Towards a Proactive Lightweight Serverless Edge Cloud for Internet-of-Things Applications

Ian-Chin Wang, ***Shixiong Qi***, Elizabeth Liri, K. K. Ramakrishnan

University of California, Riverside

October 26th, 2021

UC RIVERSIDE

Internet-of-Things (IoT)

Why IoT?

Build smarter world with connected sensors

Features of IoT

🤔 limited operational cost

😄 smart h e

IoT needs an **efficient low-cost** backend to host the service **on-demand**

IoT needs to access backends with low latency

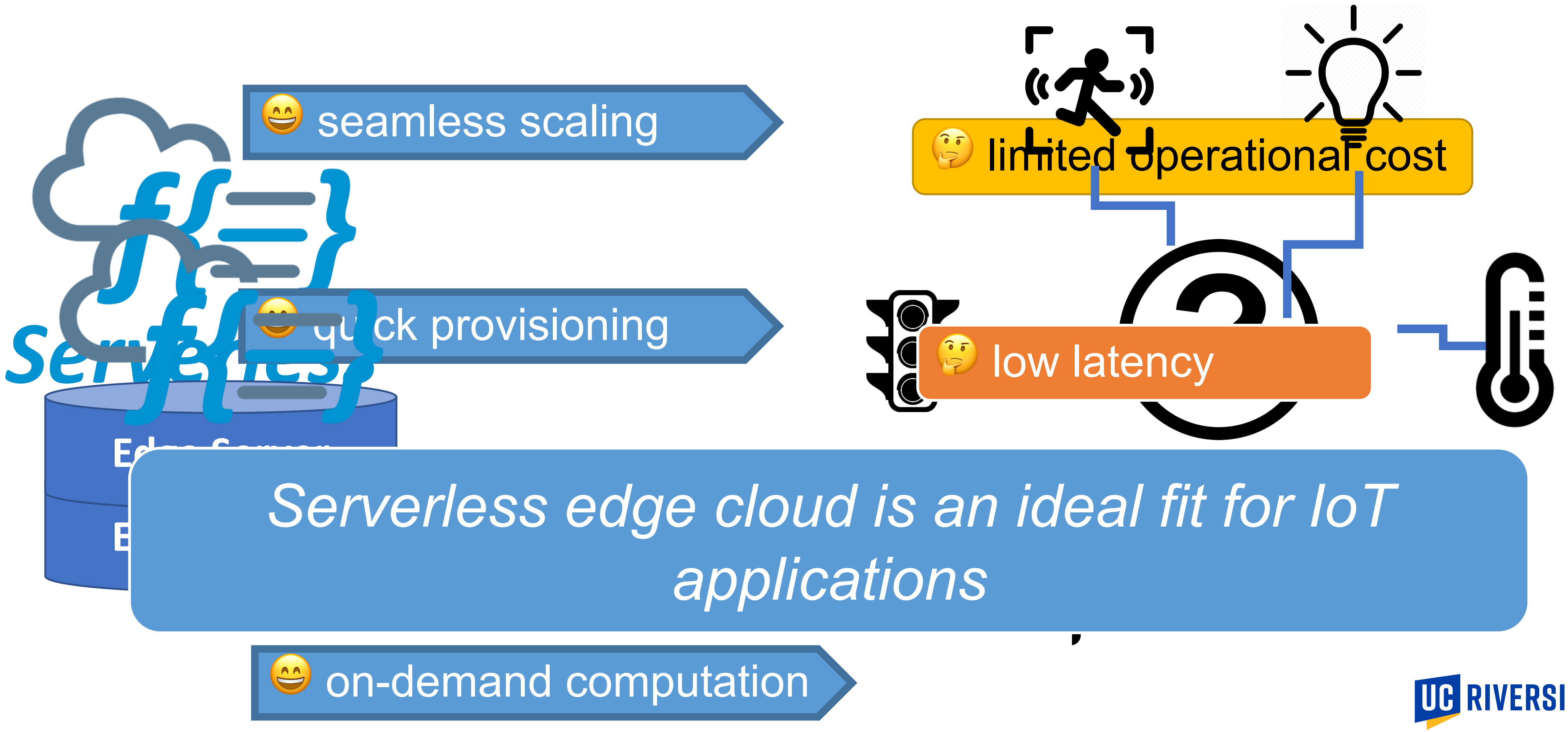
