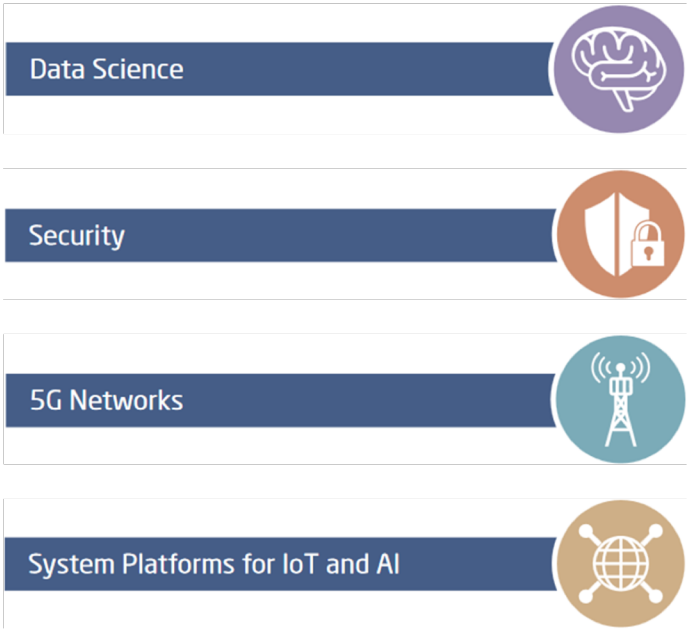
\Orchestrating a brighter world NEC

Fog Function: Serverless Fog Computing for Data Intensive IoT Services

Bin Cheng, Jonathan Fuerst, Gurkan Solmaz (NEC Labs Europe, Germany)
Takuya Sanada (NEC Solution Innovators, Ltd., Japan)

NEC Labs Europe

- Located at Heidelberg, Germany, 1 hour away from Frankfurt by train
- ~100 employees, 70~80 researchers, working in 4 major domains
- We are involved in the research projects from both European Commission and Internal business units

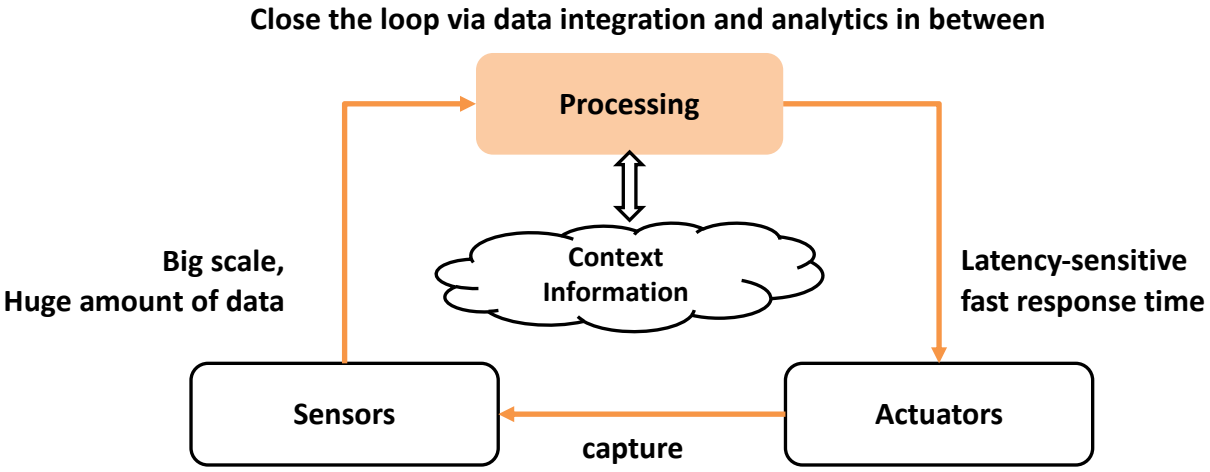


Agenda

- Background
- Motivation and gap analysis
- Fog Function as a service for fog computing
- Major design issues
- System evaluation
- Use case validation
- Conclusion and future work

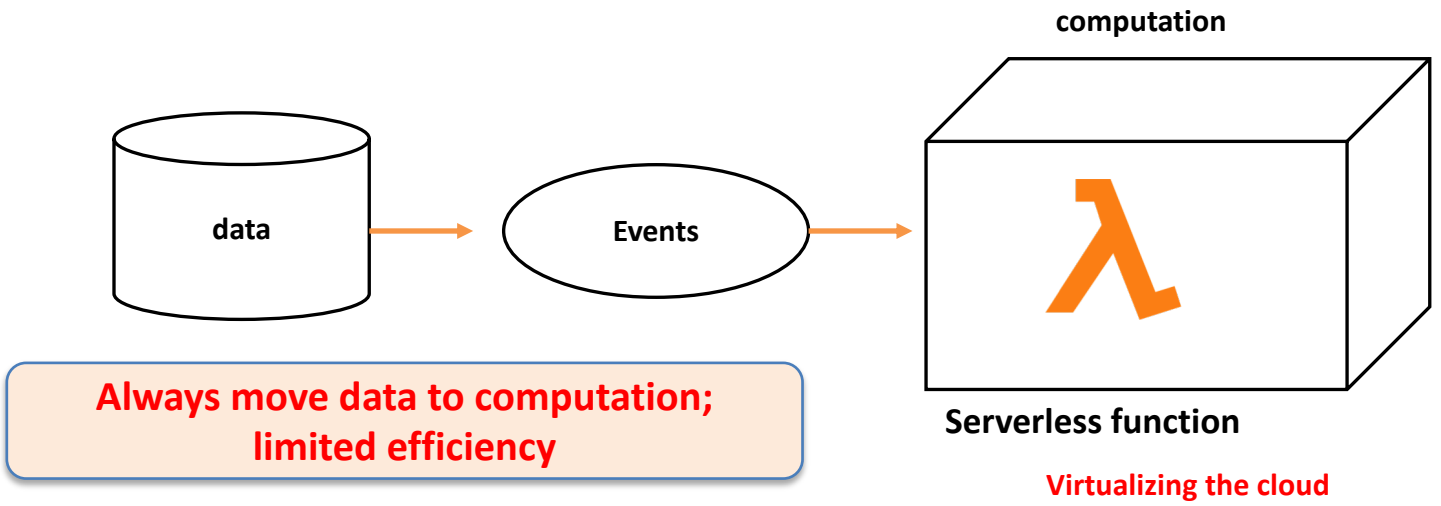
Background (1): Inter of Things (IoT)

