MPSoC’24
2024/07/08

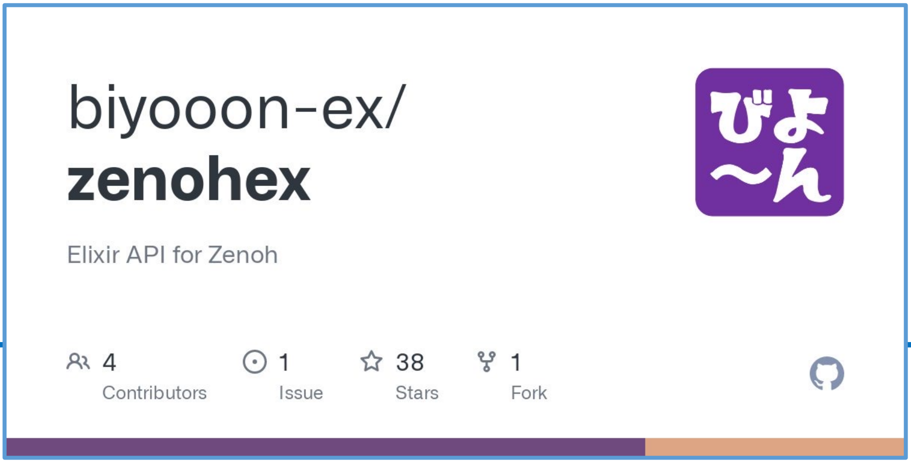
Zenohex: a Pub/Sub based Communication Library from Device to the Cloud

Hideki Takase
(The University of Tokyo)

Collaborators:
Shintaro Hosoai (Institute of Technology)
Mitsuhiro Osaki, Kazuma Nishiuchi (CityNet Inc.)
Yutaka Kikuchi (Kochi University of Technology)

Agenda

- TL;DR: Zenohex = Zenoh + Elixir
- Background & Motivation:
  - What is the issue in the wide-area distributed system
  - Publish/Subscribe based communication
- Introduction of Zenoh and Elixir
  - Basic features, ecosystem, and communication method
- Zenohex
  - How do we realize this
  - Demonstration



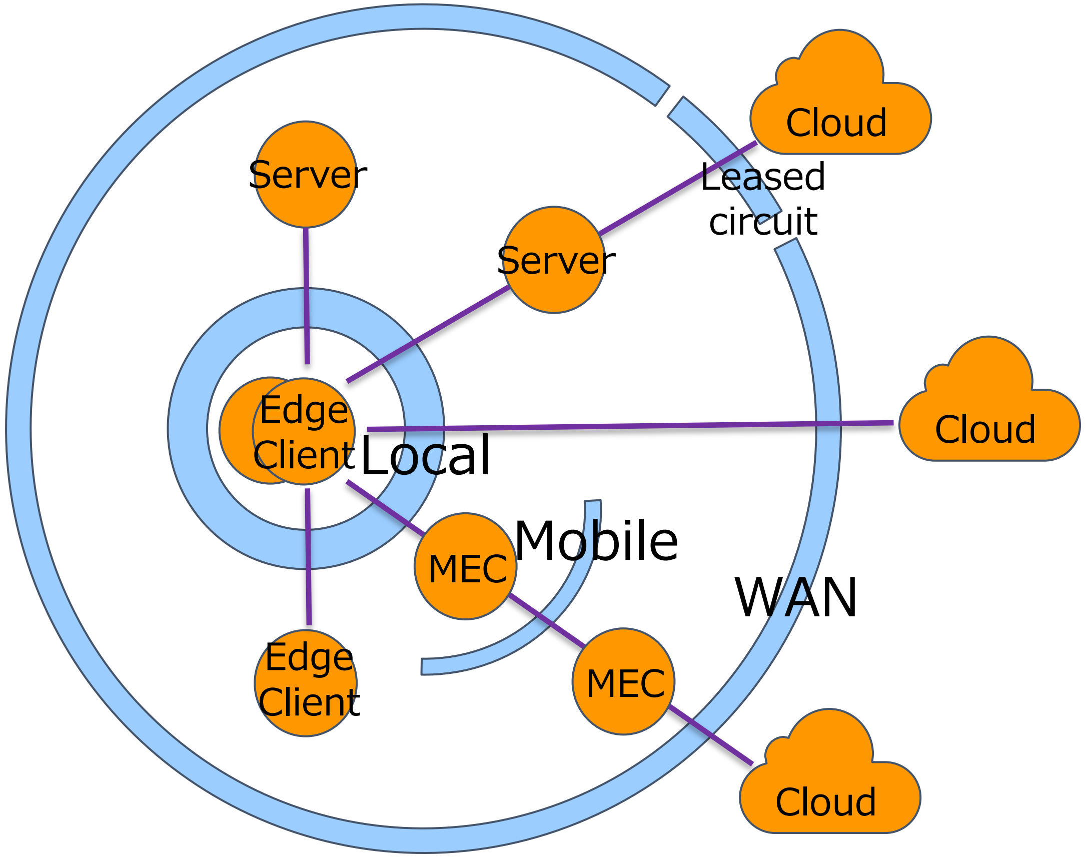
biyooon-ex/
zenohex

Elixir API for Zenoh

4 Contributors 1 Issue 38 Stars 1 Fork

What is the issue

- More and more complex system configurations



Develop

Edge-Client, Server, MEC, Cloud
- ・Spec
- ・OS
- ・Language
- ・Cloud-Service Configuration

