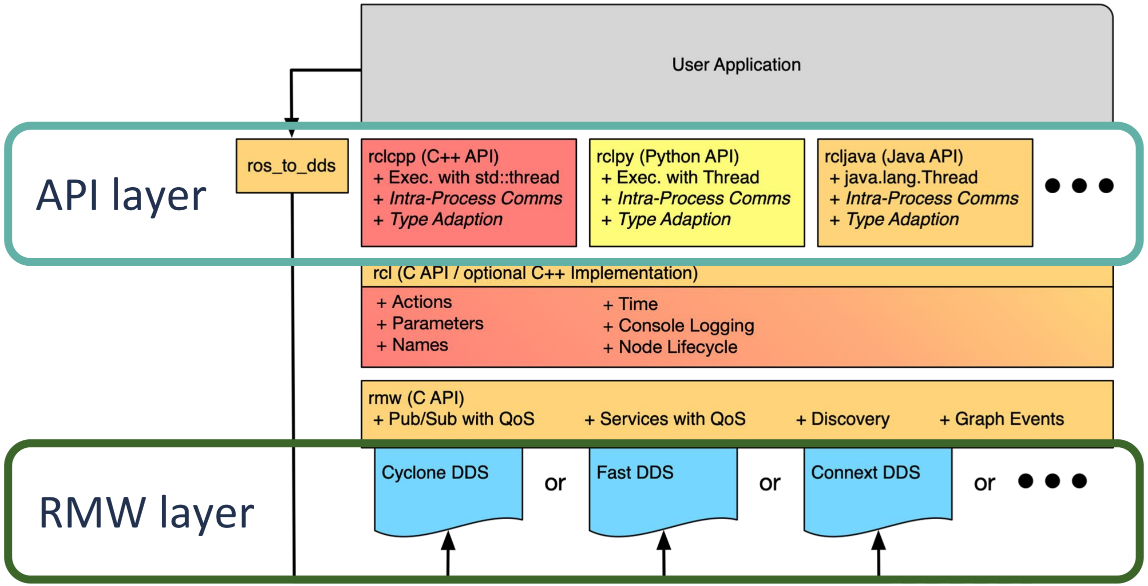
Senior Solution Architect @ ZettaScale

GitHub @JEnoch

LinkedIn @julienenoch

https://design.ros2.org

ROS 2 has a modular architecture



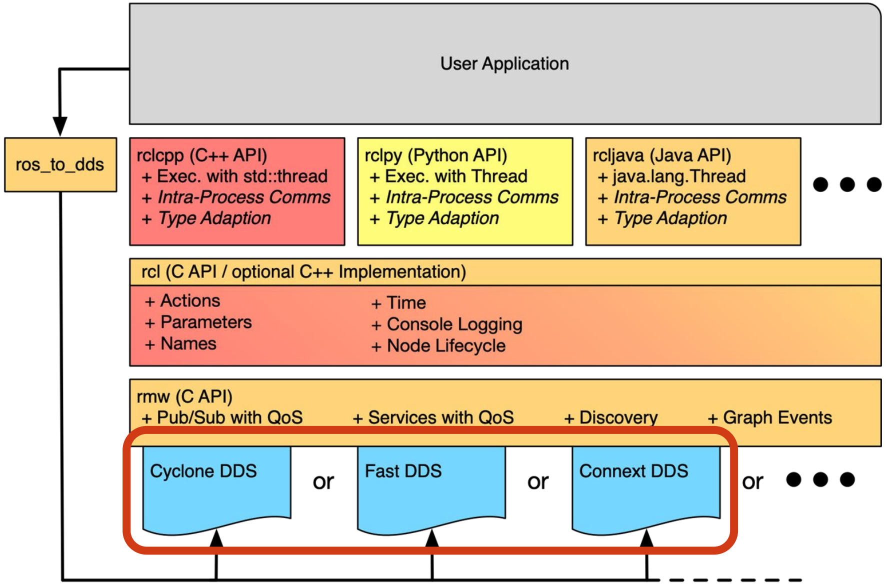
* *Intra-Process Comms* and *Type Adaption* could be implemented in the client library, but may not currently exist.

with a runtime-swappable middleware layer!

ROS™

https://design.ros2.org

But all tier-1 middlewares are DDS-based



* *Intra-Process Comms* and *Type Adaption* could be implemented in the client library, but may not currently exist.

credible

flexible

reliable

For good reasons…

https://design.ros2.org/articles/ros_on_dds.html

https://design.ros2.org

Technical Credibility

DDS has an extensive list of varied installations which are typically mission critical. DDS has been used in:

- battleships
- large utility installations like dams
- financial systems
- space systems
- flight systems
- train switchboard systems

and many other equally important and varied scenarios. These successful use cases lend credibility to DDS's design being both reliable and flexible.

Not only has DDS met the needs of these use cases, but after talking with users of DDS (in this case government and NASA employees who are also users of ROS), they have all praised its reliability and flexibility. Those same users will note that the flexibility of DDS comes at the cost of complexity. The complexity of the API and configuration of DDS is something that ROS would need to address.