- Contextual data are constantly generated and to be used at edges
- Many IoT services required the closed loop of sensing, analysing, decision-making, and reacting; fast response time; automated workload management
- Cloud-only based architecture is not enough to meet service requirements for IoT, due to inefficient bandwidth consumption, latency limit, privacy concerns



Background (2): Serverless Computing

- Function-as-a-Service: a simplified programming model to program your cloud services
- Auto-scaling: make the deployment automatic and transparent so that reducing the operation cost
- Cost model: pay-per-invocation, more reasonable than pay per VM
- Decouple storage and computation and always move data to computation



"Cloud Programming Simplified: A Berkeley View on Serverless Computing"; https://arxiv.org/pdf/1902.03383.pdf

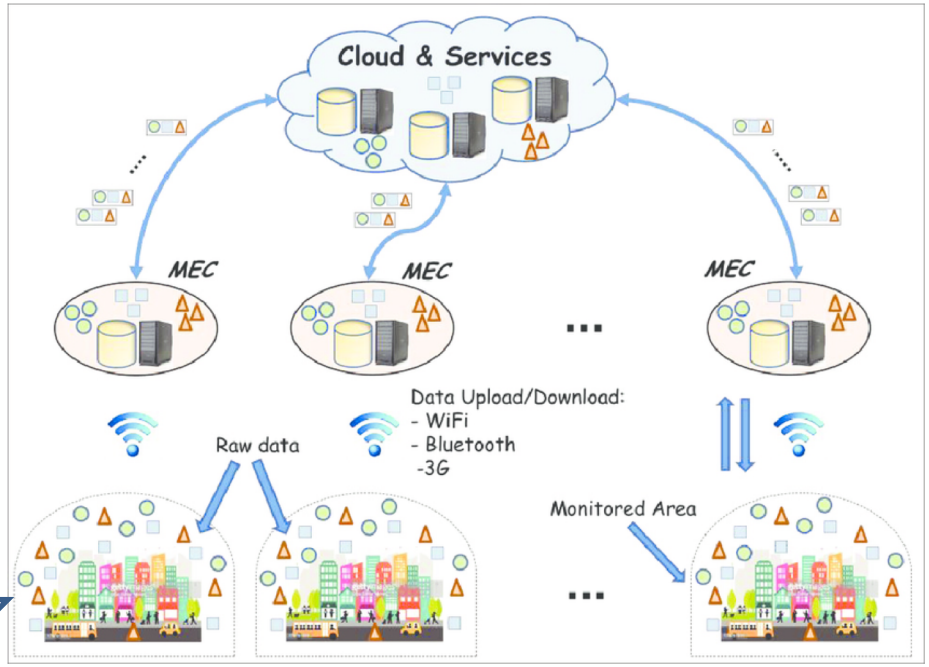
Background (3): Edge Computing/Fog Computing

Cloud-edge infrastructure

huge cloud(s)
(public/private cloud)

heavy edge
(IoT Gateway)

light edge
(IoT device)



Technical benefits of edge/fog computing

- ❑ *Low end-to-end latency for latency sensitive application*
- ❑ *low bandwidth consumption*
- ❑ *Better privacy-preserving*

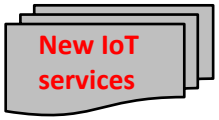
Virtualizing the cloud and edges

Our Target

edge computing has great potential to reduce bandwidth consumption and end-to-end latency, but it raises much more complexity than cloud computing since the cloud-edge environment is more open, heterogeneous, and dynamic

service/application providers

New requirements come frequently



service realization during the development phase

Complicate to realize services due to lack of programming model and poor interoperability: spend months for each service

?

Can we program applications over cloud-edges easily, like programming them in the cloud?

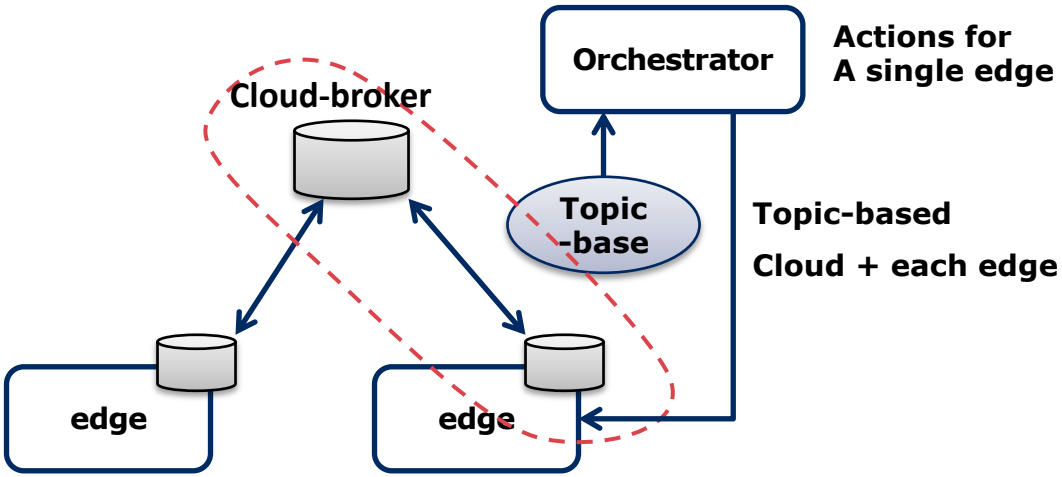
resource management during the deployment phase

No approach of dealing with dynamics like device mobility, instant service usage, temporary failure: applications have to face those issues

?

Can we let the cloud-edge platform to automatically manage and optimize its own resources under such dynamics?