🤔 low latency

😄 security

🤔 low power

UC RIVERSIDE

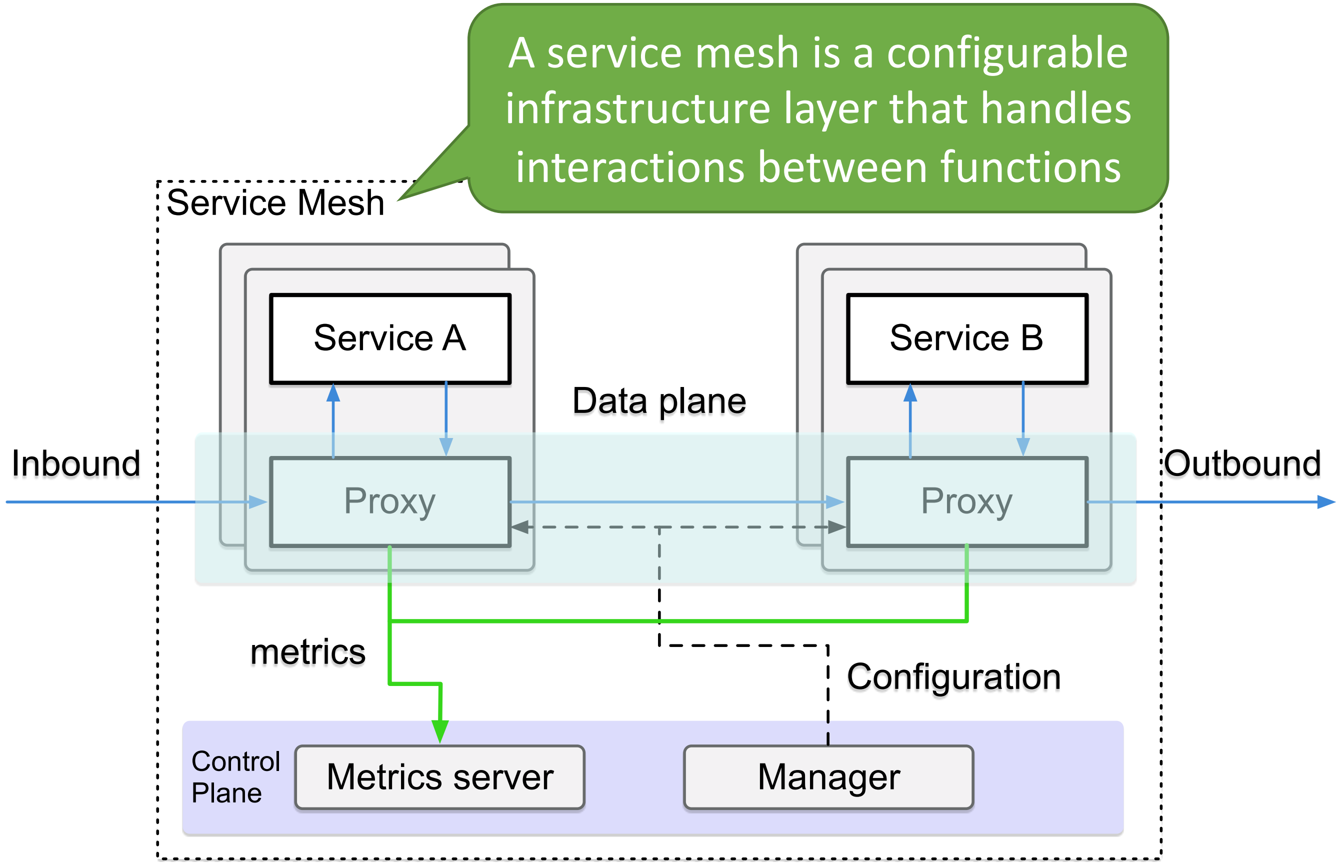
Serverless Edge for IoT Applications



How we build up a serverless edge cloud for IoT

Service mesh for serverless functions

- Managing the service can be challenging when the number of elements increases
  - Use **sidecar proxies** to build up a service mesh: **decoupled** from service instances
- Main functions of sidecar proxy
  - Control plane: monitoring
  - Data plane: routing, load balancing