The DDS wire specification (DDSI-RTPS) is extremely flexible, allowing it to be used for reliable, high level systems integration as well as real-time applications on embedded devices. Several of the DDS vendors have special implementations of DDS for embedded systems which boast specs related to library size and memory footprint on the scale of tens or hundreds of kilobytes. Since DDS is implemented, by default, on UDP, it does not depend on a reliable transport or hardware for communication. This means that DDS has to reinvent the reliability wheel (basically TCP plus or minus some features), but in exchange DDS gains portability and control over the behavior. Control over several parameters of reliability, what DDS calls Quality of Service (QoS), gives maximum flexibility in controlling the behavior of communication. For example, if you are concerned about latency, like for soft real-time, you can basically tune DDS to be just a UDP blaster. In another scenario you might need something that behaves like TCP, but needs to be more tolerant to long dropouts, and with DDS all of these things can be controlled by changing the QoS parameters.

ros_to_

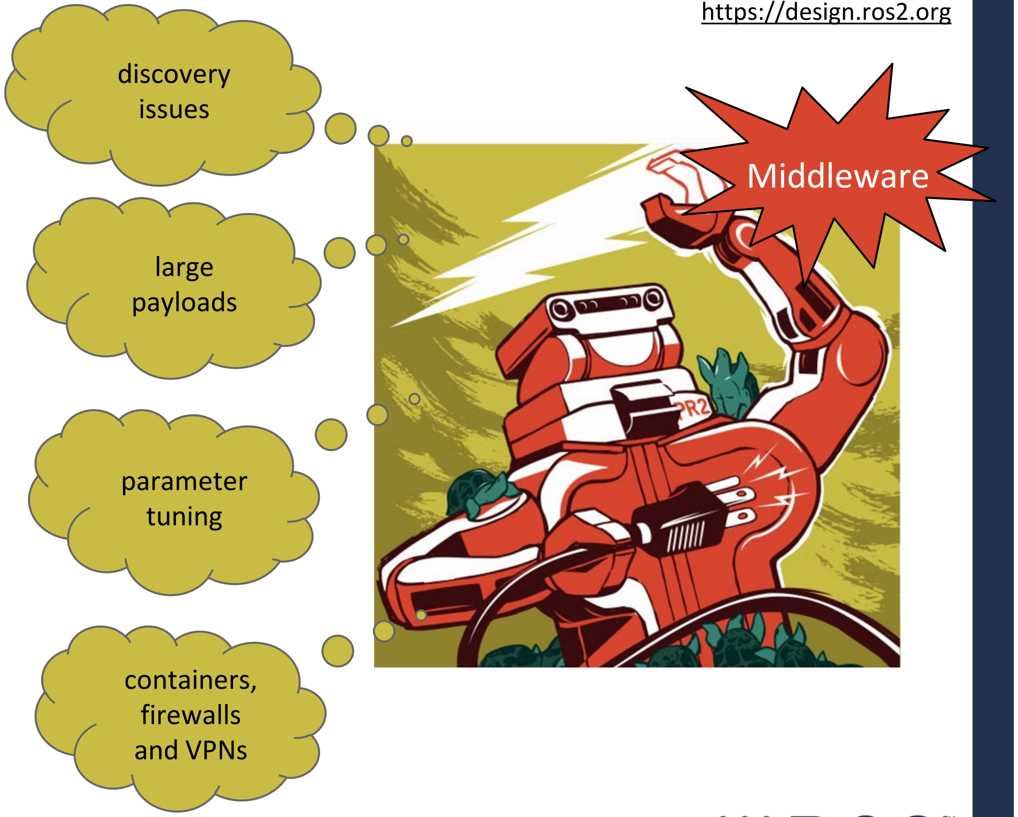
* *Intra-Process Comms* and *Type Adaption* could be implemented in the client library, but may not currently exist.

ROS™

https://design.ros2.org

But we learnt of some challenges

that lead to poor out-of-the box experience with ROS 2.

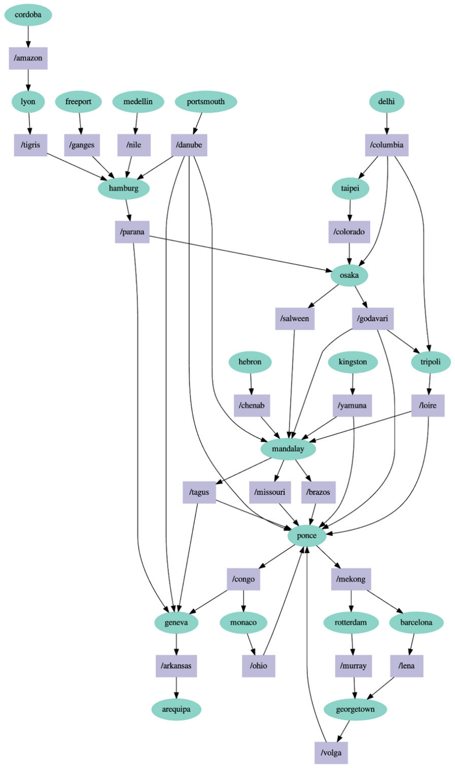


https://discourse.ros.org/t/ros-2-alternative-middleware-report/33771

ROS™

Challenges are inherent to DDS

- **Peer-to-peer only**
  All participants discovers details from all other participants.
- **Heavy discovery protocol**
  $O(N^2)$ discovery messages.
  N is number of Participants, Topics, Readers, Writers.
- **Over UDP (multicast and unicast)**
  Fragile to message losses, especially with large payloads.
  Retransmissions ⇒ more traffic ⇒ more losses
  No NAT/Firewall traversal
  No optimization nor hardware acceleration such as for TCP.



ROS™

Some must-haves for the new middleware

Pub/Sub

For data-centric distributed systems

Transport

Ability to send multi-megabyte payloads reliably.

Low latency

Especially for sending small messages at a very high frequency.

Resilient

Robust against network disconnections.

Discovery

Built-in discovery and the ability to restart discovery without restarting all the nodes.