State of the Art: device-oriented approach

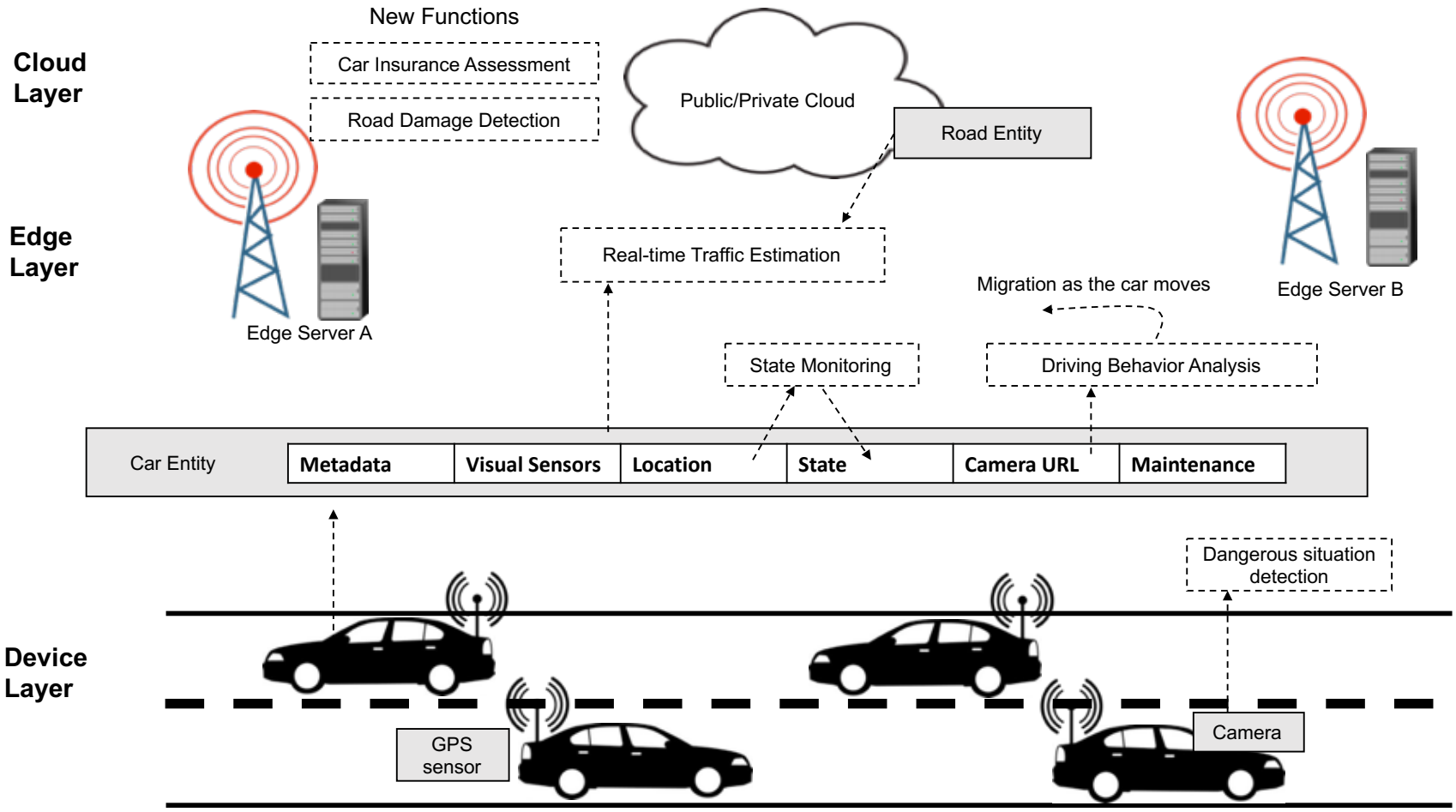


Device-oriented deployment

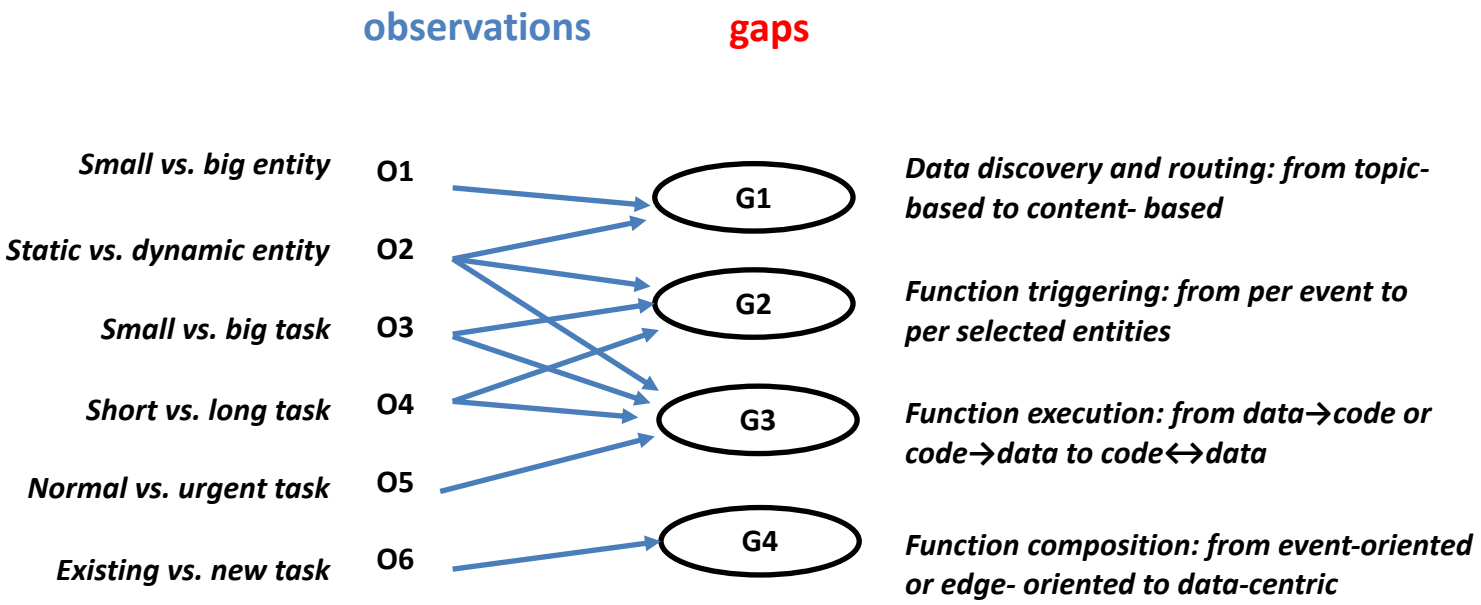
Statically deployment;
Manually configured data path;
Limited flexibility