Connected Network

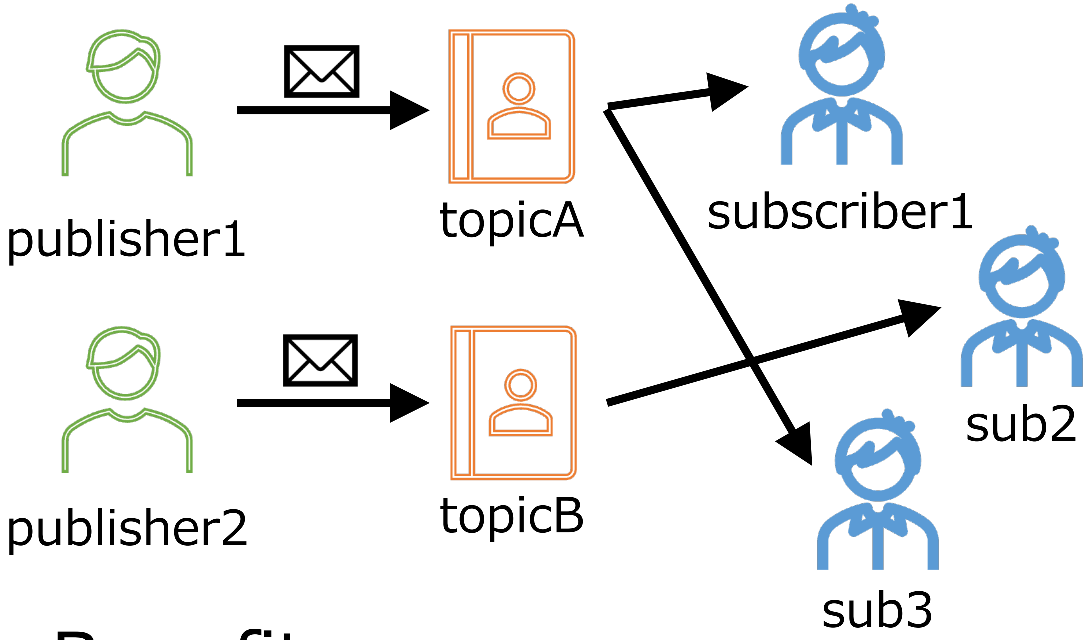
Local, Mobile, WAN, Leased circuit

Communication Protocol

Serial, http(s), MQTT, ROS, WebSocket
REST API, gRPC, FTP, SMTP

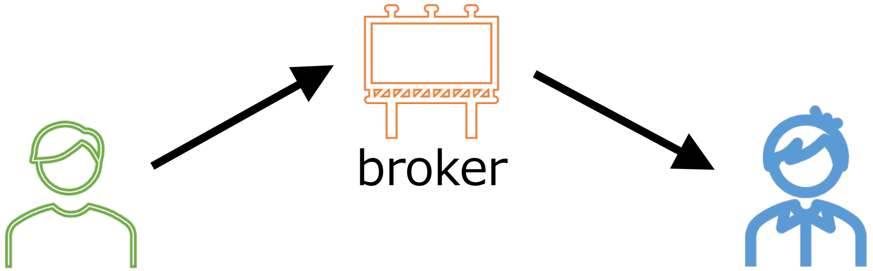
・・and more

Publish/Subscribe Messaging

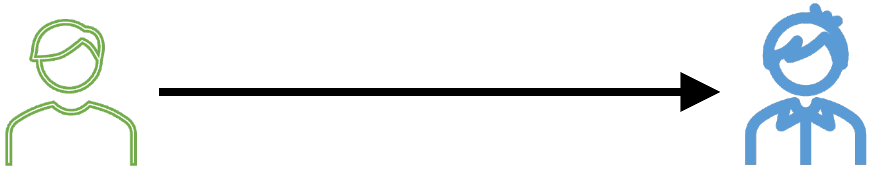


- Benefits
  - Easy to construct asynchronous and loosely coupled architecture
  - nodes can be added/deleted/ restarted independently

- Brokered (e.g., MQTT)
  - need to know where is broker



- Peer-to-Peer (e.g., DDS)
  - autonomous search for partners
  - typically limited on the same NW

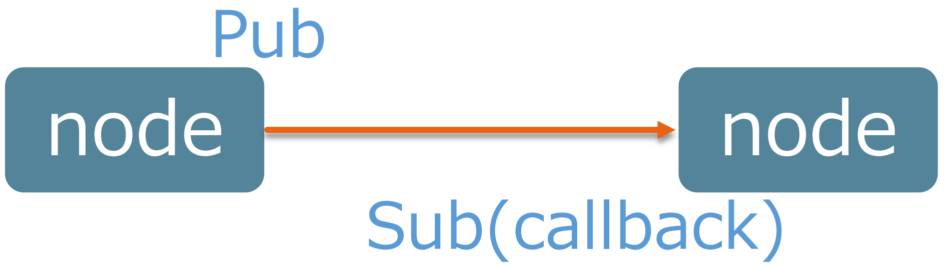


Zenoh What is??

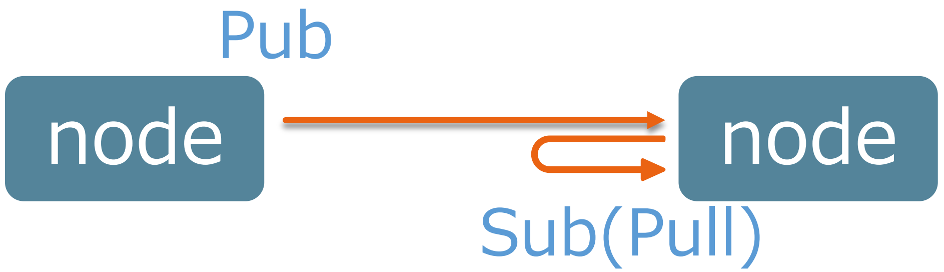
- Zero Overhead Pub/Sub, Store/Query and Compute
  - **Ze**ro **n**etwork **o**ver**h**ead protocol
  - DDS-like communication within a network and MQTT-like communication between networks
- Dev leader: [ZettaScale Technology Ltd.]()
  - GitHub: [https://github.com/eclipse-zenoh/]()
    - ✓One of the Eclipse Project
    - ✓Eclipse Public License 2.0 and/or Apache 2.0
  - Initially implemented in OCaml, and then migrated to Rust in Oct 2020

Zenoh Eloquent

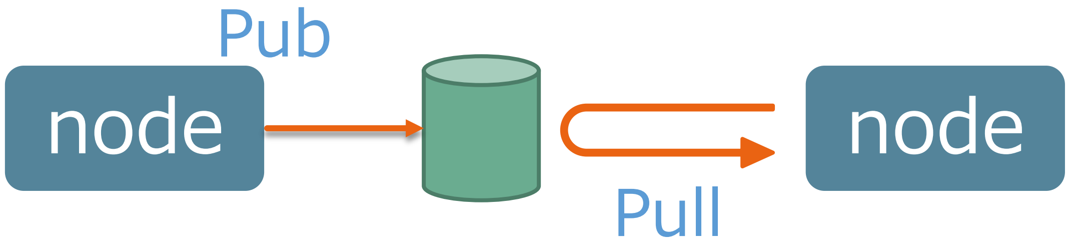
- Pub / Sub (Push)
  - basic pub/sub method



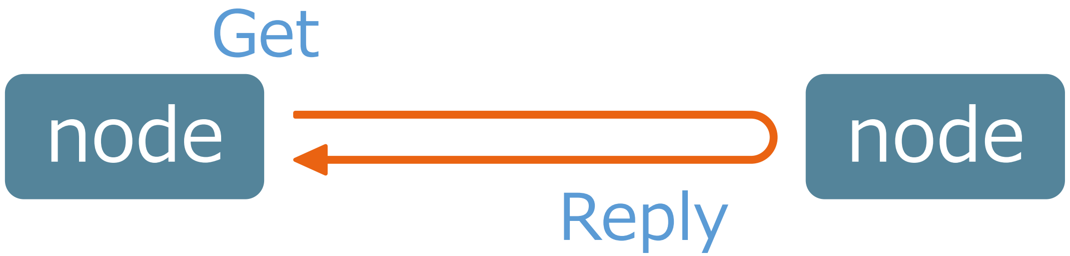
- Pub / Sub (Pull)
  - Sub receives in its own timing