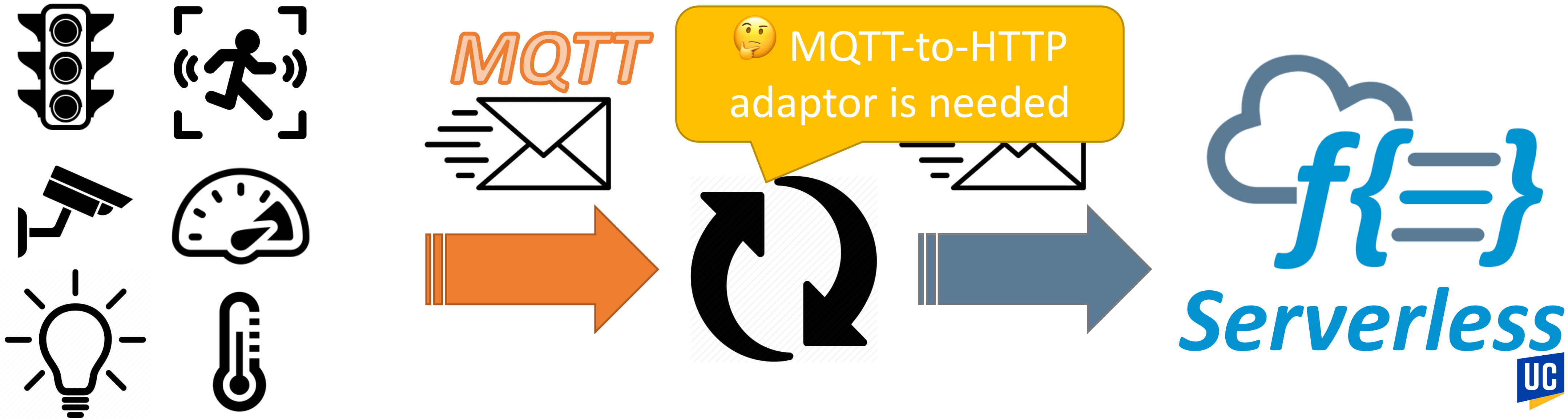
How we build up a serverless edge cloud for IoT