EdgeX, Azure IoT Edge, AWS Greengrass

Motivating Use Case



Use Case Analysis



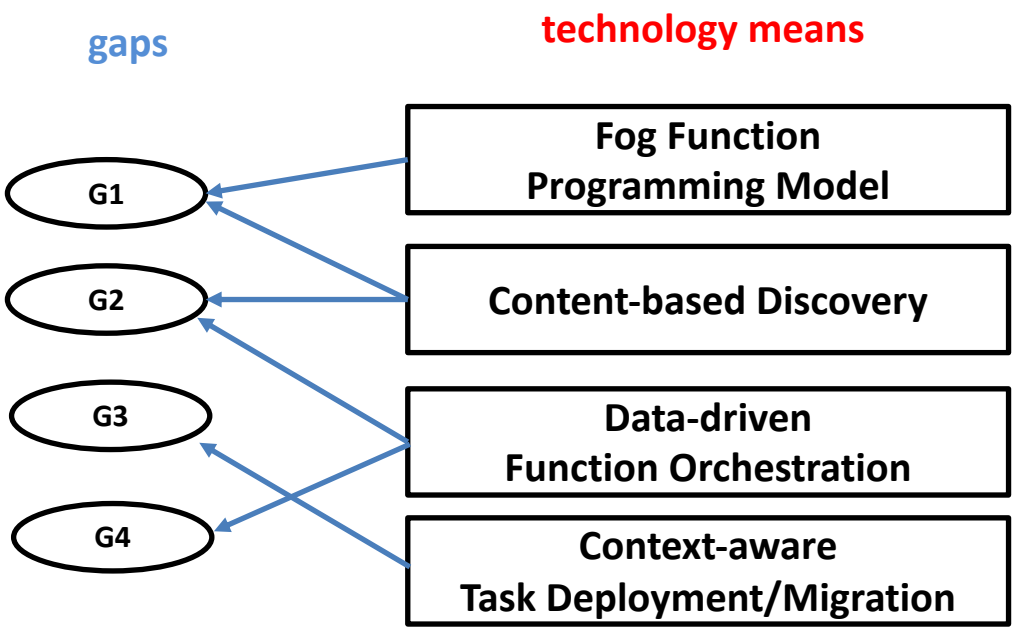
Our Approache to Fulfil the Gaps

Data discovery and routing: from topic-based to content- based

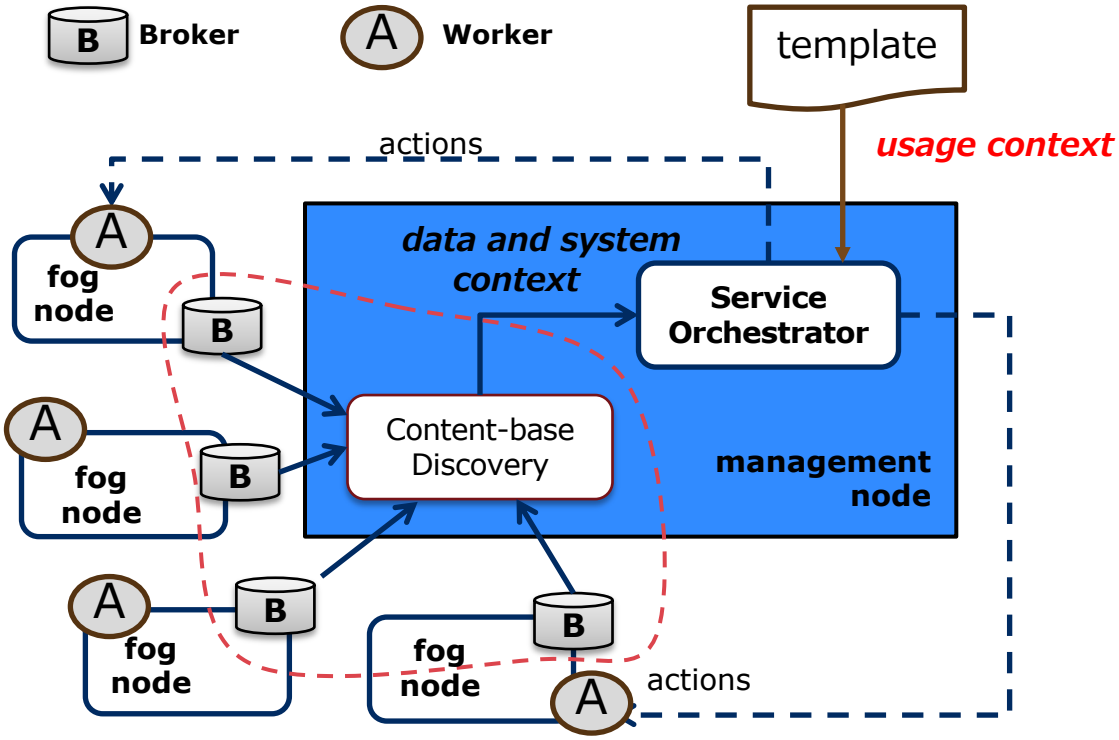
Function triggering: from per event to per selected entities

Function execution: from data→code or code→data to code↔data

Function composition: from event-oriented or edge- oriented to data-centric



System Design



- Easy-to-use programming model
- Content-based Discovery
- Data-driven Orchestration
- Context aware Optimization

Comparison

Systems	FogFlow	Others (EdgeX, Azure IoT Edge/AWS Greengrass)
Triggering-mechanism	**Content-based**	Topic-based
View for orchestration	**Global (all edges + cloud)**	Each edge + backup broker in the cloud
Programming model(s)	**Data-oriented**	Device-oriented
Mobility support	**Yes**	No

Programming Model: Fog Function

```
function(entity, publish, query, subscribe)
```

- *Name*: a unique identity of the function.
- *Operator*: the name of a data processing operator. The operator is implemented as a dockerized application based on the interface of fog function described below. The specified operator is instantiated at runtime by a Worker as a task with its configured inputs and outputs. The task is deployed in a dedicated Docker container running on a fog node.
- *Inputs*: a set of selected inputs required by the operator to do internal data processing.
- *Outputs*: the entity type generated by the operator.
- *Geoscope*: the geoscope to be applied when selecting input data for this fog function.
- *Priority*: the priority of this fog function, which will be taken into account by workers to decide how to assign their limited resources to different functions at fog nodes.
- *SLO*: the expected **S**ervice **L**evel **O**bjective, which is defined as various optimization goals, for example, minimizing the latency to produce outputs, maximizing the accuracy of generated results, or minimizing the bandwidth usage across fog nodes. Different SLO leads to different task deployment plans.