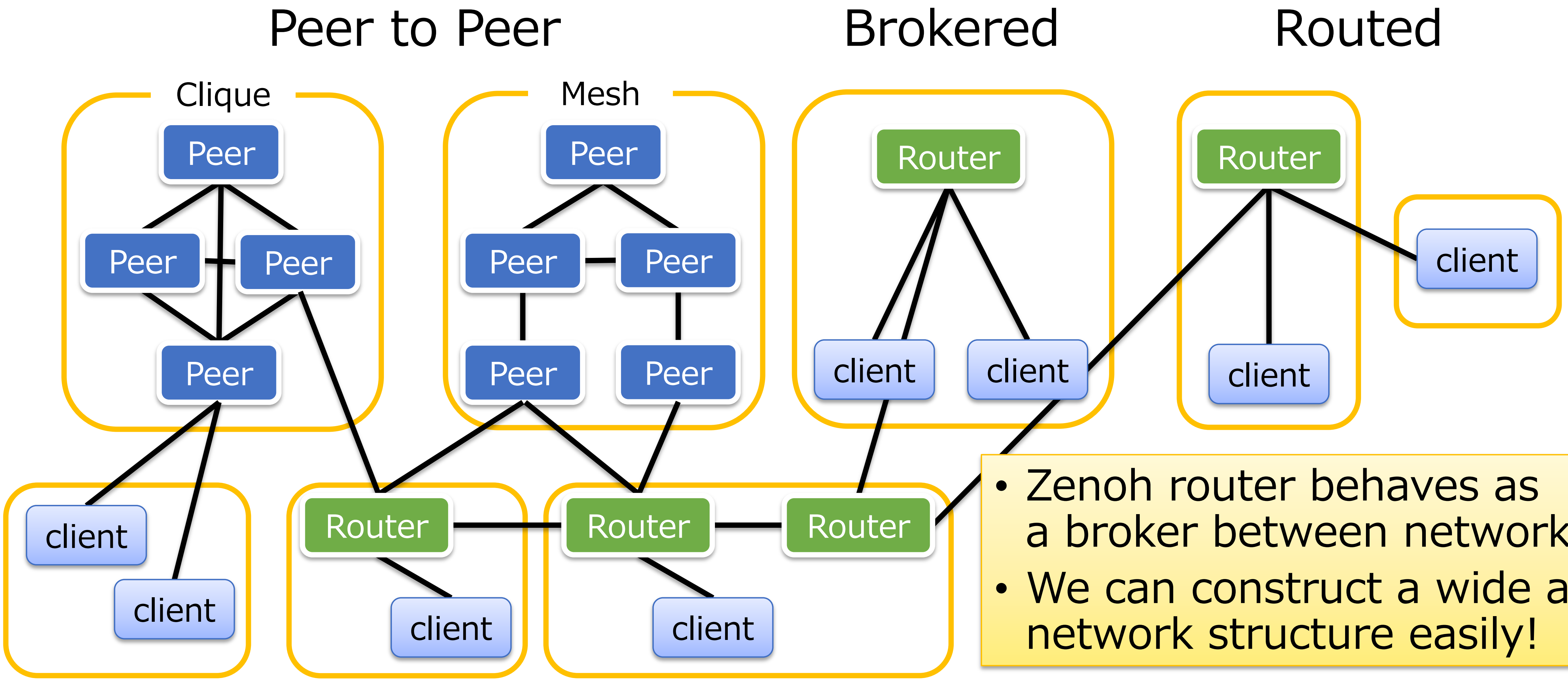
- Pub / Store / Get
  - KVS based computation



- Get / Reply
  - RPC-like communication



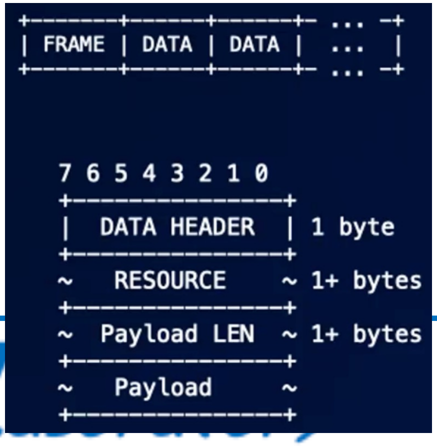
Zenoh Scalable



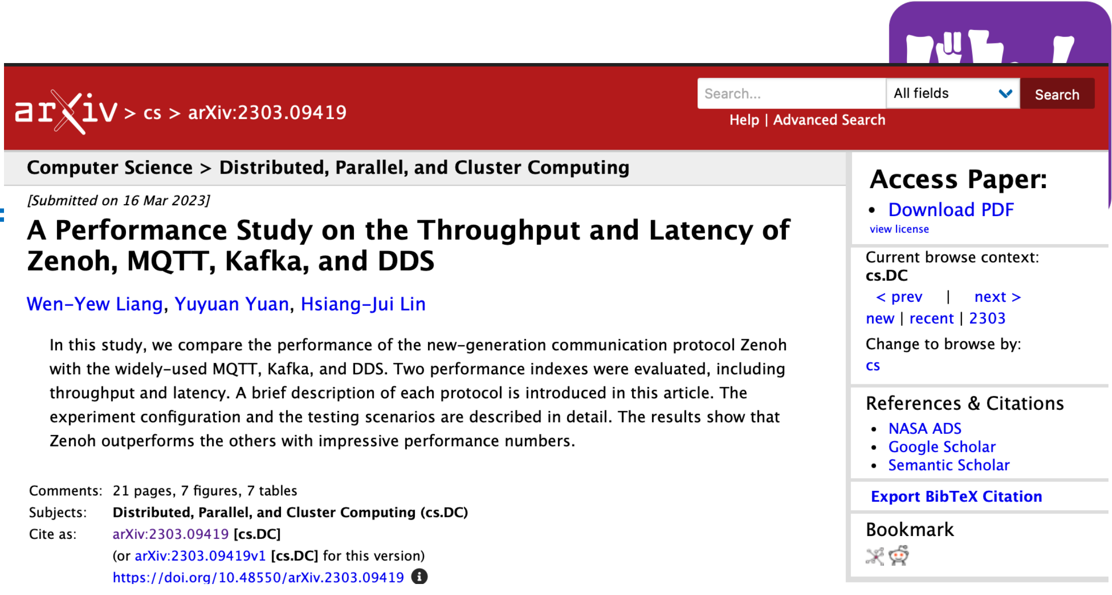
- Zenoh router behaves as a broker between networks
- We can construct a wide area network structure easily!

Zenoh Fast

- Low latency and High throughput
  - 10 us latency in the single machine, 16 us in multiple machines (P2P config.)
  - ~70 Gbps at 8 KB payload
    - ✓35x higher than MQTT, 23x than Kafka, 3.3x than DDS
- Why?: minimum wire overhead
  - only 5 bytes for delivering messages



```
+-------+-------+-------+- ... -+
| FRAME | DATA  | DATA  |  ...  |
+-------+-------+-------+- ... -+

 7 6 5 4 3 2 1 0
+---------------+
| DATA HEADER   | 1 byte
+---------------+
~   RESOURCE    ~ 1+ bytes
+---------------+
~  Payload LEN  ~ 1+ bytes
+---------------+
~   Payload     ~
+---------------+
```



arXiv > cs > arXiv:2303.09419

Computer Science > Distributed, Parallel, and Cluster Computing

[Submitted on 16 Mar 2023]

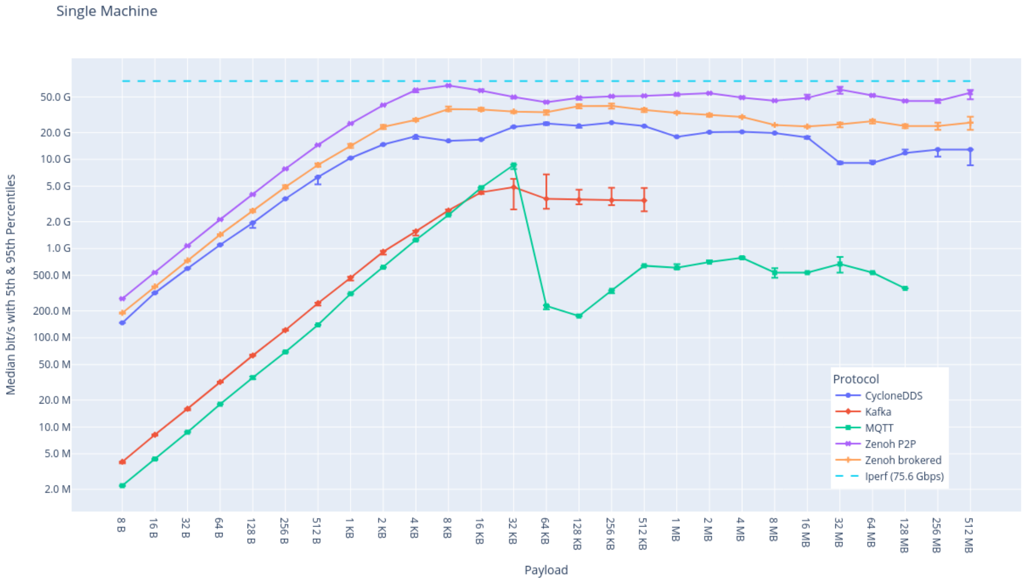
A Performance Study on the Throughput and Latency of Zenoh, MQTT, Kafka, and DDS

Wen-Yew Liang, Yuyuan Yuan, Hsiang-Jui Lin

In this study, we compare the performance of the new-generation communication protocol Zenoh with the widely-used MQTT, Kafka, and DDS. Two performance indexes were evaluated, including throughput and latency. A brief description of each protocol is introduced in this article. The experiment configuration and the testing scenarios are described in detail. The results show that Zenoh outperforms the others with impressive performance numbers.

