Zenoh: Unifying Communication, Storage and Computation from the Cloud to the Microcontroller

Angelo Corsaro, PhD, Luca Cominardi, PhD, Olivier Hecart, Gabriele Baldoni, Julien Enoch
Pierre Avital, PhD, Julien Loudet, PhD, Carlos Guimares, PhD, Michael Ilyin, Dmitrii Bannov
ZettaScale Technology, France
{name.surname}@zettascale.tech

***Abstract*—An increasing number of systems span from the data-center down to the micro-controller and need to smoothly operate across this continuum composed by extremely heterogeneous network technologies and computing platforms. Building these systems is quite challenging due to limitations of existing technological stacks. This paper introduces Zenoh a Pub/Sub-/Query protocol that unifies data at rest, data in motion and computations. Zenoh has been designed ground-up to address the needs of the cloud to micro-controller continuum. It has a minimal wire overhead of 5 bytes, it runs and perform on constrained as well as on high end networks and hardware.**

I. Introduction

An increasing number of systems span from the data-center down to the microcontroller and need to smoothly operate across this continuum composed by extremely hetherogeneous network tehcnologies and computing platforms. Building these systems today is quite challenging due to limitations of the existing technological stacks that are explained below.

A. Connectivity Islands

Existing protocols were designed to work on a very specific use case and in a way address a "connectivity island." As an example, the Data Distribution Service (DDS) [DDS(2017)] was designed to provide a pub/sub protocol that works best for applications running on resourceful hardware connected by multicast-enabled (UDP/IP) wired Local Area Network (LAN). Another assumption in DDS' design is that peer-to-peer communication is quintessential and most of the applications consume data from every other application.

At the opposite side of the spectrum we have MQTT [OASIS(2014)], which was designed to support pub/sub via a client to broker architecture over TCP/IP networks. What is interesting is that both DDS and MQTT provide pub/sub. Yet, their implementations force onto the user very specific communication topologies that are completely orthogonal to the concept of pub/sub. This introduces architectural inflexibility and scalability issues. As an example, DDS is notoriously hard to work with and scale on a Wide Area Network (WAN) as a consequence of its (flat) peer-to-peer only model and its reliance on multicast IP. MQTT, on the other hand, makes communicating across a WAN easy, as far as one can accept to have a hub-and-spoke architecture and a topology not ideal for several edge applications.

But things are even worse. While Message Queueing Telemetry Transport (MQTT) is often referred-to as a lightweight protocol, it relies on TCP/IP and this is not always available nor desirable for constrained hardware and constrained networks. Thus, other protocols are often used to deal with constrained hardware, such as Constrained Application Protocol (CoAP) [Shelby et al.(2014)Shelby, Hartke, and Bormann].

At this point, the legitimate question to ask is: how can we deal with systems that include constrained hardware and networks, require high-performance peer-to-peer on the edge, and need to efficiently scale over the Internet?

Thus far, the solution has been to use different protocols on different segments of the system and integrate them together hoping to have some meaningful end-to-end semantics. This is tedious, error-prone and inefficient. A consequence of the inability of established protocols to deal with the cloud-to-device continuum – they weren't simply designed for it.

B. Data in Motion and Data at Rest

Pub/Sub protocols have emerged as the technology of choice to deal with data in motion, while databases as the technology of choice to deal with data at rest. These two technology ecosystems are intimately related. Data in movement needs, at some point, to be stored, thus becoming at rest, and eventually retrieved. Yet, from a programmer perspective there is no unified API to deal with both of them. Additionally, while pub/sub features location transparency, databases are location centric. In other words, when expressing a subscription in a pub/sub system one doesn't need to know the location of the publisher(s), yet, when submitting a query it is required to know the location of the database. As a consequence, either one has to keep all the data in a central location – like the solutions provided by cloud storage – or has to deal with the complexity of tracking data's location. This is a major challenge for edge applications. As for these applications it is key to have data stored in a distributed manner, to avoid the cost, including energetic cost, of shipping it to the cloud and to reduce the latency to retrieve it.