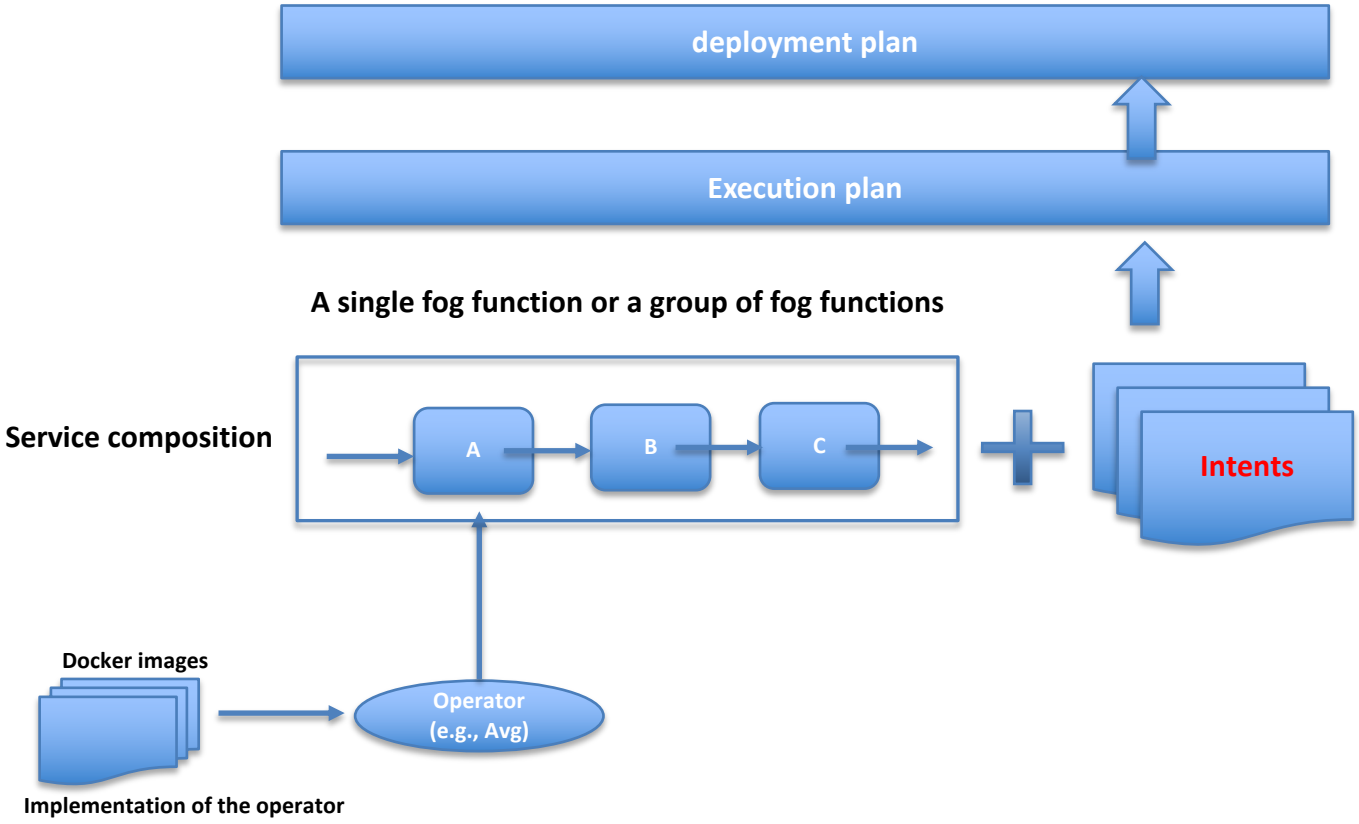
Operator

+

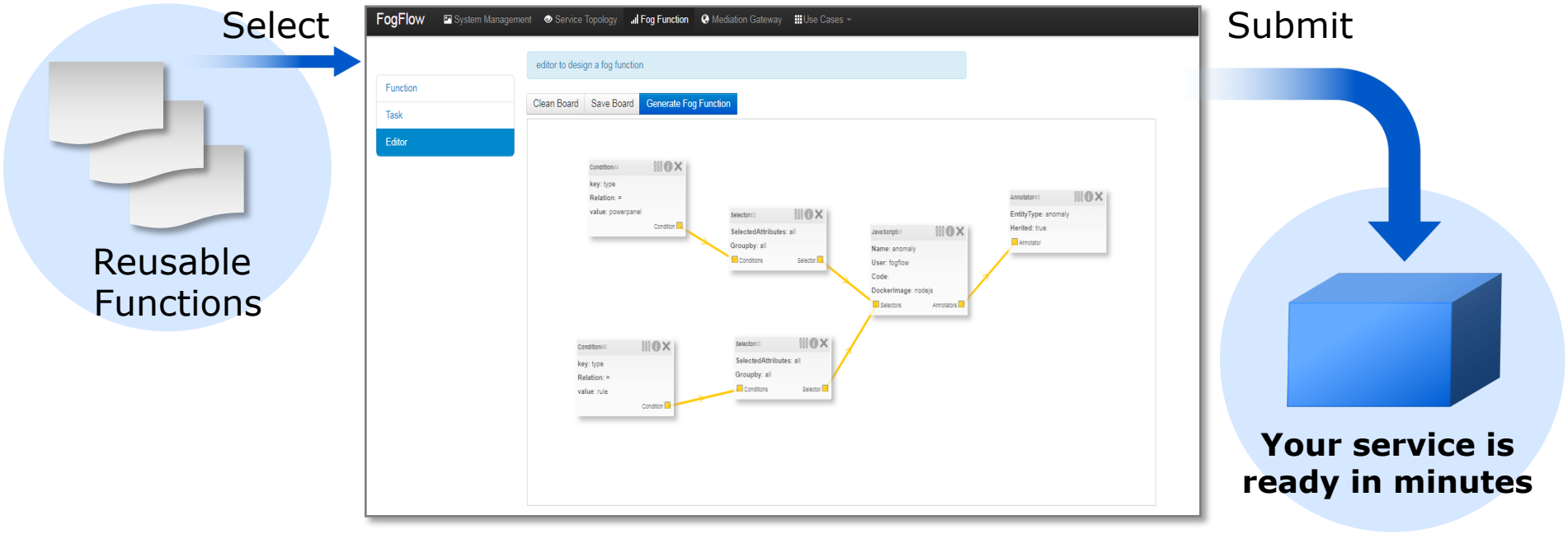
Function annotation

- *SelectedType*: the entity type of this selected input.
- *AttributeSet*: the required attributes of the selected entity.
- *Constraints*: the filters to further select input entities based on some specific attribute values.
- *GroupBy*: the granularity to control how many tasks should be instantiated and how the selected input entities should be assigned across its tasks. It can be defined as "per entityID", "per entityType", or "per attributeValue".
- *Scoped*: this could be true or false and it is used to decide whether geoscope should be applied to select input data when the geoscope is defined with the fog function.