Protocol adaptor: MQTT to HTTP

Serverless frameworks are HTTP-oriented

IoT devices adopt lightweight MQTT protocol to reduce bandwidth & save power

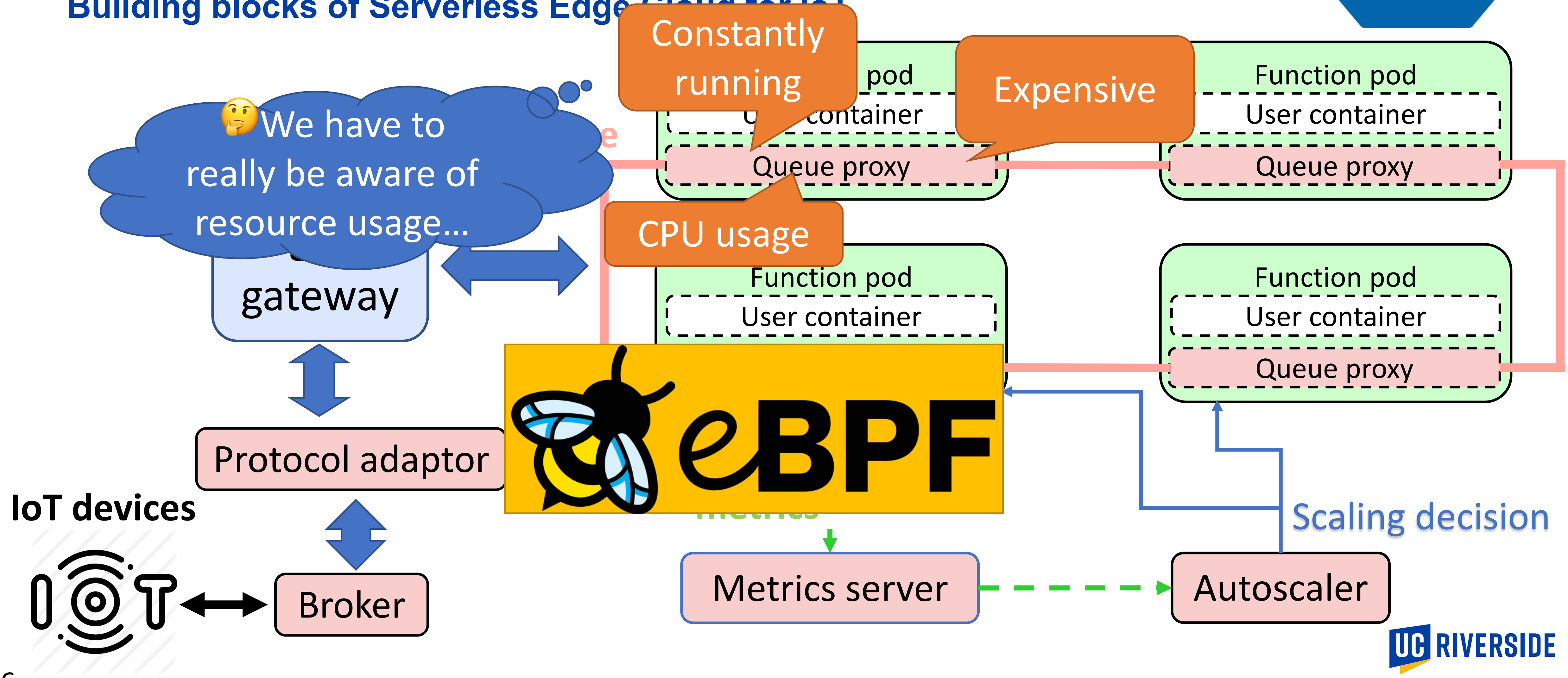
IoT devices



UC RIVERSIDE

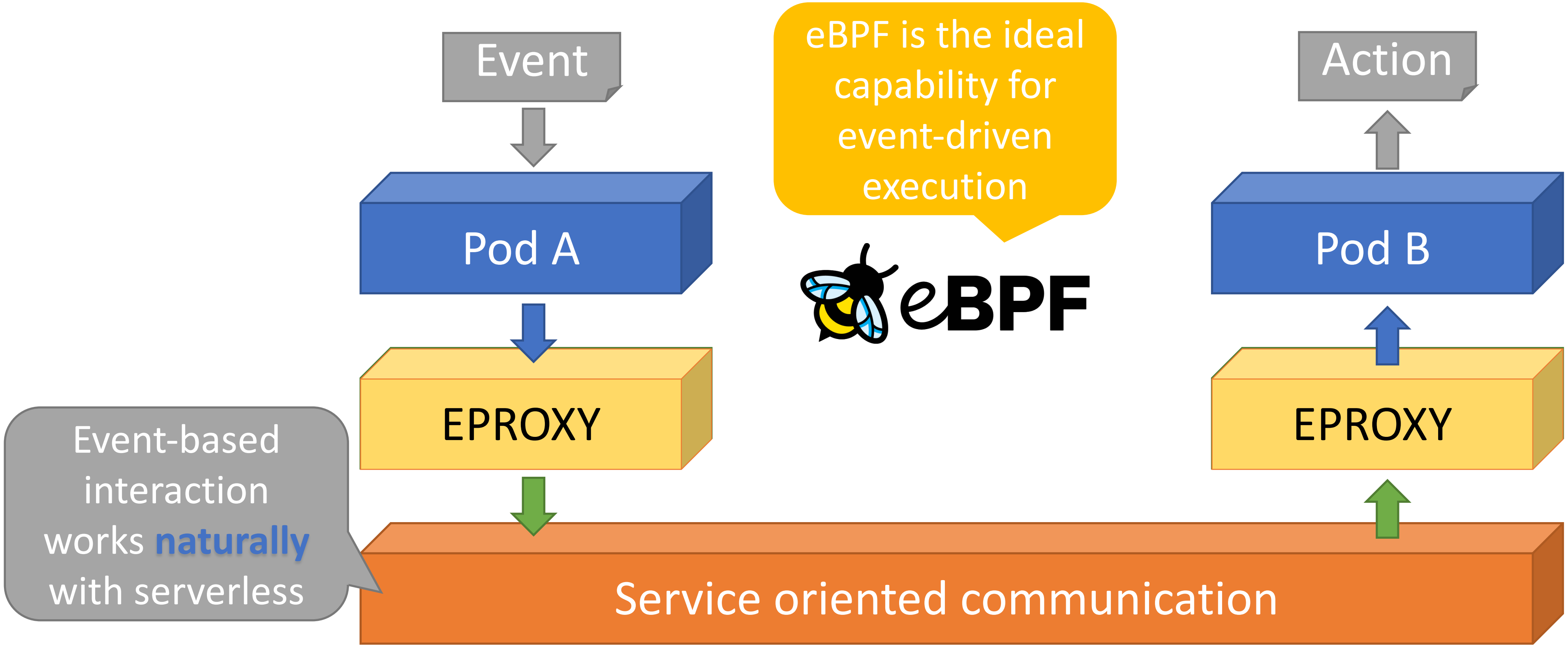
Existing approach and its limitations

Building blocks of Serverless Edge Cloud for IoT



Enhancement: Event-driven Interaction via eBPF

Event-driven proxy (EPROXY)