C. Computations

While distributed applications can be modeled as data flows, with computations being triggered only by data, it is rare that a distributed application is entirely based on this paradigm. Often it is convenient to have services and be able to trigger, and invoke their execution. This in turn requires reliance on

2023 26th Euromicro Conference on Digital System Design (DSD) | 979-8-3503-4419-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/DSD60849.2023.00065

2771-2508/23/$31.00 ©2023 IEEE
DOI 10.1109/DSD60849.2023.00065

yet another technology ecosystem that supports request/reply. Which means that in turn our developer needs to learn yet another set of abstractions and APIs. Additionally, existing request/reply frameworks are host-centric, making it hard to deal with load-balancing, and fault-tolerance.

Zenoh [Corsaro(2018)] was born from the ambition to address these problems in a structured manner. We did a systematic review of all the available protocols, including emerging Named Data Networking (NDN) [Zhang(2014)], capitalized on the 20+ years of experience of our team in working in distributed systems, ranging from embedded systems to Pan-European Air Traffic Control and management. After a few years of R&D, our team identified the minimal set of orthogonal primitives that would allow us to deal with data in motion, data at rest and computations – from the data-center to the micro-controller. The result of this effort was Zenoh.

The reminder of this paper is organised as follows, Section II introduces the Zenoh protocol and explains its key features. Section III provides a compartive analysis of Zenoh's wire efficiency and performance against MQTT and DDS. Finally Section IV discusses some of the innovations we are currently working along with conluding remarks.

II. Zenoh

Zenoh is a Pub/Sub/Query protocol that provides a set of unified abstractions to deal with data in motion, data at rest and computations at Internet Scale. Zenoh runs efficiently on server-grade hardware and networks as well as on micro-controller and constrained networks. Finally, Zenoh supports peer-to-peer, routed and brokered communication, thus allowing for an optimal communication model at each stage of the system. In the reminder of this section we will introduce the key Zenoh's concepts.

A. Positioning the Protocol

As we are talking about protocols, the first thing we should do is to position it with respect to the ISO/OSI model. Figure 1 shows that Zenoh can run above a Data Link, the Network or the Transport Layer. Which as a consequence, indicates that the minimal requirement for Zenoh is to have available a best effort data-link. As of today, Zenoh supports Serial Links, Bluetooth, LORA, Unix Sockets, TCP/IP, UDP/IP, QUIC, WebSockets, CANbus, and OpenThreadX.

The reason for having a protocol that can run over the Data Link is that in embedded systems the IP protocol stack is not always available or not necessarily desirable for wire-overhead reasons.

B. Protocol Abstractions

1) Resources, Key Expression and Selectors: Zenoh operates over resources. A resource is a $(key, value)$ tuple, where the key is an array of arrays of characters. When representing keys we usually use the "/" as a separator. Thus, `home/kitchen/sensor/C2O2` is a Zenoh key.

A set of keys can be expressed by means of a key selector, which may include $*$ or $**$ which expand respectively to an arbitrary array of characters not including the separator, and an array of arrays of characters. For instance `home/kitchen/sensor/*` would represent the set of keys including all the sensors in my kitchen, while `home/*/sensor/C2O2` would represent the set of keys representing all the C_2O_2 sensors in my house.

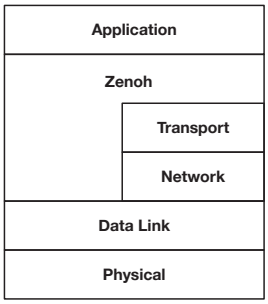

Figure 1. Zenoh protocol stack positioning.

Zenoh allows to select a set of resources by using a selector. The syntax supported Zenoh's the selector is `keyexpr?arg1=val1&arg2=value` – where `keyexpr` is a key expression as defined above. Some $args$, such as those for indicating filters, projections and time intervals are built-in, application-specific semantics can be added by defining additional arguments. As an example the selector `home/*/sensor/temperature?_filter="temp>25"&_project="hum"`, among all the temperature sensor in my house it would select those whose value is greater than 25 and project their humidity.