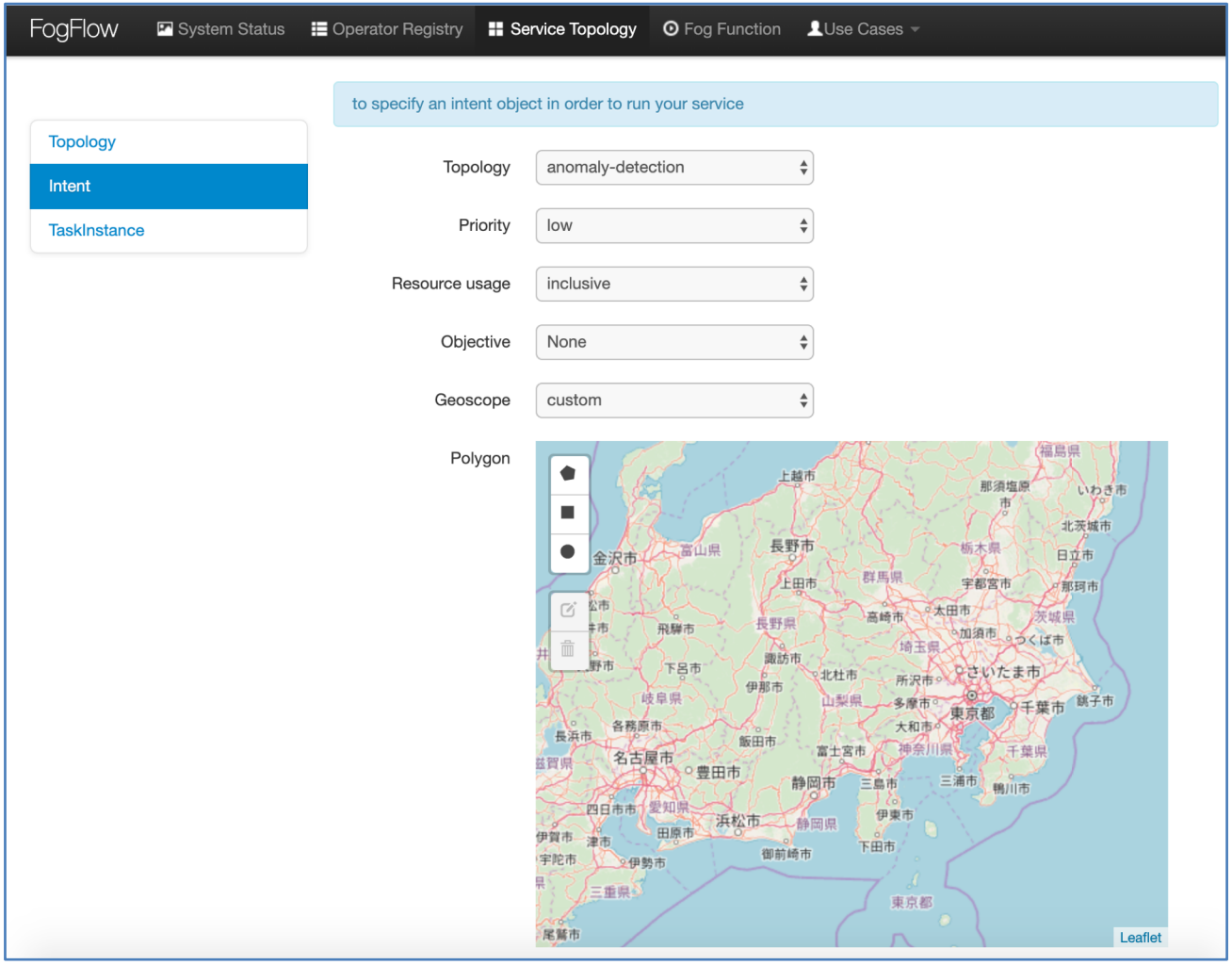
Dynamic Service Composition



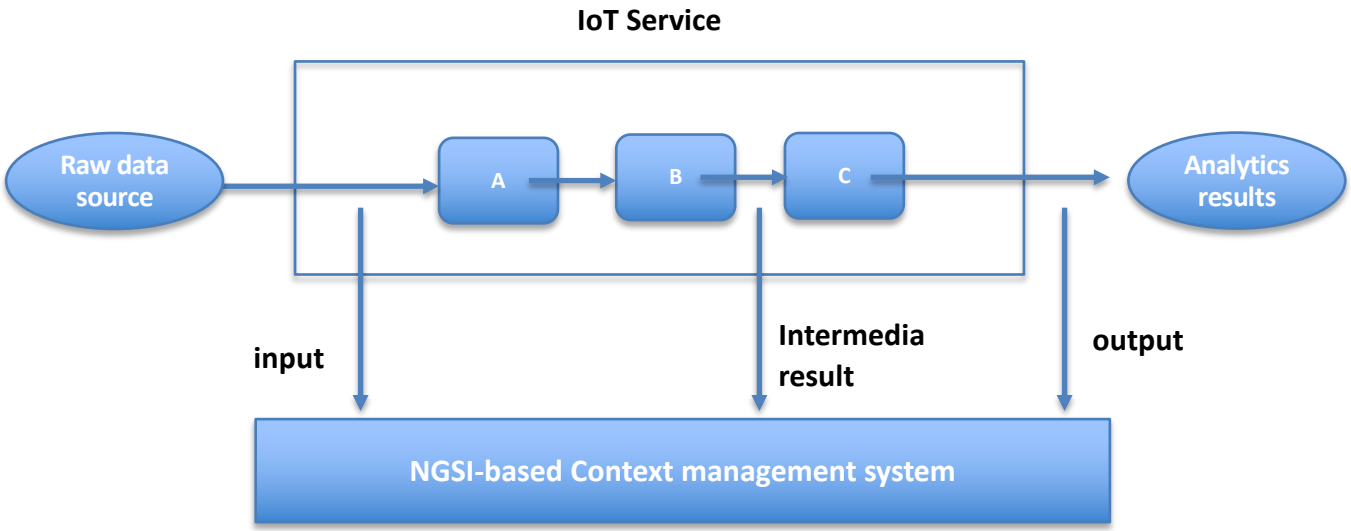
Graphical Editor to Programme IoT Services



Triggered by an “Intent”