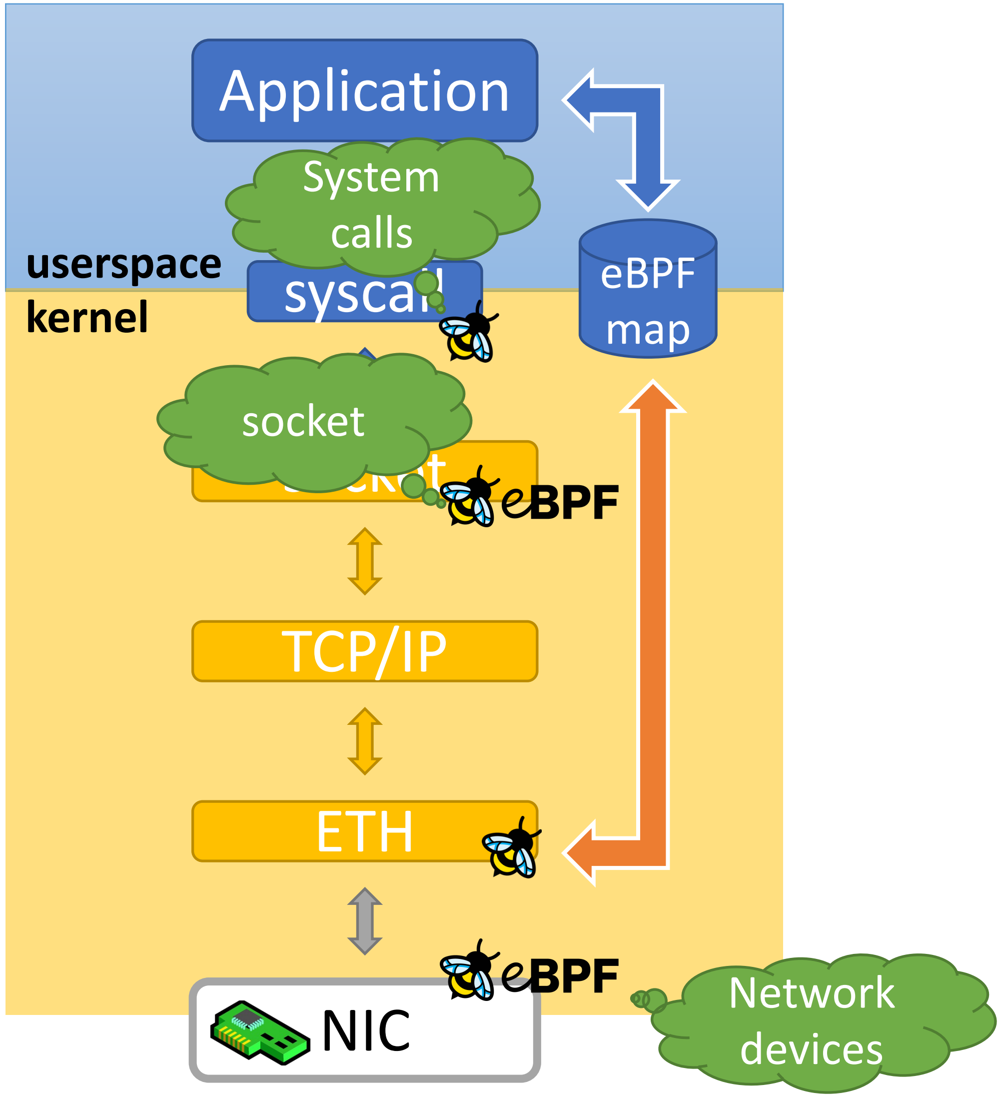


Enhancement: Event-driven Interaction via eBPF

Features of eBPF

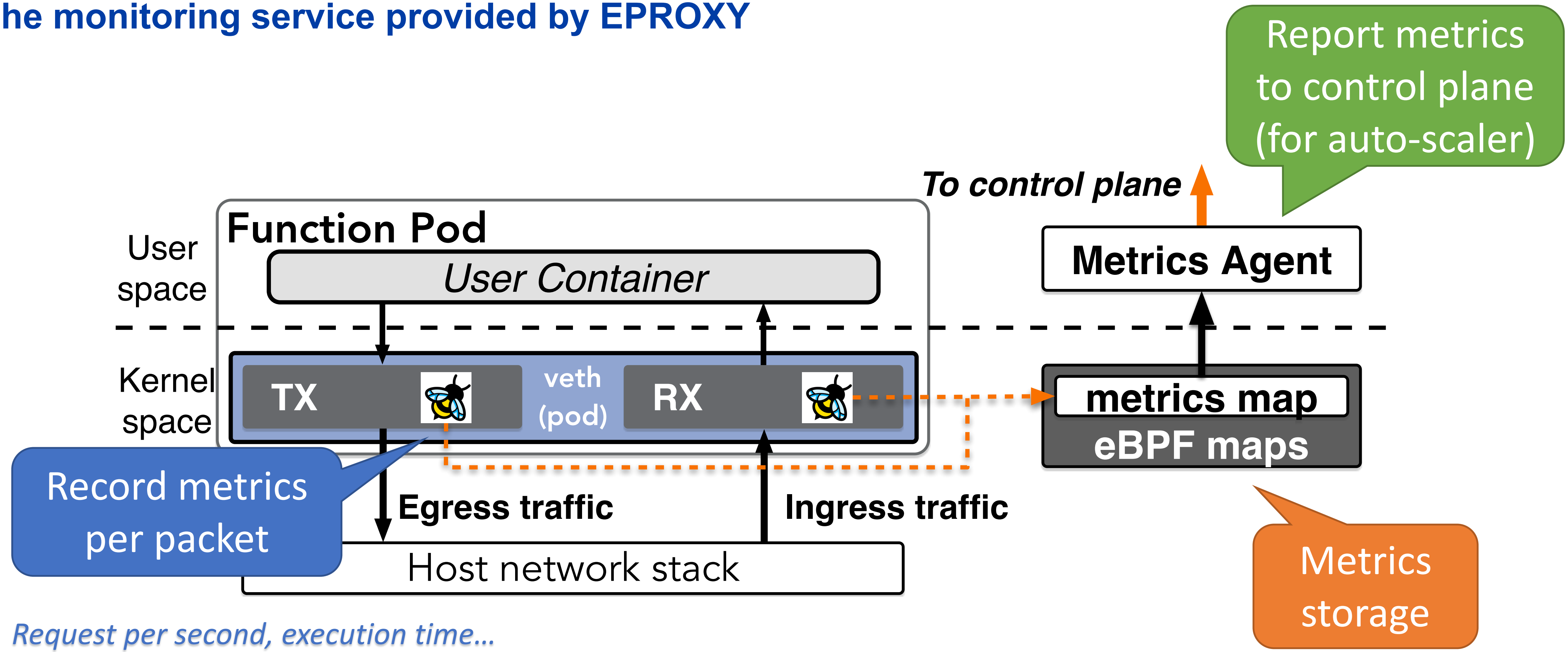
- In-kernel execution
- naturally event-driven
- Various hook points in kernel
  - Used for packet processing, packet filtering, traffic monitoring
- Userspace-kernel interaction
  - eBPF Maps
- Limitations
  - run to completion
  - limited instructions in a single program