Security

Access control, authentication and encryption.

https://discourse.ros.org/t/ros-2-alternative-middleware-report/33771

ROS™

Alternative middlewares considered

OpenCyphal

TCPROS

eCAL

ØMQ

MQTT

LCM

OPC UA

kafka

NNg

Zenoh

ROS™

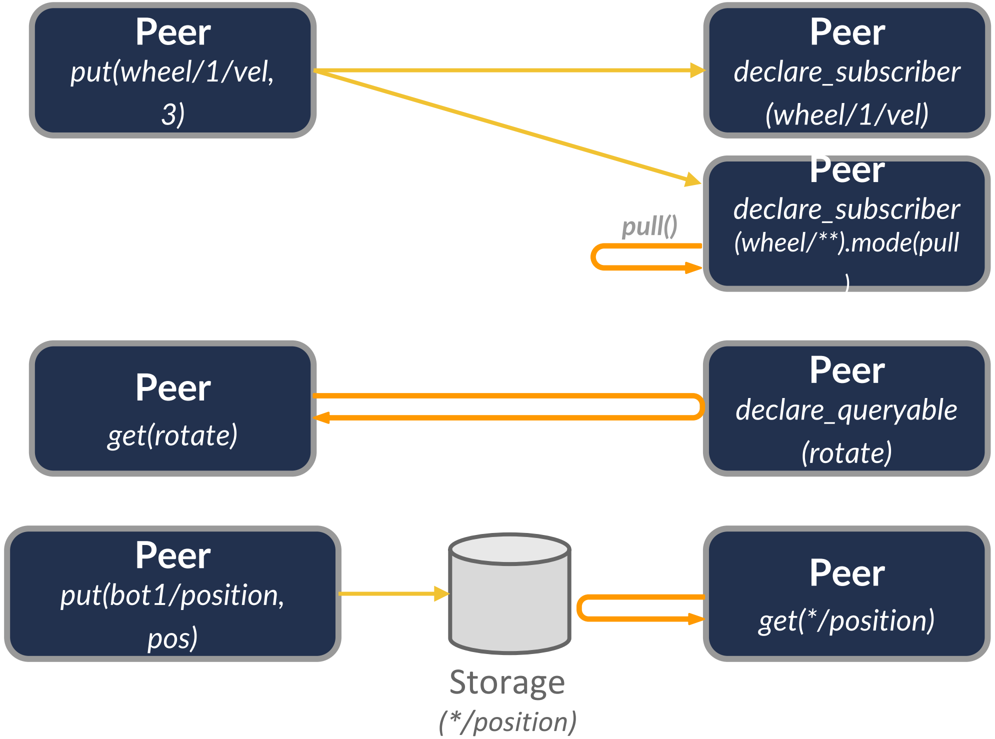
Requirements		Middleware Options										
Item	Level	Cyphal	DDS	eCal	Kafka	LCM	MQTT	NNG	OPC-UA	TCPROS	Zenoh	ZeroMQ
Configure which interface to use	Must	Yes (IP address)	Yes	Yes (via standard Linux configuration)	Yes	Yes	Yes, implementati dependan See mosqu			Indirectly ...h the	Yes, configuration variable	Yes (both tcp and udp during zmq_bind())
Ability to send multi-megabyte messages	Must	Y	Borderline; in UDP mode									Should be since TCP, see https://github.com/jeffbass/imagezmq#why-use-imagezmq
Ability to send fast small messages	Must	Y									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
		Linux		Linux	Java, so in	Linux		Linux		Linux, Windows	Linux	Linux

https://discourse.ros.org/t/ros-2-alternative-middleware-report/33771

Zenoh

Zero overhead network protocol

Pub/Sub/Query protocol



Unifies data in motion, data at rest and computations from embedded microcontrollers up to the data center.

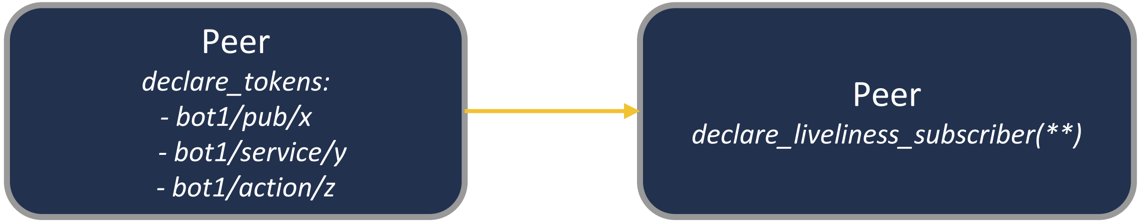
ROS™

Zenoh Extensions

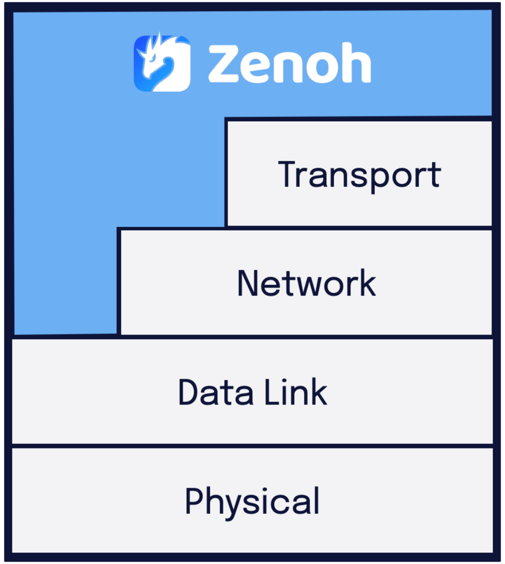
Publication Cache

Querying Subscriber

Liveliness Tokens:
for discovery and
supervision of anything



Runs everywhere



Native libraries and **API bindings** for many programming languages.