2) Publisher, Subscriber and Queryable: The Zenoh protocol defines three different kinds of network entities, publisher, subscribers and queryables. A publisher should be thought as the source for resources matching key expression. As an example, a publisher could be defined for a key, such as `home/kitchen/sensor/C2O2`, or for a set of keys, such as, `home/kitchen/sensor/*` or `home/kitchen/**`.

Symmetrically, a subscriber should be thought as a sink for resources matching key expression. As an example, a subscriber could be defined for a key, such as `home/kitchen/sensor/C2O2`, or for a set of keys, such as, `home/**/sensor/*`.

A queryable should be through of as a well for resources whose key match a key expression. As such a queryable for `home/kitchen/**` essentially promises that if queried for keys that match this key expression it will have something to say.

Finally, it is worth mentioning that at a protocol level a the declaration of publisher is optional and is just seen as an optimization for recurring publications over a set of keys.

3) Primitives: Zenoh has a very constrained number of orthogonal primitives, these are:

- **Declarations**. These primitives, namely, `declare_resource`, `declare_publisher`, `declare_subscriber`, and `declare_queryable` allow to declare a resource, a publisher a subscriber and a queryable respectively. A declaration is either used to optimize certain aspects of the protocol, such as automatically mapping keys to small integers, or to inform

```
import zenoh
# Opens a zenoh session
z = zenoh.open()
key = 'demo/sensor/temp'
# Zenoh supports natively various data types
session.put(key, 25)
```

Figure 2. Producing data with Zenoh

```
import zenoh
# Opens a zenoh session
z = zenoh.open()
def listener(sample: Sample):
    key = sample.key_expr
    value = sample.payload.decode()
    print(f">> Received: ('{key}', '{value}')")

# declare a subscriber
sub = session.declare_subscriber(key, listener)
```

Figure 3. Declaring a Zenoh subscription.

```
import zenoh
# Opens a zenoh session
z = zenoh.open()
key_expr = 'example/sensor/*'

# issue a query
replies = session.get(key_expr, zenoh.Queue())
for reply in replies.receiver:
    try:
        print(">> Received ('{}': '{}')"
              .format(reply.ok.key_expr,
              reply.ok.payload.decode()))
    except:
        print(">> Received (ERROR: '{}')"
              .format(reply.err.payload.decode("utf-8")))
```

Figure 4. Issuing a Zenoh distributed query.

the rest of the Zenoh network that a specific endpoint is available. That said, differently from protocols, such as DDS, in which the dynamic discovery information provides extremely precise information on who and what is available on the system, inducing as a consequence severe scalability problems, zenoh uses sets and set-theory operation to generalize the information distributed across the network. As such, and as independently measured in several instances [OSR(2022)], it is not hard for a Zenoh system to have a fraction of the discovery traffic generated by an equivalent DDS system.

- **Producing Data**. The `put` operation is used to produce a $(key, value)$. This operation provides options that allow to specify the congestion control applied to it, the associated priority and a few other non-functional properties.
- **Deleting Data**. Zenoh provides a `delete` operation that makes it possible to indicate the desire that a resource shall be deleted.
- **Query**. Zenoh provides a `get` operation that allows to issue a query. This query will be served by a set of queryable that cover, in a set-theoretical sense, the key expression portion of the query. Additionally, among all the set of sets that cover the query, Zenoh will select the one that is closest in routing terms. Zenoh provides options to control if only one of such set will be triggered or if all the matching queryable will. It also allows to control wether a partial cover is acceptable or not. The `get` operation also allows to control how data will be consolidated on the way back, and if consolidation is required at all. Finally a query can have a body attached.

C. Zenoh's Code Example

Figure 2, 3 and 4 show how Zenoh's resources can be published, subscribed and queries respectively. These code examples show the simplicity and orthogonality of the API. It is also worth noticing that in Zenoh, as previously mentioned, a publisher is not required to publish data. On the other hand it can be used as an optimisation when producing the same resource, or set of resources recurrently.