The eBPF code needs to be carefully written by the developer



Enhancement: Event-driven Interaction via eBPF

The monitoring service provided by EPROXY



Request per second, execution time…

MQTT Broker & MQTT-to-HTTP Adaptor

- We use **Apache Mosquitto** as the MQTT broker
  - direct the data flow from the IoT sensors to the serverless function chains
  - It can be horizontally scaled by Kubernetes based on traffic rate

- We use **Apache Camel middleware** as adaptor
  - Support MQTT based event message to be processed in HTTP based microservice chains
  - Send the converted message to function pod directly

Overall evaluation

Experiment setup

Cluster setup

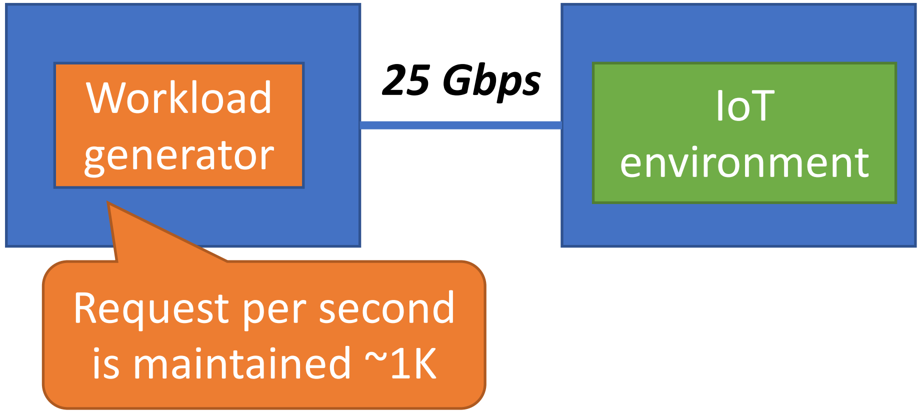
2 nodes connected by 25 Gbps link

Two Protocol models:

HTTP vs. MQTT

Two different sidecar proxies

Queue proxy vs. Eproxy



CPU overhead breakdown

CPU usage of queue proxy is very close to the user-container!

Undesirable in a resource-constrained environment

Components	User container	Queue proxy	Adaptor	Broker	Other	Total
HTTP-QPROXY	6.53	**7.07**	NULL	NULL	1.50	**15.1**
HTTP-EPROXY	6.07	NULL	NULL	NULL	1.92	**7.99**
MQTT-QPROXY	6.59	**6.19**	**1.92**	**0.83**	1.24	**16.77**
MQTT-EPROXY	5.88	NULL	**1.77**	**0.88**	1.45	**9.98**