Over various **network technologies**: from **transport layer** to **data link**.
With support of TCP, UDP, TLS, QUIC, serial…

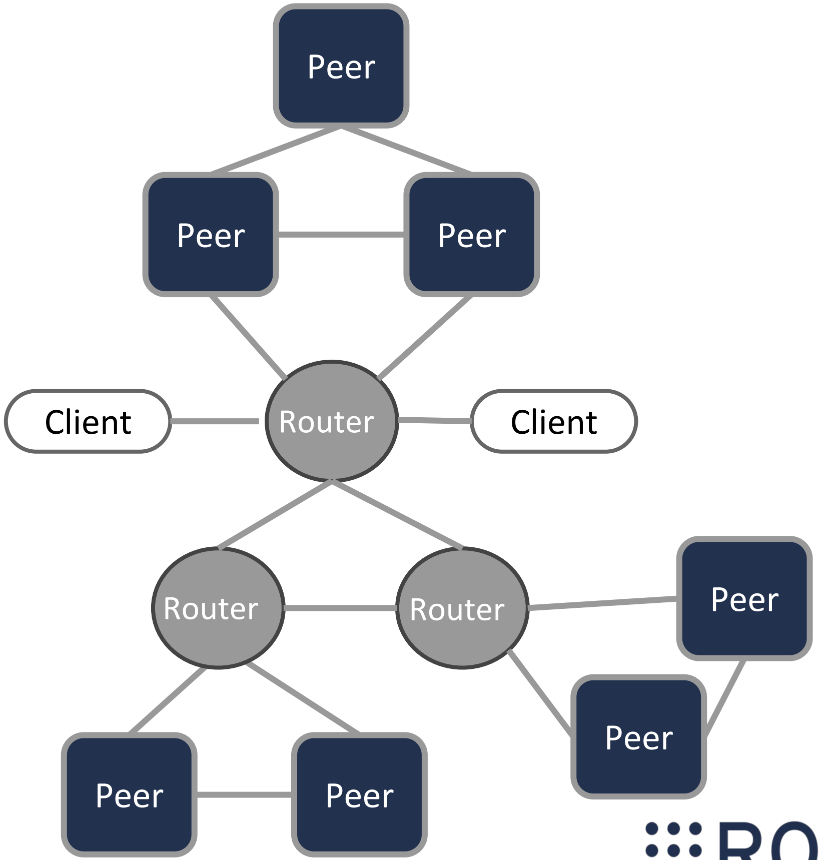
On **embedded** and **constrained devices**

Any topology

Peer-to-peer
Clique and mesh topologies

Brokered
Clients communicate
through a router or a peer

Routed
Routers forward data and requests
between peers and clients



rmw_zenoh

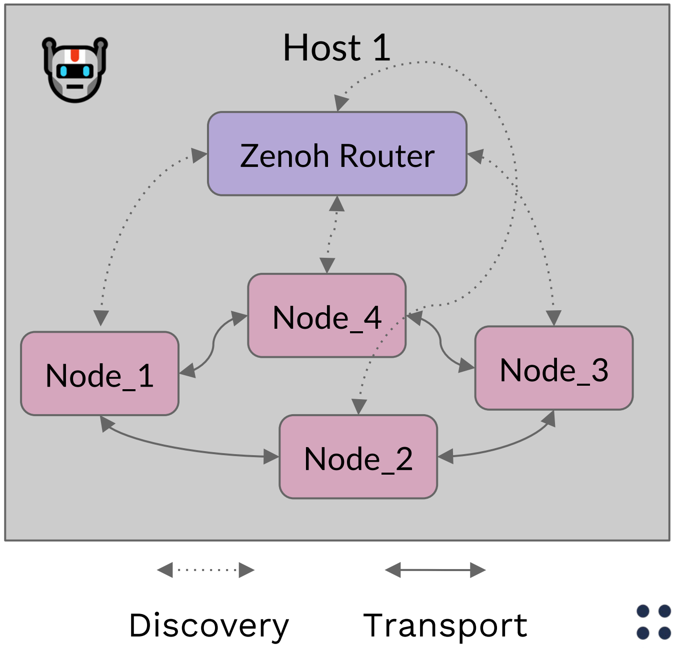
An RMW implementation based on Zenoh

https://github.com/ros2/rmw_zenoh

rmw_zenoh

- RMW implementation written with zenoh-c(pp) binding.
- TCP for discovery and transport.
- No QoS mismatches.
- Default topology
  - Discovery is brokered by the Zenoh router
  - Data transmission is P2P.
  - Discovery range is localhost only