D. Universality of Zenoh's Primitives

In distributed systems usually we need to deal with (1) data in motion, which is often addressed by pub/sub technologies, (2) data at rest that is addressed by databases, and (3) distributed computations which usually are triggered using RPC mechanisms.

Zenoh is the first technological stack that has a set of primitives that are orthogonal and complete with respect to the abstractions required for distributed computing and thus universal.

Zenoh trivially supports pub/sub as it has first class abstractions for publishers and subscribers.

Zenoh supports data at rest since the combination of a queryable and a subscriber can be used to represent a database. Zenoh goes one step further and actually, provides off the shelf the integration with a large number of DBMS systems, which can be now leveraged as geo-distributed data stores and queried using the `get` operation. Additionally, Zenoh has a built-in, data-based independent, alignment protocol [Zen(2022)] that ensures eventual consistency in spite of disconnections and network partitions.

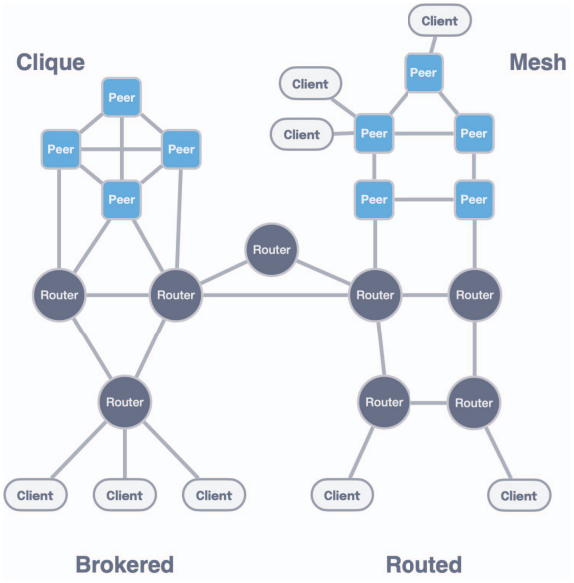

Figure 5. Zenoh supported topology.

Finally, a queryable can be used to represent a distributed computation. Additionally, this computation will be named as opposed to be bound to a specific address, you'll be able to control through the `get` options if you trigger one or several computations matching your query and as such even implement mechanism such as map/reduce.

E. Communication Topology

The Zenoh protocol does not impose any topological constraints on how application may communicate. As shown in Figure 5, Zenoh supports peer-to-peer over complete connectivity graphs as well as over arbitrary mesh. It supports routed communication and both routers as well as peers, can broker communication for clients. This generality allows to support a multitude of use cases and to scale the protocol at Internet scale. Finally, it is worth mentioning that Zenoh's routers are software-based and can run very efficiently on a raspberry pi 2 or 3.

F. Ordering and Consistency

Zenoh leverages Hybrid Logical Clock (HLC) [Kulkarni(2014)] to totally order events. While this decision may let some distributed systems purist disappointed, the reality is that sufficiently aligned clocks are a reality on modern networks. GPS and radio receivers for clock signals are extremely cheap and could be deployed with routers if necessary. But the reality is that as cellular networks are ubiquitous, decent clock synchronization is a commodity. As a consequence of total ordering it is straightforward for Zenoh to provide an *Eventually Consistent* consistency model. Stronger consistency model can be easily built by leveraging the ability to control quorums for `put` and `get`.

G. Security

Zenoh is implemented in Rust for improving security and performance. As recently reported by the NSA [NSA(2022)], 70% of security vulnerabilities are caused by memory mismanagement. These problems are ruled out using a memory safe programming language such as Rust. Additionally, in the design of Zenoh we try to limit the attack surface – as an example, our session opening protocol does not create state on the infrastructure, it let's the opening side keep track of the state by using encrypted cookies. The protocol supports pluggable authentication mechanisms along with mutual authentication and secure channels.