Using MQTT introduces slight extra overhead

~50% Reduction of CPU by using EPROXY

*"Other" contains EPROXY consumption

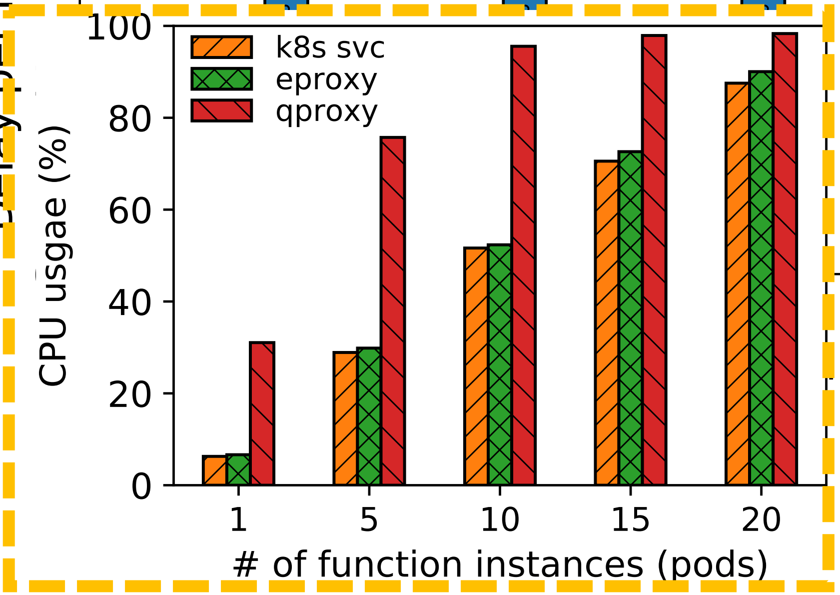
CPU usage and request processing delay

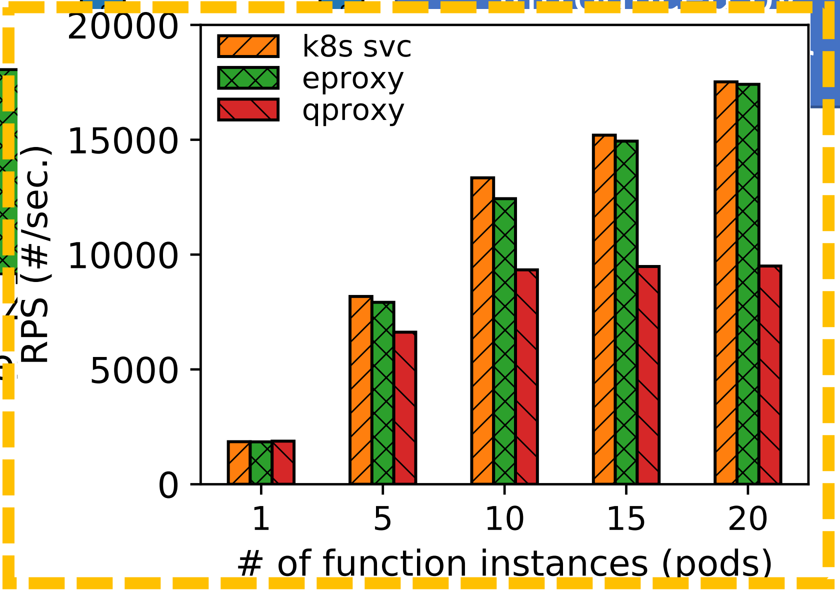
We use apache benchmark (HTTP traffic) to stress different configured systems to compare performance

The CPU usage reduction with the EPROXY can help considerably **at higher loads** when the *queue proxy* will become a **bottleneck** and start contributing an increasing amount of queue delay

MQTT mode adds extra latency compared to HTTP mode for both the Queue proxy and EPROXY

Extra delay introduced by





Conclusion & Future/in-progress work

- Lightweight eBPF-based protocol adaptor
- An eBPF-based event-driven dataplane
- Accurate traffic load predictor

- reduce the resource usage and improve performance
- Address cold start and improve QoS

A desirable serverless edge environment for IoT applications