export RMW_IMPLEMENTATION=rmw_zenoh_cpp

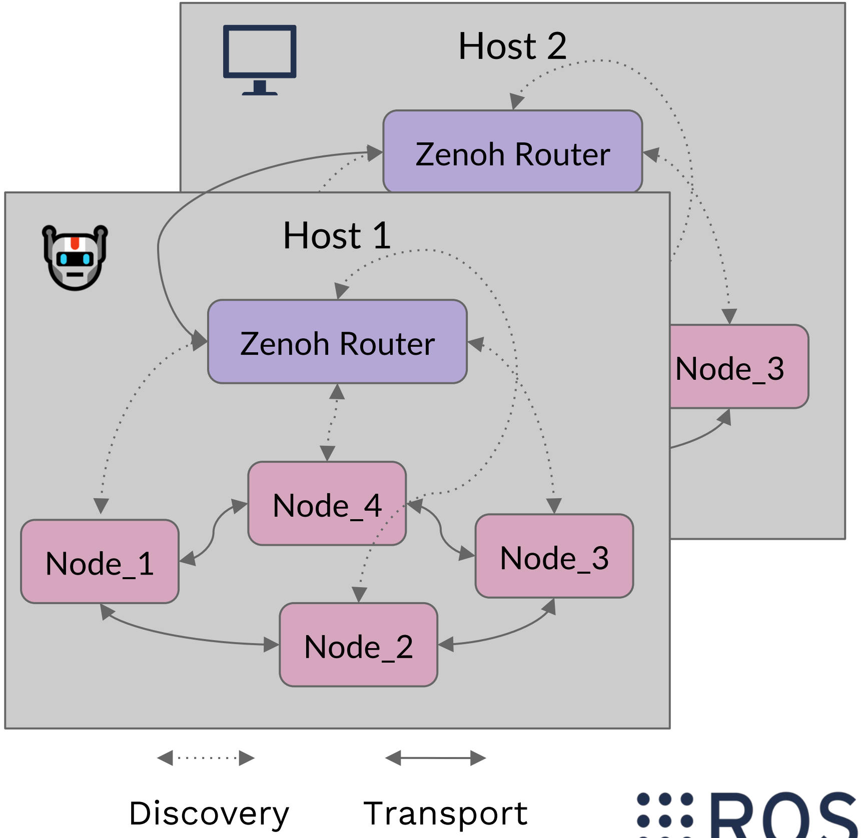


ROS™

https://github.com/ros2/rmw_zenoh

rmw_zenoh

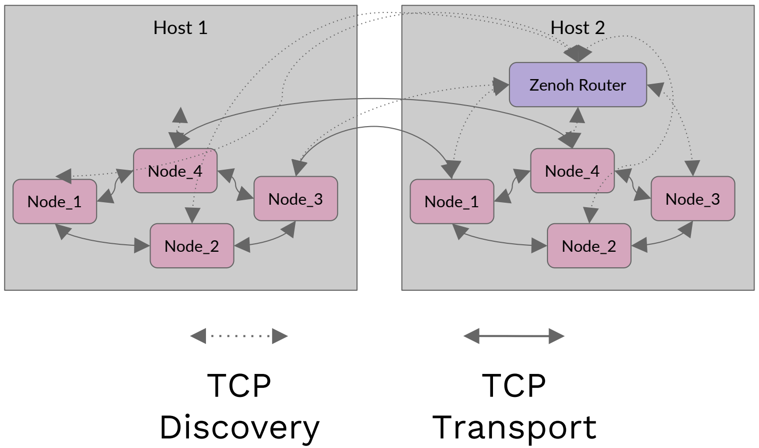
- Default multi-host topology
  - Different hosts connect through routers
    - Brokered data transfer
    - Opportunity to downsample

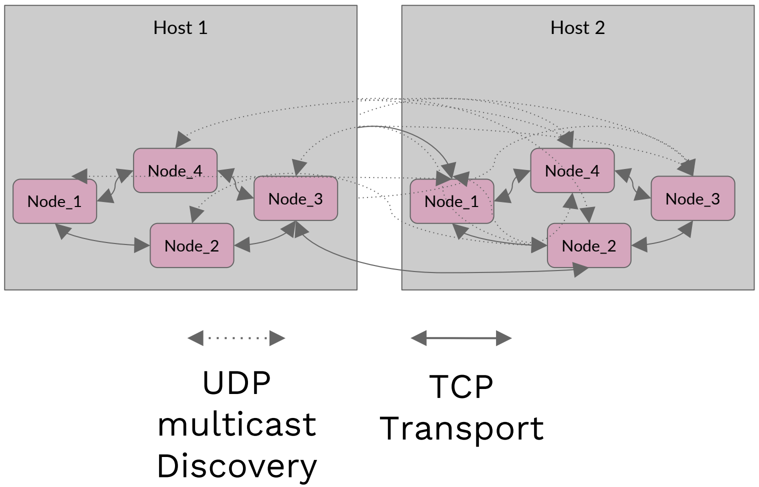


ROS™

https://github.com/ros2/rmw_zenoh

Other configurable topologies





https://github.com/ros2/rmw_zenoh

Let’s talk about the router

- **Is it similar ROS 1’s roscore?**

  Yes, but it does a lot more.

- **What if the router crashes?**
  No impact on running Nodes.
  ROS Daemon still present for graph cache.
  Just restart the router! **No need to re-launch your Nodes.**