H. Who is using it

Zenoh's early adopters where in the telecommunication industry, as a consequence was quickly identified by ETSI [ETSI()] as a key technologies for Multi-Access Edge Computing (MEC). Furthermore, ITU recently recommended Zenoh for standardization as the protocol to be used for Intelligent Transport Systems (ITS) [ITU(2022)]. Over the past year or so, Zenoh has established itself as the protocol of choice for R2X (Robot-to-Anything) communication, it is swiftly growing in popularity in V2X (Vehicle-to-Anything) and in distributed computing as shown by recent deployments in Internet Scale analytics frameworks and Industry-4.0 soft-PLCs.

III. Performance

Efficiency and performances are important for a communication protocol that aims at addressing the cloud-to-microcontroller continuum. In this section we'll provide an analytical analysis of Zenoh's wire overhead and an empirical evaluation of Zenoh's throughput and latency when compared to DDS and MQTT.

A. Wire Efficiency

The best way to evaluate the wire efficiency of a protocol is to look at its message structure. Figure 6 and Figure 7 show the structure of the data messages for MQTT and DDS. From this it can be seen how the wire overhead added by MQTT is linear into the length of the topic name. This is a big issue since the topic name is a UTF-8 encoded string which tends to be several tens of bytes.

The minimal wire-overhead is thus 6 bytes plus the length of the topic name. To give you a concrete example, if you have an MQTT topic called `/com/acme/mysystem/devicekind/id`, this would add 32 bytes overhead to every data message. DDS on the other hand has a wire overhead of 56 bytes assuming that no inline QoS are sent.

Let's look now into Zenoh. As shown in the Figure 8, Zenoh sends frames, where a frame may contain multiple messages. This allows to pack efficiently multiple data messages – or other protocol messages – and further improve the wire efficiency. If we look at the Zenoh's data message, reported in Figure 9, its minimal wire overhead is 3 bytes. Taking into account the 2 bytes added by the frame we get to a total of

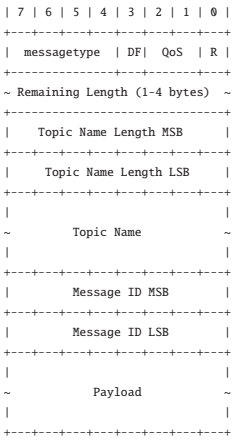

Figure 6. MQTT Data Message

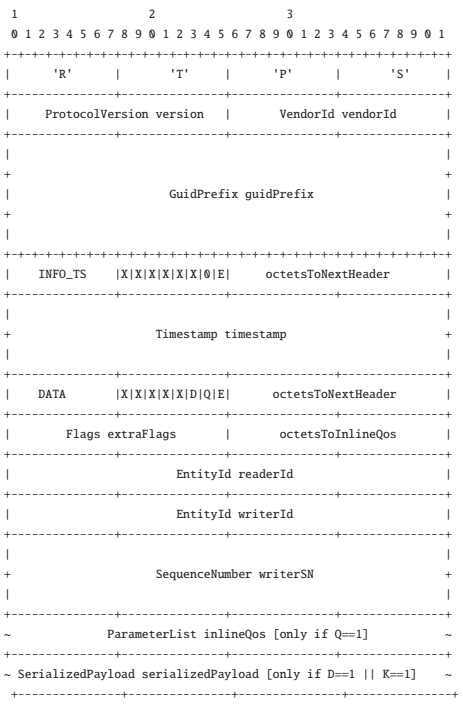

Figure 7. DDS Data Message

5 bytes overhead – when sending a single data message. If Zenoh is able to batch N messages, then the wire overhead becomes (3N + 2)/N which tends to 3 bytes fairly quickly in N.

Based on the analysis above, we can deduce that Zenoh is almost 20x more wire efficient than DDS. For MQTT, it really depends on the length of the topic name. In the given example, which represents an average topic length, Zenoh is 10x more wire efficient.