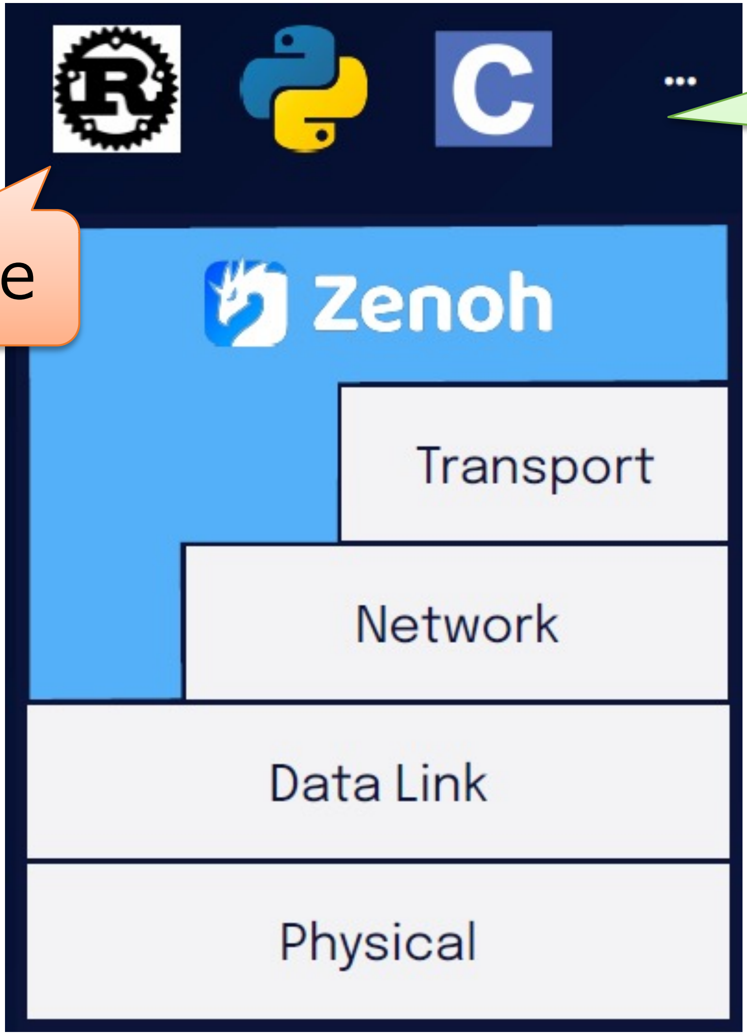
Comments: 21 pages, 7 figures, 7 tables
Subjects: **Distributed, Parallel, and Cluster Computing (cs.DC)**
Cite as: arXiv:2303.09419 **[cs.DC]**
(or arXiv:2303.09419v1 **[cs.DC]** for this version)
https://doi.org/10.48550/arXiv.2303.09419

arxiv:2303.09419



https://zenoh.io/blog/2023-03-21-zenoh-vs-mqtt-kafka-dds/

Zenoh Runs Everywhere!



Native

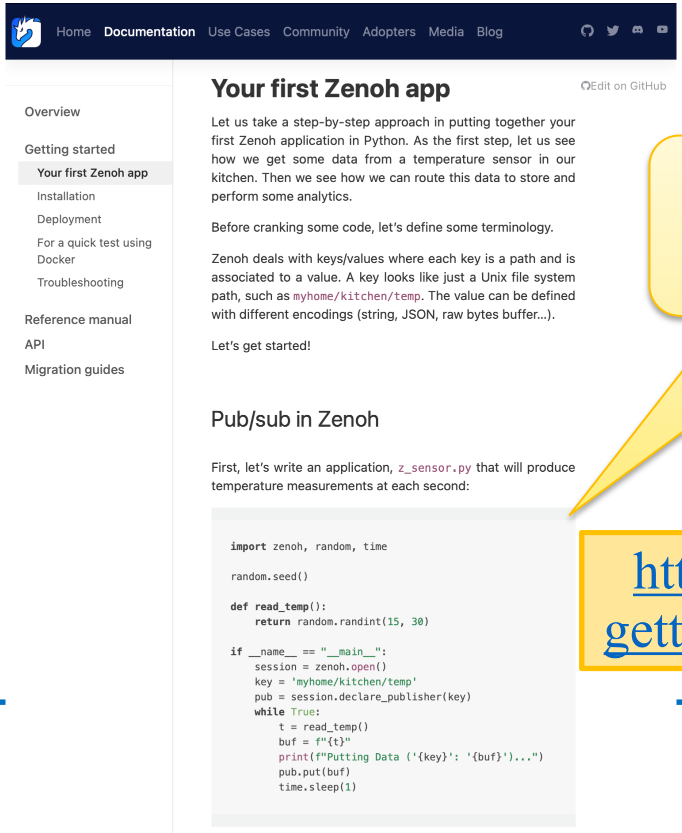
APIs for various languages

- zenoh-python
- zenoh-c
- zenoh-cpp
- zenoh-java
- zenoh-kotlin
- zenoh-csharp
- zenoh-go

QUIC, TLS, TCP,
UDP Unicast,
UDP Multicast

IPv4, IPv6
6LoWPAN

WiFi, Ethernet,
Bluetooth, Serial



Your first Zenoh app

Let us take a step-by-step approach in putting together your first Zenoh application in Python. As the first step, let us see how we get some data from a temperature sensor in our kitchen. Then we see how we can route this data to store and perform some analytics.

Before cranking some code, let's define some terminology.

Zenoh deals with keys/values where each key is a path and is associated to a value. A key looks like just a Unix file system path, such as `myhome/kitchen/temp`. The value can be defined with different encodings (string, JSON, raw bytes buffer...).

Let's get started!

Pub/sub in Zenoh

First, let's write an application, `z_sensor.py` that will produce temperature measurements at each second:

```python
import zenoh, random, time

random.seed()

def read_temp():
    return random.randint(15, 30)

if __name__ == "__main__":
    session = zenoh.open()
    key = 'myhome/kitchen/temp'
    pub = session.declare_publisher(key)
    while True:
        t = read_temp()
        buf = f"{t}"
        print(f"Putting Data ('{key}': '{buf}')...")
        pub.put(buf)
        time.sleep(1)
```

Getting Started with Python

https://zenoh.io/docs/getting-started/first-app/

We love Elixir!!