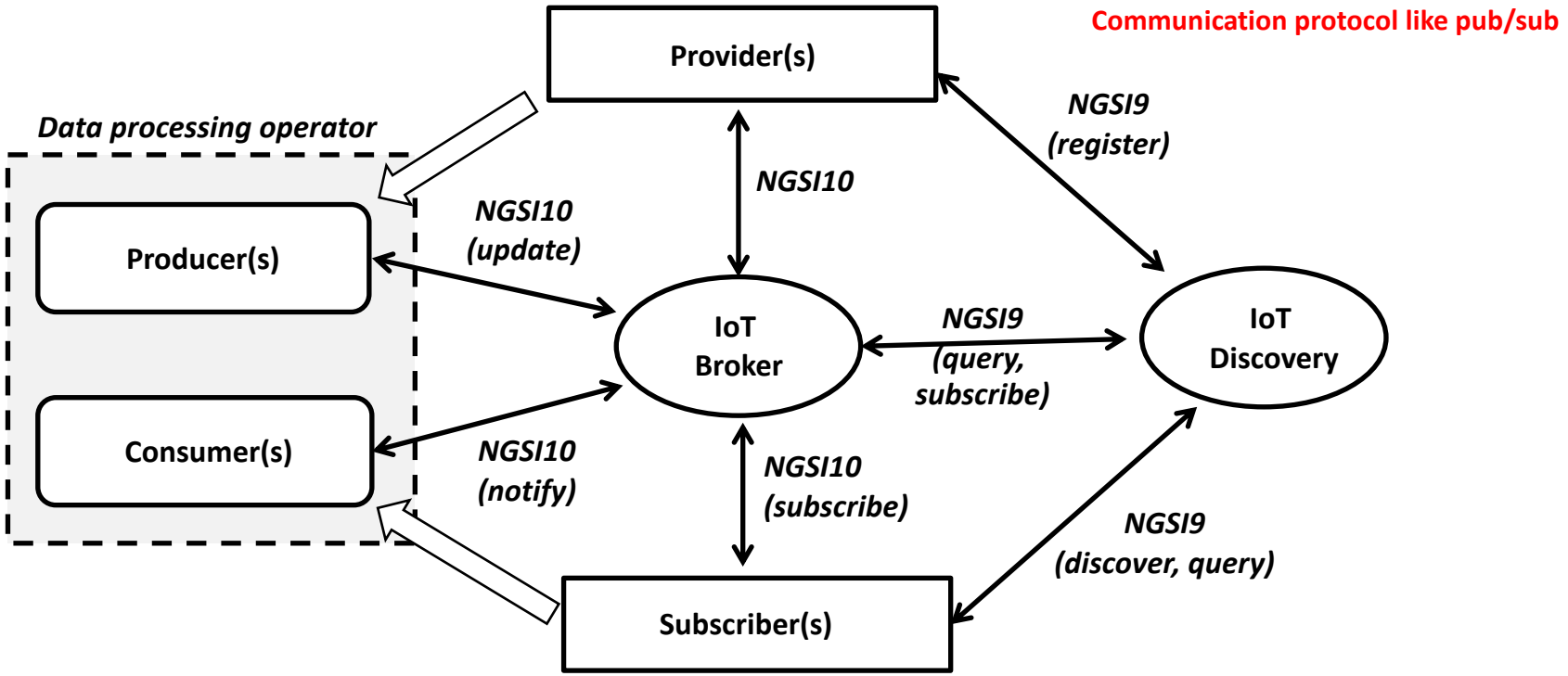
Context Information Management: Standard-based, NGSI