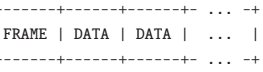

Figure 8. Zenoh Frame Message

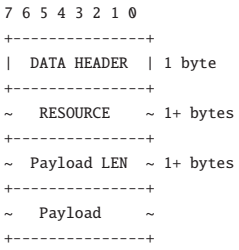

Figure 9. Zenoh Data Message

Please notice that the relative difference is extremely important in spite of the small absolute value of this overhead. If you consider applications that send data 24x7x365 like robots, cars, IoT devices, etc., then you can see that over time the differences really diverge.

B. Throughput

To evaluate the throughput we use a program publishes messages back-to-back along with a subscriber that receives the published data and calculates the message rate (msg/s). This test is ran for payload sizes ranging from 8 bytes to 512MB to evaluate the impact of the payload size on throughput. Figure 10 and Figure 11 reports the measured throughput in messages per second and bytes per second. We report both graph, because in spite of using a logaritmic scale for the y axis, it is only looking at this two different representation of the same data that one can properly appreciated the difference in performance for small and big data payloads. The dashed line on these figures represents the throughput achieved by iperf.

The results show that the best throughput is achieved by Zenoh P2P across all payload-sizes, with a peak throughput of 50 Gbps. The second best is Zenoh in a routed configuration with 34 Gbps peak throughput. Cyclone DDS follows in third

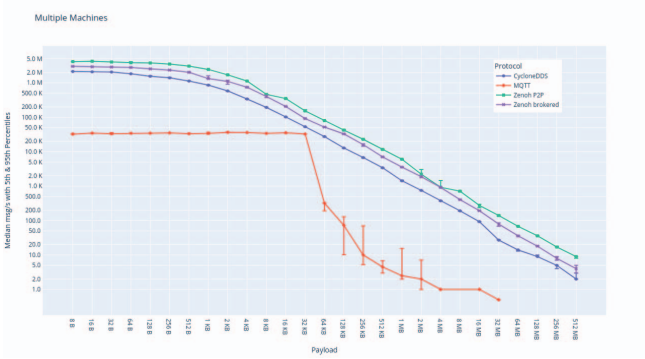

Figure 10. Throughput in messages per second.

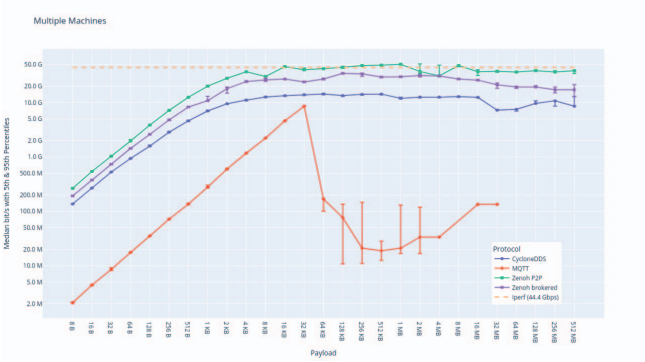

Figure 11. Throughput in bytes per second per second.

	Single-Host	Multi-Host
MQTT	27	45
Cyclone DDS	8	37
Zenoh Routed	21	41
Zenh P2P	10	16
Zenoh Pico P2P	5	13
Ping	1	7

Figure 12. Single and multi-host latency

position with a peak throughput of 14 Gbps, while MQTT peaks at 9 Gbps but then shows a rather erratic behaviour for larger payloads.

C. Latency

The latency of the various protocol was evaluated using a *ping* program that publishes the ping message, and a *pong* program immediately echos back the received message. The tested payload size is fixed to 64 bytes (aligned with ICMP echo/reply). As Zenoh can run both peer-to-peer and routed/brokered, the two configuration were tested. Additionally, when running the multiple host scenario, three hosts where used, one for the publisher, one for the subscriber and one for the router/broker.

The latency is defined as half of the median round-trip time covering the ping and pong operations. Figure 12 shows the results of the tests. The Linux ping utility was included as a baseline of the minimum latency that can be achieved.