Functional language (appeared in 2012)

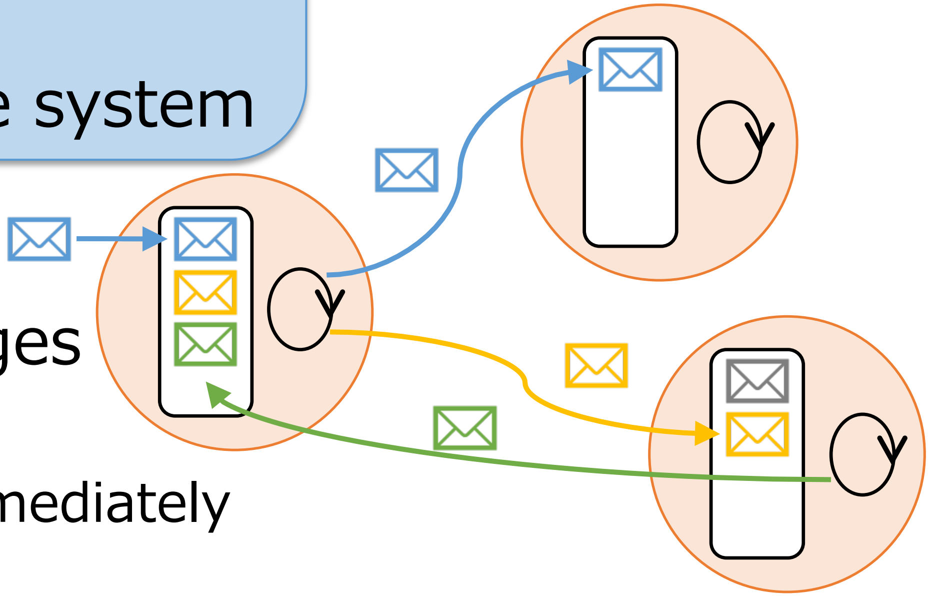
elixir

ERLANG

Operated on Erlang VM (BEAM)

- lightweight processes with robustness
- highly concurrency/parallelism
- soft real-time feature
- easy to realize distributed and fault tolerance system

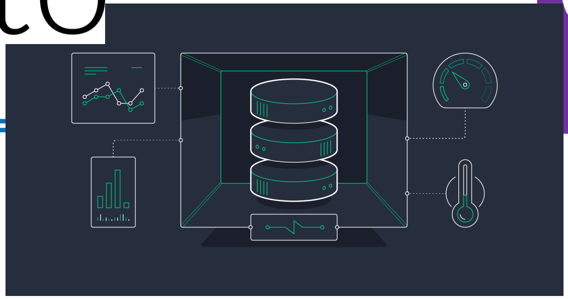
- Similar to **Actor Model**
  - Actors (processes) send and receive messages
  - "Let it Crash”: The problematic process should be promptly crashed and restored immediately
  - Suitable to IoT system development!



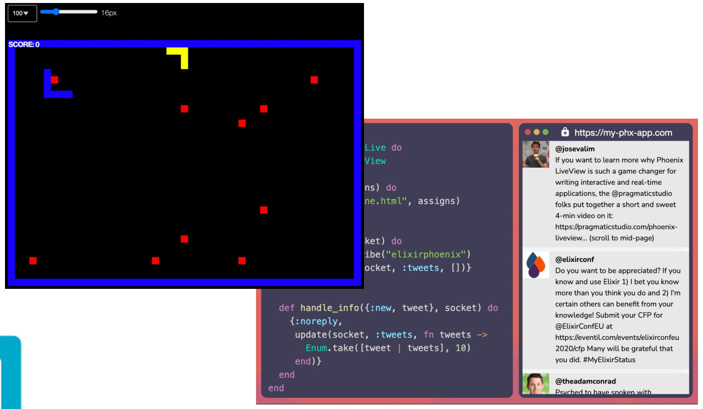
* The copyrights of these logos belong to their respective creators

We love Elixir!!

- Awesome ecosystem!



Phoenix Framework



```
Live do
View

ns) do
ne.html", assigns)

ket) do
ibe("elixirphoenix")
ocket, :tweets, [])}

  def handle_info({:new, tweet}, socket) do
    {:noreply,
     update(socket, :tweets, fn tweets ->
       Enum.take([tweet | tweets], 10)
     end)}
  end
end
```

https://my-phx-app.com

@josevalim
If you want to learn more why Phoenix LiveView is such a game changer for writing interactive and real-time applications, the @pragmaticstudio folks put together a short and sweet 4-min video on it: https://pragmaticstudio.com/phoenix-liveview... (scroll to mid-page)

@elixirconf
Do you want to be appreciated? If you know and use Elixir 1) I bet you know more than you think you do and 2) I'm certain others can benefit from your knowledge! Submit your CFP for @ElixirConfEU at https://eventil.com/events/elixirconfeu2020/cfp Many will be grateful that you did. #MyElixirStatus

@theadamconrad
Psyched to have spoken with

ecto

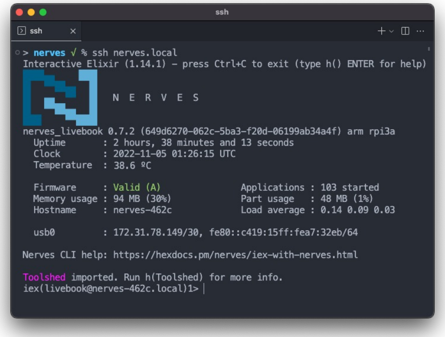
Nx

Nx

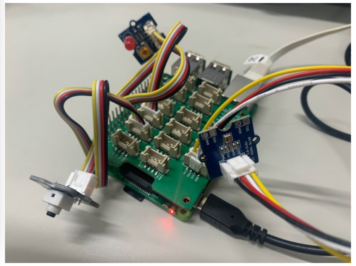
- Multi-dimensional typed arrays (aka tensors)
- Numerical definitions (defn)
  - A subset of Elixir for numerical computation
  - Automatic differentiation

```
model =
  Axon.input({nil, 784})
    |> Axon.dense(128, activation: :relu)
    |> Axon.dropout(rate: 0.5)
    |> Axon.dense(10, activation: :softmax)
```

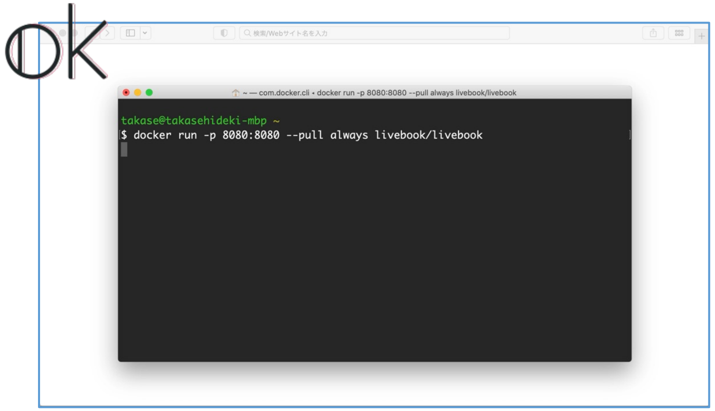
NERVES



```
> nerves √ % ssh nerves.local
Interactive Elixir (1.14.1) - press Ctrl+C to exit (type h() ENTER for help)
              N E R V E S
nerves_livebook 0.7.2 (649d6270-062c-5ba3-f20d-06199ab34a4f) arm rpi3a
  Uptime       : 2 hours, 38 minutes and 13 seconds
  Clock        : 2022-11-05 01:26:15 UTC
  Temperature  : 38.6 ºC

  Firmware     : Valid (A)               Applications : 103 started
  Memory usage : 94 MB (30%)             Part usage   : 48 MB (1%)
  Hostname     : nerves-462c             Load average : 0.14 0.09 0.03

  usb0         : 172.31.78.149/30, fe80::c419:15ff:fea7:32eb/64

Nerves CLI help: https://hexdocs.pm/nerves/iex-with-nerves.html

Toolshed imported. Run h(Toolshed) for more info.
iex(livebook@nerves-462c.local)1> |
```