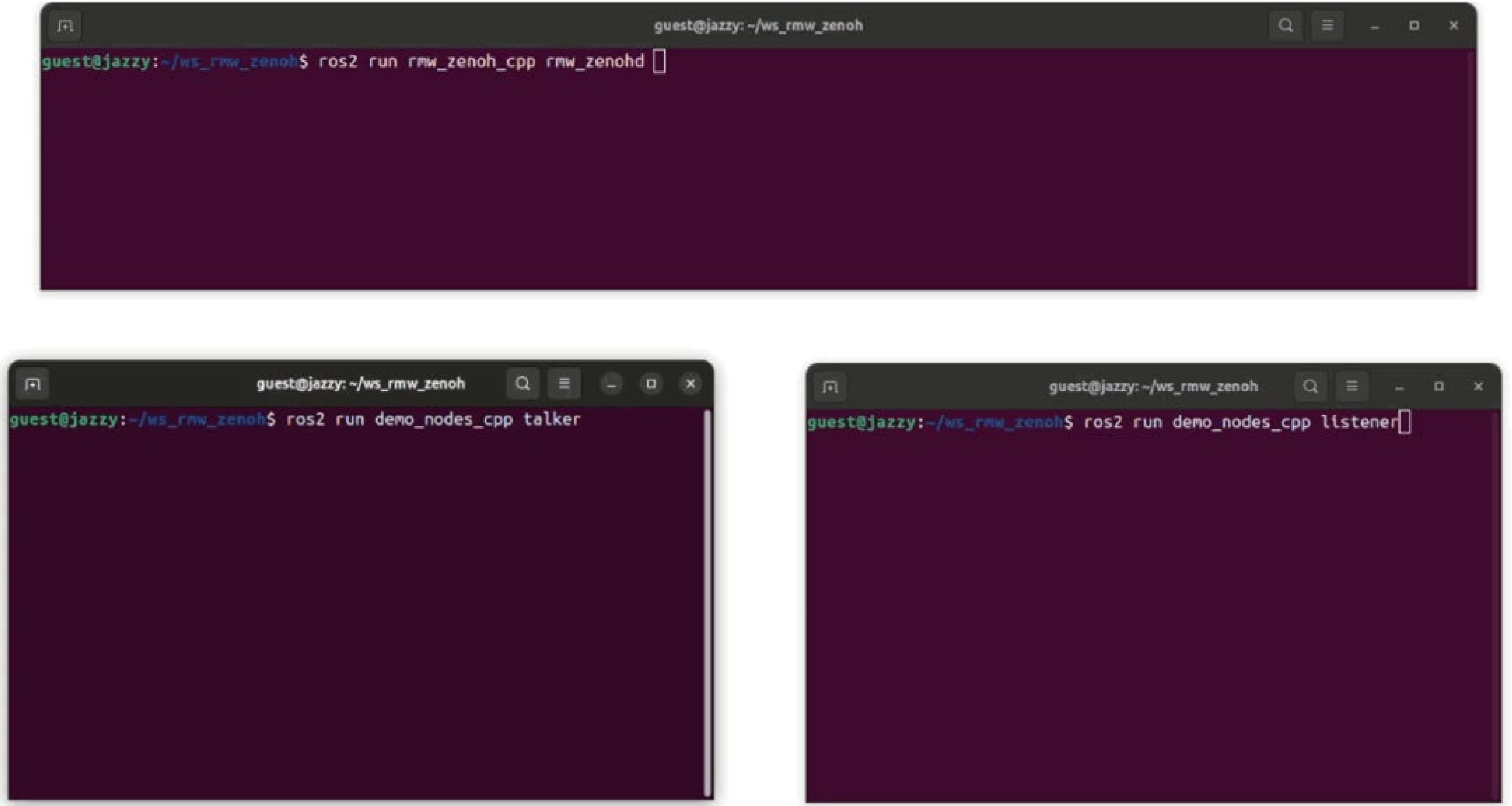
- **Is the router mandatory?**
  No. You can configure Zenoh for UDP multicast discovery.

1.

Router for discovery,
but peer-to-peer
communications

ROS™

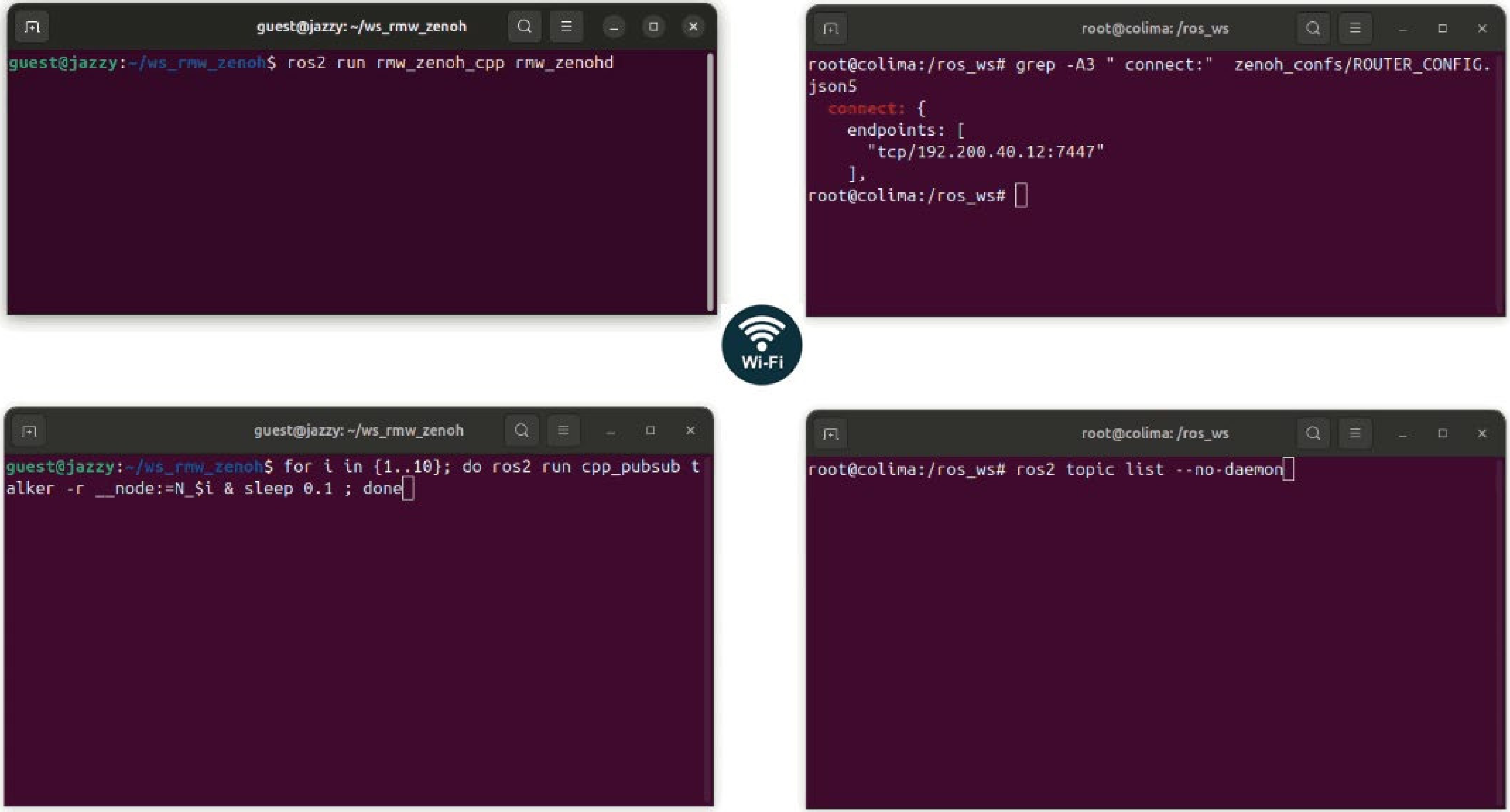
If router crashes, peer-to-peer communications remain



2.