As it can be seen in the table reported on the Figure 12 and focusing on the single host results, MQTT and Kafka have a latency of 73 μs and 27 μs, respectively. As for Zenoh, while the client mode Zenoh brokered has a latency of 21 μs, Zenoh P2P shows a latency of 10 μs. Cyclone DDS, has a latency slightly lower than Zenoh, achieving 8 μs – this number is lower than Zenoh P2P essentially for two reasons, (1) the advanced packet scheduling and batcthing performed my Zenoh and (2) the use of using UDP/IP. This explanation is validated by looking at the latency provided by Zenoh-Pico which is a Zenoh implementation that does not support arbitrary mesh topologies. When testing Zenoh-Pico's latency while running on UDP/IP we get a latency of 5 μs – which is the best over all.

The multiple-host scenario is ran over a 100 Gb Ethernet. In this test scenario Zenoh brokered has a median latency of 41 μs, while Zenoh P2P has a latency of 16 μs. MQTT has a latency of 45 μs, and Cyclone DDS of 37 μs, while Zenoh-pico, it remains the best one at 13 μs. In conclusion Zenoh provides the best latency on all cases.

Finally, the sourcecode for the tests used to evaluate these performances are available at https://github.com/ZettaScaleLabs/zenoh-perf.

IV. Concluding Remarks

In this paper we have introduced Zenoh, a novel protocol that addresses the needs of applications running from the cloud to the micro-controller continuum and which provides a set of abstractions that unify data in motion, data at rest and computations. We have demonstrated that Zenoh has the lowest wire overhead, when compared to other mainstream protocols and the highest performances both in terms of throughput s well as in terms of latency.

At the present stage we are working on extending Zenoh with a data-flow computing framework. This framework will allow to define data-flow computations spanning from the data-center down to the micro-controller and will be an extremely natural way of integrating machine learning algorithms with devices such as robots and cars. We are also working toward making Zenoh's routing completely pluggable. As of today we support two different algorithms, but going forward, advanced users will be able to define they own routing algorithm.

Finally, we'd like to thank the EU Horizon Europe research and innovation programme under grant agreement no. 101070177 (ICOS) for funding part of this research.

V. Acronyms

CoAP Constrained Application Protocol
DDS Data Distribution Service
HLC Hybrid Logical Clock
LAN Local Area Network
MEC Multi-Access Edge Computing
MQTT Message Queueing Telemetry Transport
NDN Named Data Networking
WAN Wide Area Network

References

[DDS(2017)] "The data distribution service," 2017. [Online]. Available: http://omg.org/spec/dds

[OASIS(2014)] OASIS, "Mq telemetry transport (mqtt) v3.1.1 protocol specification," OASIS, Tech. Rep., October 2014.

[Shelby et al.(2014)Shelby, Hartke, and Bormann] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7252

[Corsaro(2018)] A. Corsaro, "Zenoh: The **Ze**ro **N**etwok **O**ver-**H**ead protocol," 2018. [Online]. Available: https://zenoh.io

[Zhang(2014)] e. a. Zhang, "Named data networking," vol. 44, no. 3, 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656887

[OSR(2022)] "Zenoh for robotics," 2022. [Online]. Available: https://bit.ly/3by0Y51
[Zen(2022)] "Keeping storages aligned in zenoh," 2022. [Online]. Available: https://zenoh.io/blog/2022-11-29-zenoh-alignment/
[Kulkarni(2014)] Kulkarni, "Logical physical clocks," in *Principles of Distributed Systems: 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings 18*. Springer, 2014, pp. 17–32.
[NSA(2022)] NSA, "Software memory safety," 2022. [Online]. Available: https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF
[ETSI()] ETSI, "Etsi mec – technology ecosystem." [Online]. Available: https://mecwiki.etsi.org/index.php?title=MEC_Ecosystem
[ITU(2022)] ITU, "Automated driving safety data protocol – specification," 2022. [Online]. Available: https://www.itu.int/dms_pub/itu-t/opb/fg/T-FG-AI4AD-2022-PDF-E.pdf