Livebook



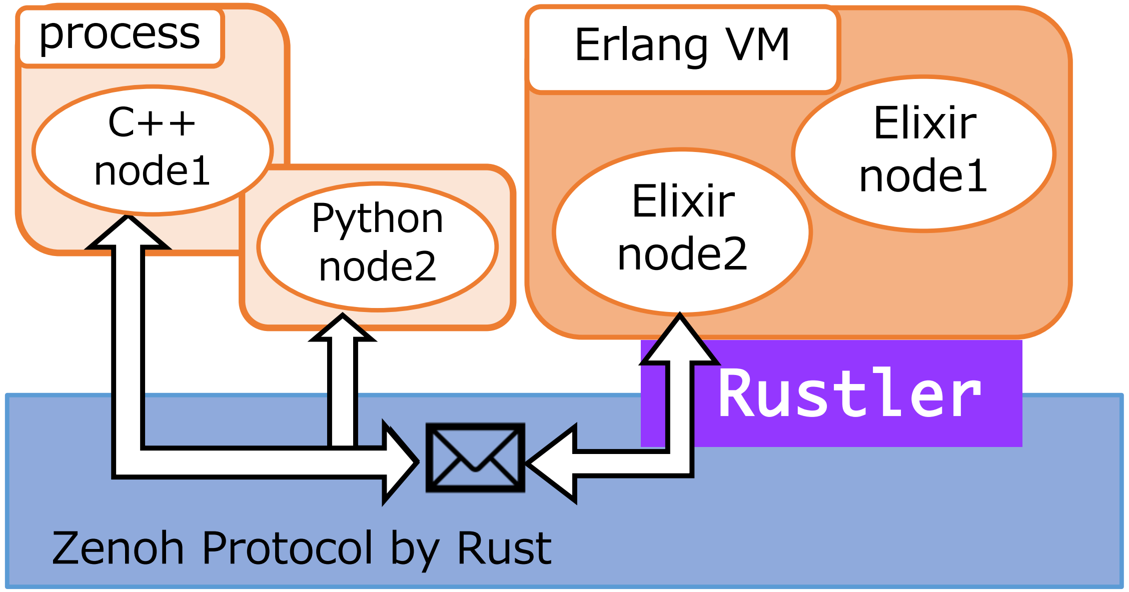
~ — com.docker.cli ◂ docker run -p 8080:8080 --pull always livebook/livebook

```
takase@takasehideki-mbp ~
$ docker run -p 8080:8080 --pull always livebook/livebook
```

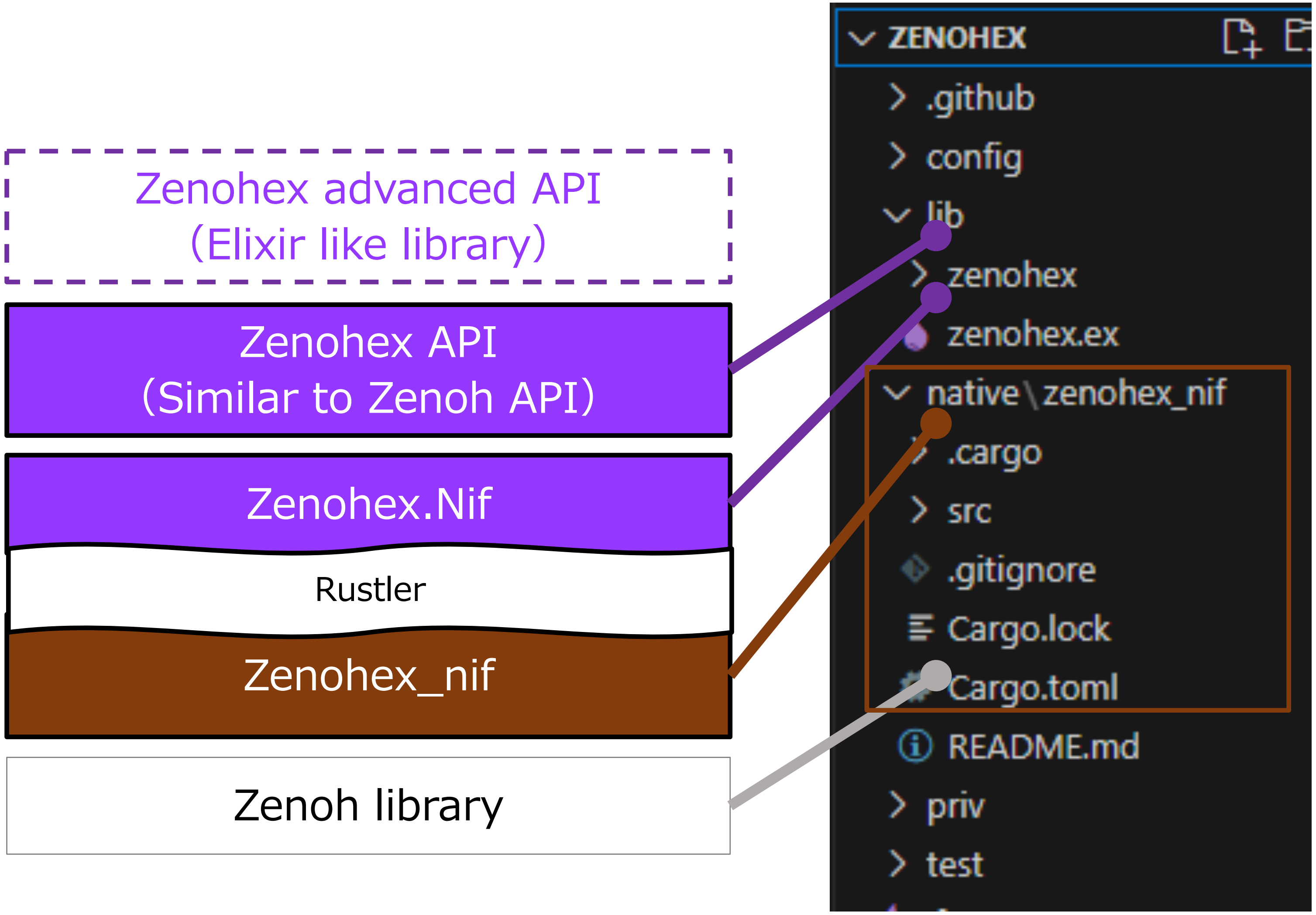
* The copyrights of these logos belong to their respective creators

Zenohex = Zenoh + Elixir

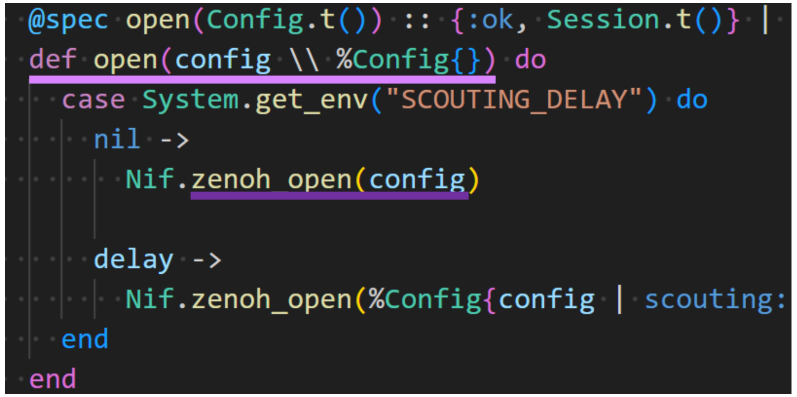
- Why Zenohex?
  - We should find a network library that is over the network
  - Marriage of Zenoh and Elixir could take balance programmability and performance
- How to Implement ?
  - Zenoh is written in Rust
  - Use Rustler
    - ✓Easy to bind Rust and Elixir
    - ✓Generate boilerplate project
    - ✓Integrate cargo and mix build



