Historical Background
IoT & IIoT

We were involved in building some of the very first IoT and IIoT systems.

In 2008 we were involved with the Nice’s Connected Boulevards, one of the world’s first Smart Cities.

In 2014 we part for the core team that build the Fog Platform for Barcelona.
It was Laborious

Building these systems was laborious

We had to stitch several technologies together already to make data flow end-to-end

We had to stitch a few more to deal with data storage, etc.
Chaos

The situation was extremely messy, yet it seemed that just a few of us were bothered by it.

Everyone was pushing for the technology they had adopted or were selling and ignoring the challenges...

We couldn't!
Key Limitations

Back in 2014-2015, the technologies considered as “emerging”, such as MQTT, DDS, etc, were already 10+ years old, and more importantly had not been designed to address the scale nor the heterogeneity required by IoT and IIoT.
Starting from 2015 we tried to push for a new wire protocol for the OMG DDS to address some of its shortcomings. Most notably its discovery overhead, and inability to scale over the Internet, its wire overhead, footprint, etc... But inertia prevailed...
A New Beginning

We decided to take up the challenge to design a new protocol that could work in the Cloud-to-Device continuum.

We set us-up for the additional challenge to unify data in motion and data at rest and as a consequence bring location transparency to data at rest.
Eclipse Zenoh
Unifies data in motion, data at rest and computations from embedded microcontrollers up the data centre.

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

Provides universal abstractions for cloud-to-device data-flow programming.

"Some people want it to happen, some wish it would happen, others make it happen." – Michael Jordan
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, Java, Kotlin

Supports network technologies from transport layer down-to the data link

Available on embedded and extremely constrained devices
Abstractions

Resource. A named data, in other terms a (key, value)
(e.g. /home/kitchen/sensor/temp, 21.5
/home/kitchen/sensor/hum, 0.67)

Key expression. An expression identifying a set of keys
(e.g. /home/kitchen/sensor/*
/home/**/temp

Selector. An expression identifying a set of resources
(e.g. /home/*/sensor/air?co2>12[humidity])
Abstractions

**Publisher.** A *spring* of values for a key expression
(e.g. `/home/kitchen/sensor/temp`  
`/home/kitchen/sensor/*`)

**Subscriber.** A *sink* of values for a key expression
(e.g. `/home/kitchen/sensor/temp`  
`/home/kitchen/sensor/*`)

**Queryable.** A *well* of values for a key expression
(e.g. `/home/**`
Primitives

**open/close** – Open/Close a zenoh session.

**declare_subscriber** – Declares a subscriber with a user provided callback that will be triggered when data is available.

**declare_publisher** – Declares a publisher and optimise the communication stack for repetitive publications. Notice that Zenoh does not require a publisher in order to perform publications, this is just an optimisation.

**declare_queryable** – Declares a queryable with a user provided callback that will be triggered whenever a query needs to be answered.
Primitives

**put** – puts a value for a key expression.

**pull** – Pulls data for a pull subscriber.

**get** – Issues a distributed query and returns a stream of results. The query target, coverage and consolidation depends on policies.
Scouting

Zenoh supports pluggable scouting protocols as a way to “discover” zenoh runtimes on the network as well as infrastructural nodes, such as routers.

At an API level a scout primitive is exposed to trigger scouting.

The scouting protocol leveraged by zenoh depends on the underlying network.
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
Extensible

Zenoh Plugins
Ease integration of other technologies

Lorem ipsum dolor sit amet
Performance

High throughput (4M msg/s – 40Gb/s)

Low latency (35 us)

Minimal wire overhead of 4-6 bytes

Test run on 10/07/2021 on Ubuntu 20.04
AMD Ryzen
32GB RAM
100Gbps ETH

“One of the things I love about music is live performance.” - Yo-Yo Ma
Throughput in perspective...

Zenoh is far more performant than MQTT for both small and large messages

“Harder, Better, Faster, Stronger.” - Daft Punk

Test run on 02/03/2022 on Ubuntu 20.04 AMD Ryzen 32GB RAM Localhost
Bandwidth efficiency in perspective...

Zenoh is far more efficient than DDS-XRCE and definitively more efficient than MQTT

“Even the largest avalanche is triggered by small things.” - Vernor Vinge
Performance in microcontrollers

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Host</th>
<th>Test run on 21/09/2021 on Zenoh-pico Various platforms 10Mbps ETH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pub</td>
<td>Rtr</td>
<td>Sub</td>
</tr>
</tbody>
</table>

“Even the largest avalanche is triggered by small things.” – Vernor Vinge
Protocol Highlights

Most wire/power/memory efficient protocol in the market to provide connectivity to extremely constrained targets.

Supports push and pull pub/sub along with distributed queries.

Resource keys are represented as integers on the wire, these integer are local to a session => good for wire efficiency.

Supports for peer-to-peer and routed communication.

Support for zero-copy.

Ordered reliable data delivery and fragmentation.

Minimal wire overhead for user data is 4-6 bytes.
In Summary
Final Thoughts

Zenoh was designed ground up to deal with data management from the Cloud-to-thing continuum

It unifies data at in movement and data at rest

It delivers incredible performances and can run on just about anything