Discovery is robust

10 Nodes, 1000 Topics discovered in less than 1s over WiFi



3.

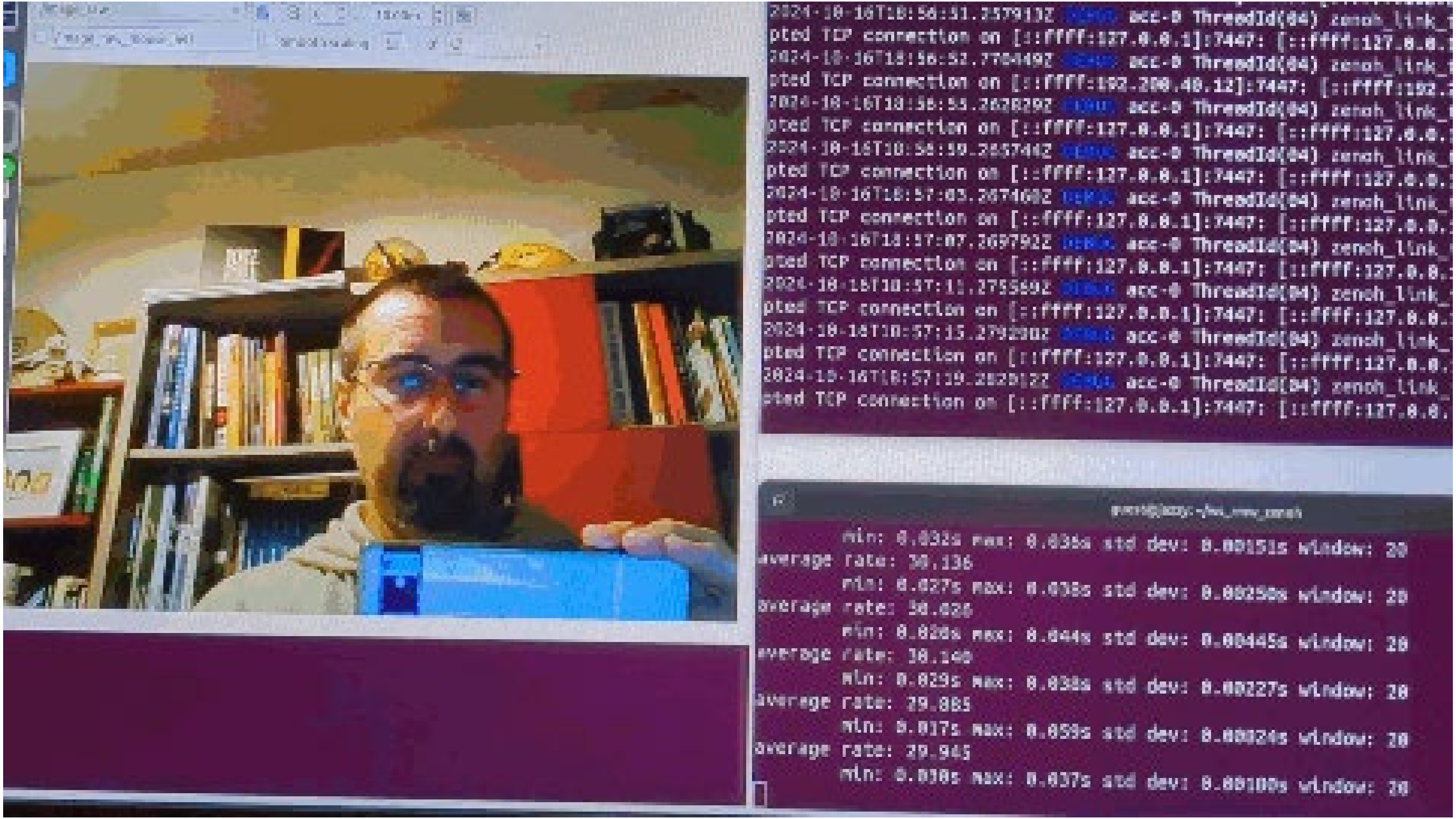
Transport reliably over many network hops


nav2 and drivers running on RPi 4!
RViz
TB3 in US
Router
RViz in FR
Router
Router

4.

Downsample when needed

ROS™


acc-0 ThreadId(04) zenoh_link_
pted TCP connection on [::ffff:127.0.0.1]:7447: [::ffff:127.0.0.
pted TCP connection on [::ffff:192.200.40.12]:7447: [::ffff:192.
pted TCP connection on [::ffff:127.0.0.1]:7447: [::ffff:127.0.0.
min: 0.032s max: 0.036s std dev: 0.00151s window: 20
average rate: 30.136
min: 0.027s max: 0.038s std dev: 0.00250s window: 20
average rate: 30.026
min: 0.026s max: 0.044s std dev: 0.00445s window: 20
average rate: 30.140
min: 0.029s max: 0.038s std dev: 0.00227s window: 20
average rate: 29.885
min: 0.017s max: 0.059s std dev: 0.00824s window: 20
average rate: 29.945
min: 0.030s max: 0.037s std dev: 0.00100s window: 20

Known limitations

- **`rclcpp::shutdown()`** must explicitly be called before program termination.
- Router must be manually started (for now).
- Liveliness and deadline QoS events not supported.