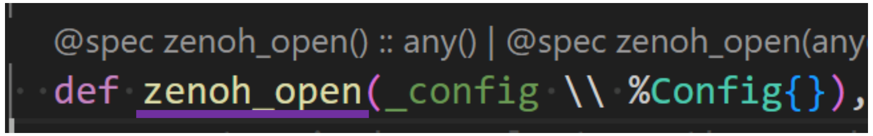
Zenohex Software Structure



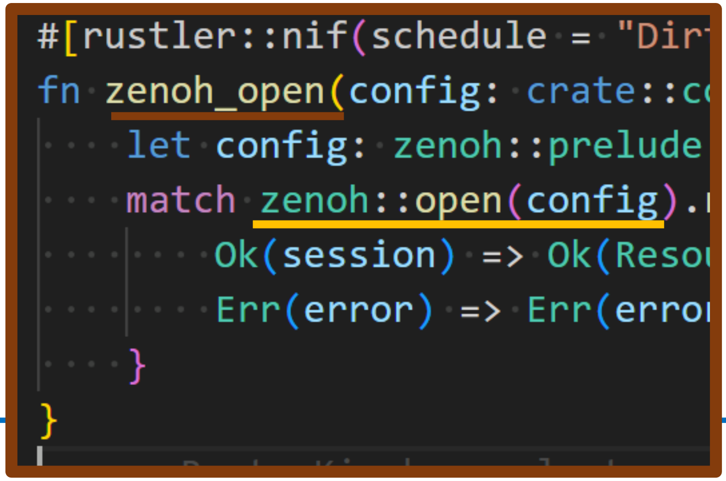
```
ZENOHEX
  > .github
  > config
  v lib
    > zenohex
      zenohex.ex
  v native \ zenohex_nif
    > .cargo
    > src
      .gitignore
      Cargo.lock
      Cargo.toml
    README.md
  > priv
  > test
```



```elixir
@spec open(Config.t()) :: {:ok, Session.t()} |
def open(config \\ %Config{}) do
  case System.get_env("SCOUTING_DELAY") do
    nil ->
      Nif.zenoh_open(config)

    delay ->
      Nif.zenoh_open(%Config{config | scouting:
  end
end
```



```elixir
@spec zenoh_open() :: any() | @spec zenoh_open(any
def zenoh_open(_config \\ %Config{}),
```

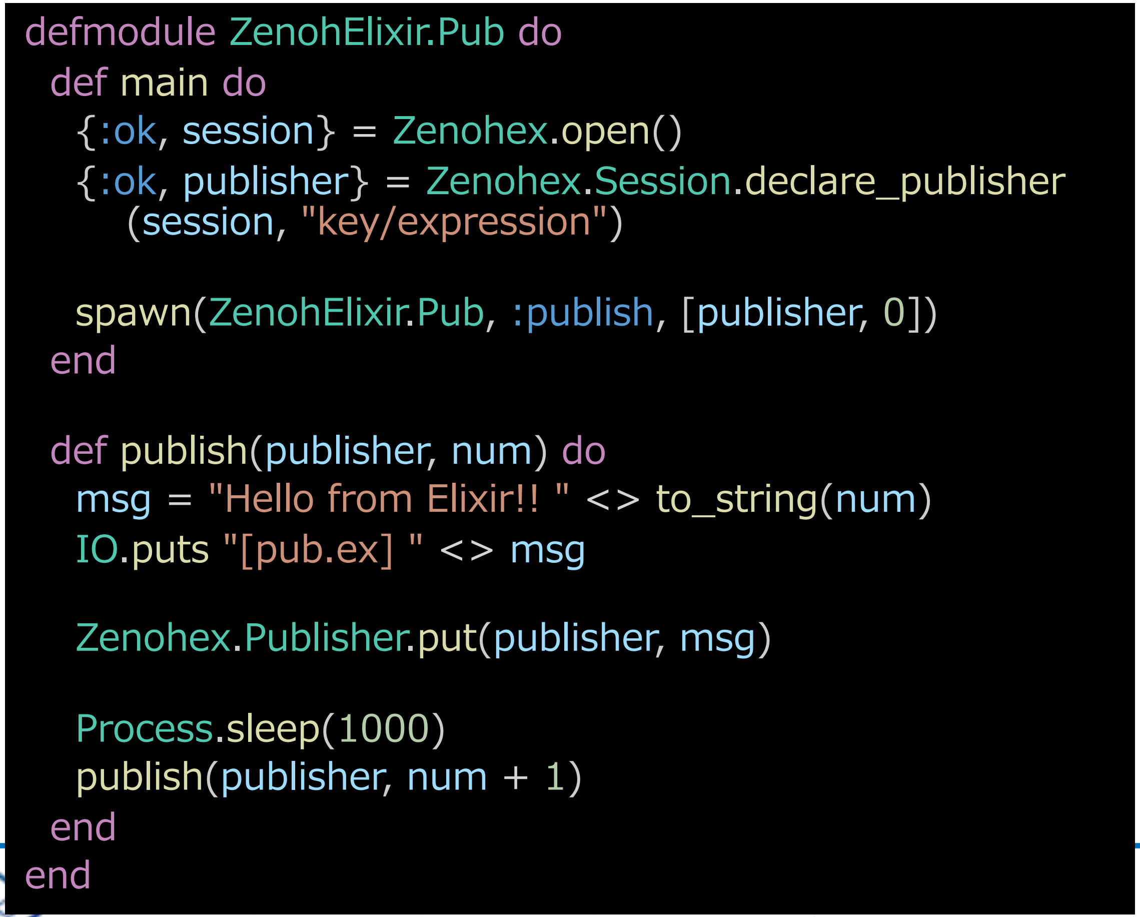


```rust
#[rustler::nif(schedule = "Dir
fn zenoh_open(config: crate::c
    let config: zenoh::prelude
    match zenoh::open(config).
        Ok(session) => Ok(Reso
        Err(error) => Err(erro
    }
}
```

How to use Zenohex

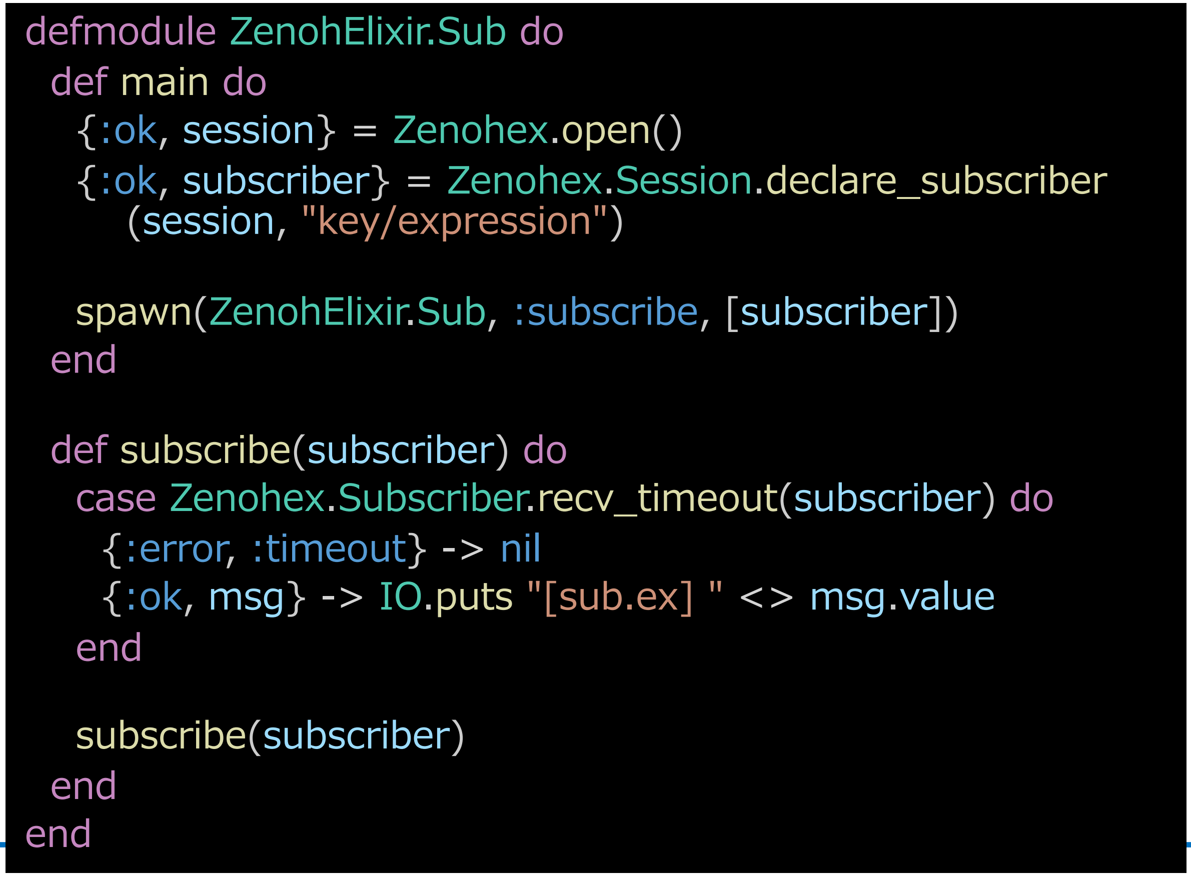
- add {:zenohex, "~> 0.3.0"} to mix.exs

Publisher



```
defmodule ZenohElixir.Pub do
  def main do
    {:ok, session} = Zenohex.open()
    {:ok, publisher} = Zenohex.Session.declare_publisher
        (session, "key/expression")

    spawn(ZenohElixir.Pub, :publish, [publisher, 0])
  end

  def publish(publisher, num) do
    msg = "Hello from Elixir!! " <> to_string(num)
    IO.puts "[pub.ex] " <> msg

    Zenohex.Publisher.put(publisher, msg)

    Process.sleep(1000)
    publish(publisher, num + 1)
  end
end
```

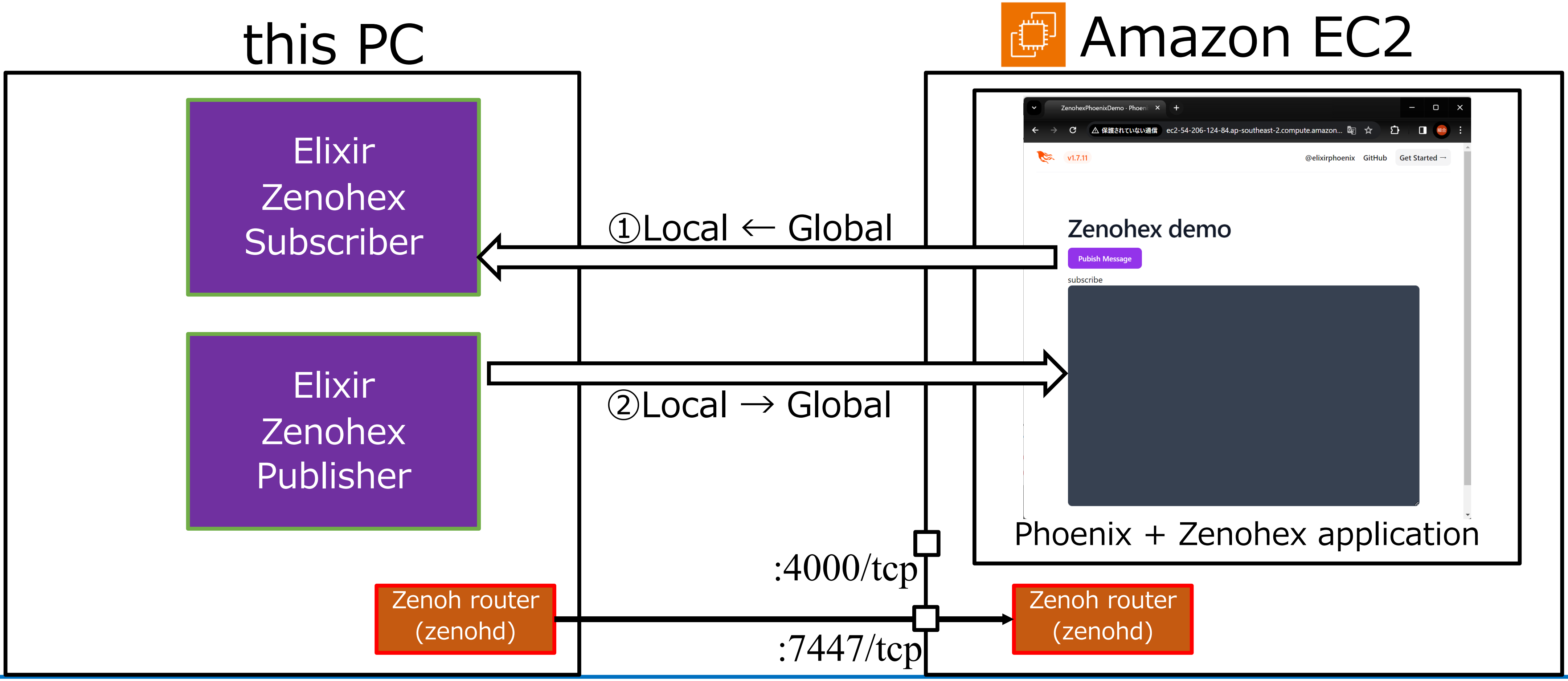
Subscriber



```
defmodule ZenohElixir.Sub do
  def main do
    {:ok, session} = Zenohex.open()
    {:ok, subscriber} = Zenohex.Session.declare_subscriber
        (session, "key/expression")

    spawn(ZenohElixir.Sub, :subscribe, [subscriber])
  end

  def subscribe(subscriber) do
    case Zenohex.Subscriber.recv_timeout(subscriber) do
      {:error, :timeout} -> nil
      {:ok, msg} -> IO.puts "[sub.ex] " <> msg.value
    end

    subscribe(subscriber)
  end
end
```

論よりRUN!! *"ron yori run"* **The RUN is mightier than the word**

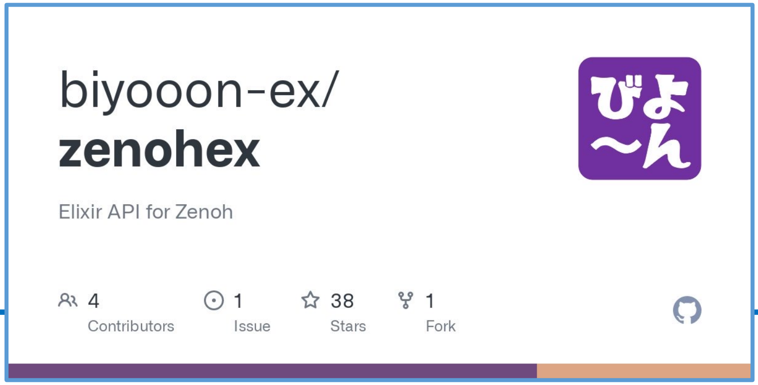
DEMO: over the network



Conclusion

- Zenohex = Zenoh + Elixir
  - Zenoh: lightweight and easy-to-deploy comm. library
  - Elixir: most promising language for IoT systems

- WiP and Future Works
  - Integration to Nerves IoT devices
  - Quantitative evaluation
  - Apply to actual wide-area distributed systems



biyooon-ex/
zenohex

Elixir API for Zenoh

4 Contributors · 1 Issue · 38 Stars · 1 Fork

A part of this work is going as collaborative research with SoftBank Corp., and was supported by the commissioned research (04001) by National Institute of Information and Communications Technology (NICT), Japan.