Road to Tier-1 status

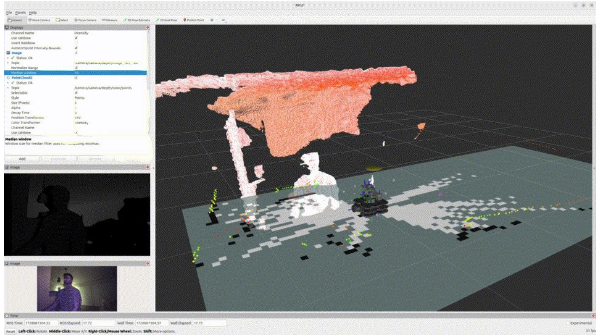
https://github.com/ros2/rmw_zenoh/issues/265

- Targeted for Kilted Kaiju release.
- Make all system tests pass.
- Windows support.
- SROS2 integration.

ROS™

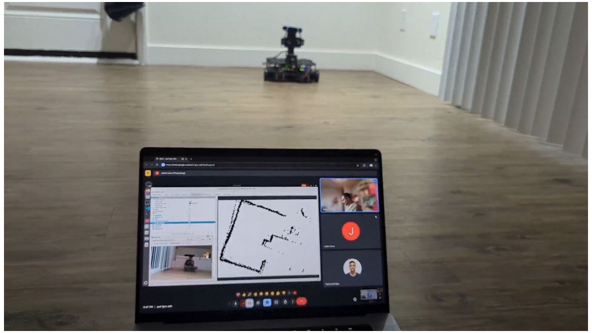
We’re testing extensively

ROS™

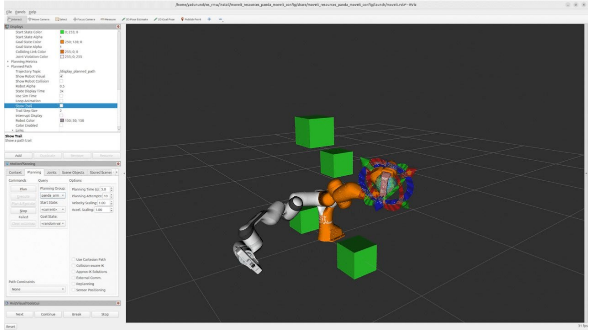


nav2, camera driver running on RPi 4
RViz opened on laptop over wifi

Open-RMF with 34 nodes 82 topics and 251 services with an additional RViz window open on a laptop over wifi.



TB3 in US, RViz in Paris, via a router in Cloud



Moveit2

ROS™

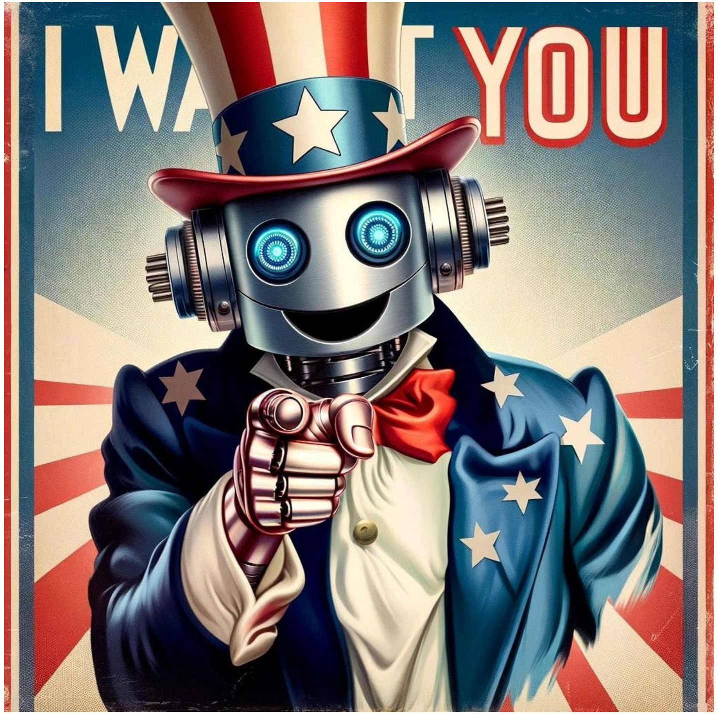
https://github.com/ros2/rmw_zenoh

Try rmw_zenoh

Run your existing applications with it!

Benchmark it!

And tell us…



ROS™

Questions?

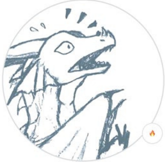
special thanks to

Chris Lalancette
clalancette

Morgan Quigley
codebot

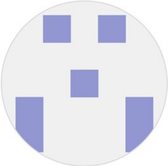
Franco Cipollone
francocipollone

Geoffrey Biggs
gbiggs



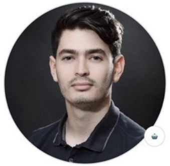
methylDragon
methylDragon

Esteve Fernandez
esteve

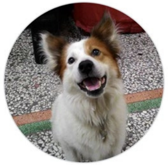


Andreas Bihlmaier
andreasBihlmaier

Alejandro Hernández Cordero



Steven Palma
imstevenpmwork



Yuyuan Yuan
YuanYuYuan

ChenYing Kuo (CY)
evshary

Luca Cominardi
Mallets

And all contributors to eclipse-zenoh :