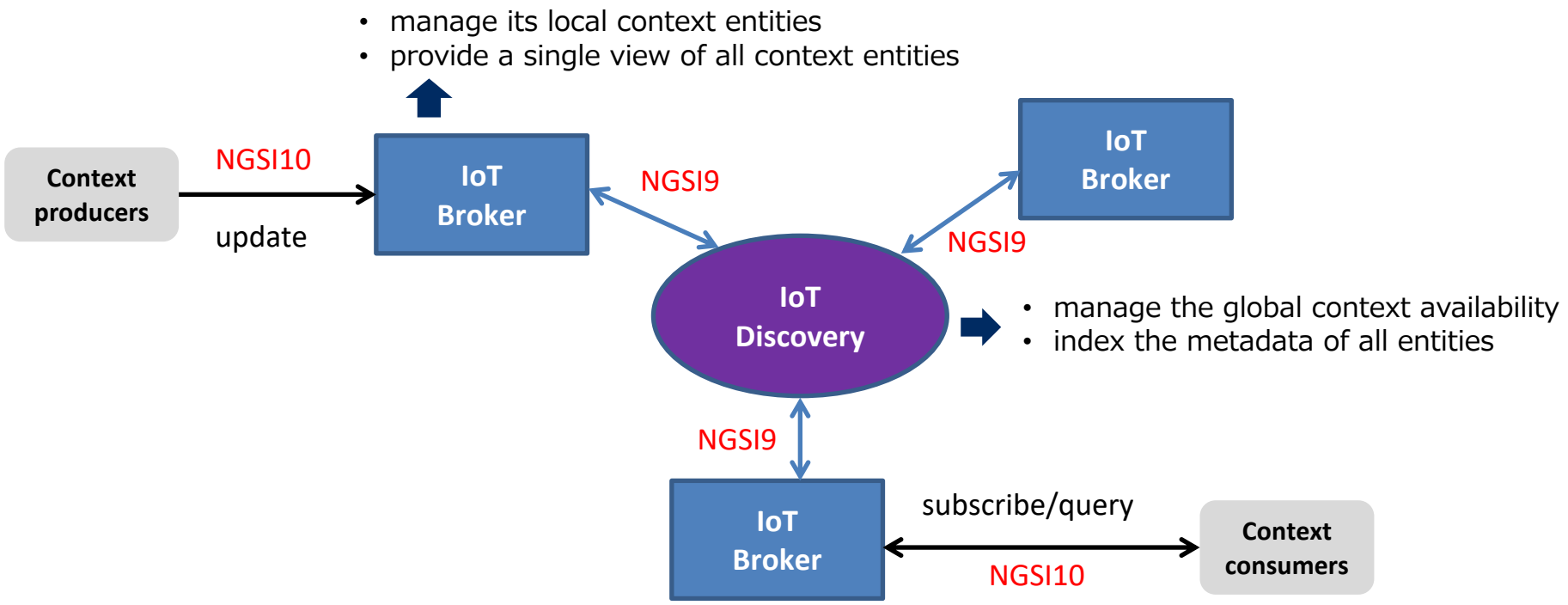
NGSI: Next Generation Service Interface

id	type	attribute	location	Other metadata

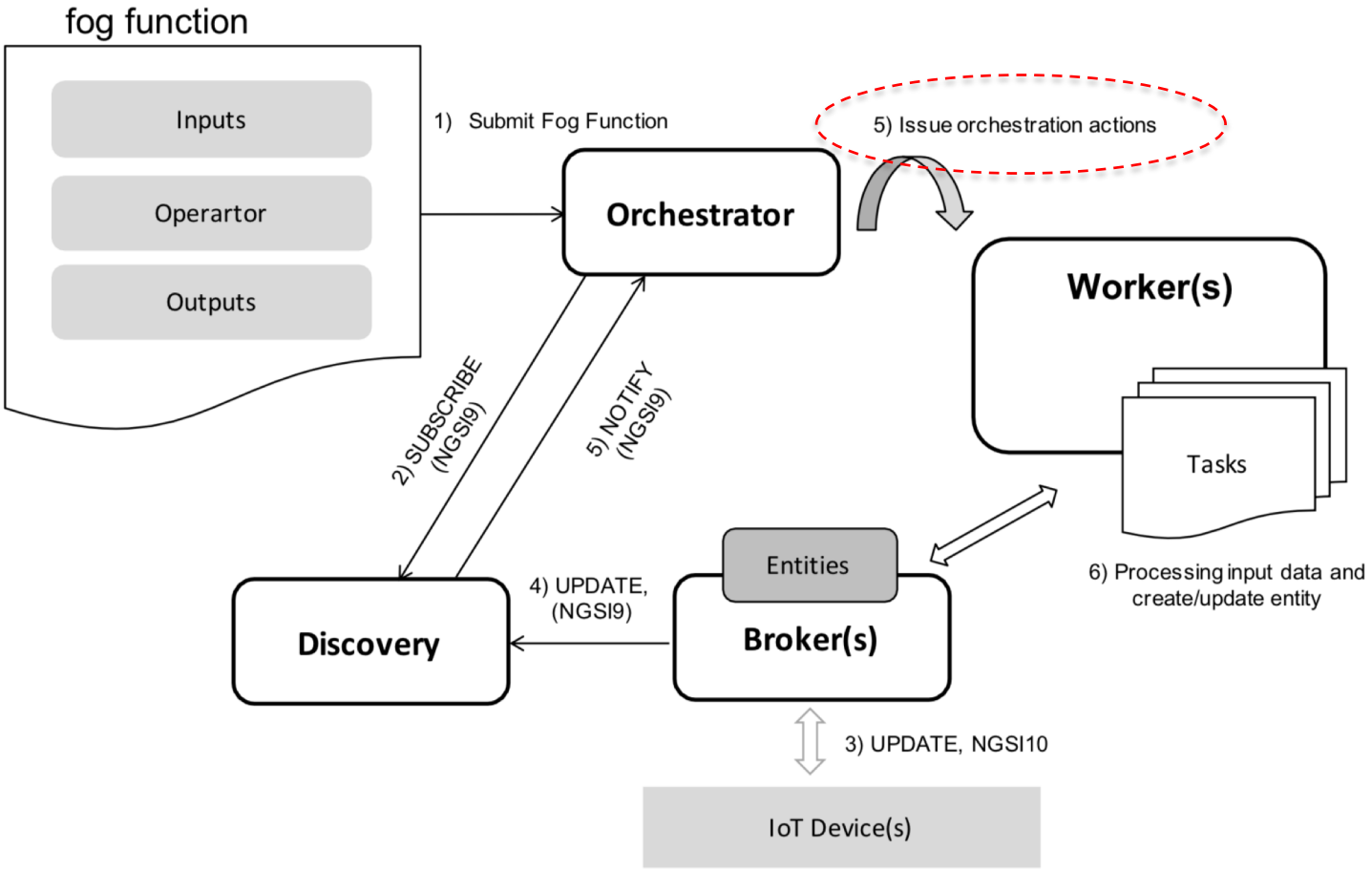
Data model for entities



Context Information Management: Two-layer & Distributed



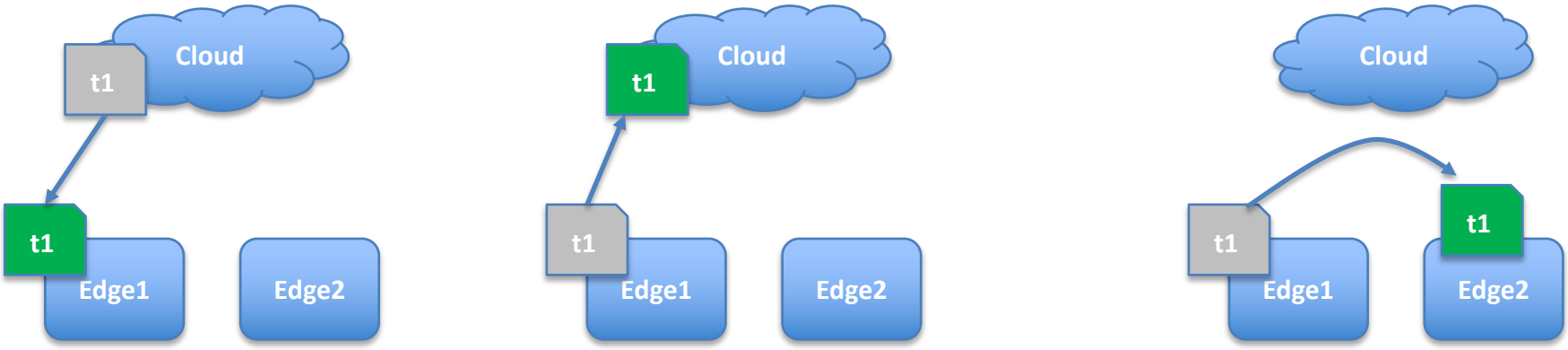
Data-driven Function Orchestration (1)



Data-driven Function Orchestration (2)

actions	What to do
ADD_TASK	To launch a new task with the given config- uration that includes the initial setting of its input streams
REMOVE_TASK	To terminate an existing running task with the given task ID
ADD_INPUT	To subscribe to a new input stream on behalf of a running task so that the new input stream can flow into the running task
REMOVE_INPUT	To unsubscribe from some existing input stream on behalf of a running task so that the task stops receiving entity updates from this input stream

Context-aware Task Migration



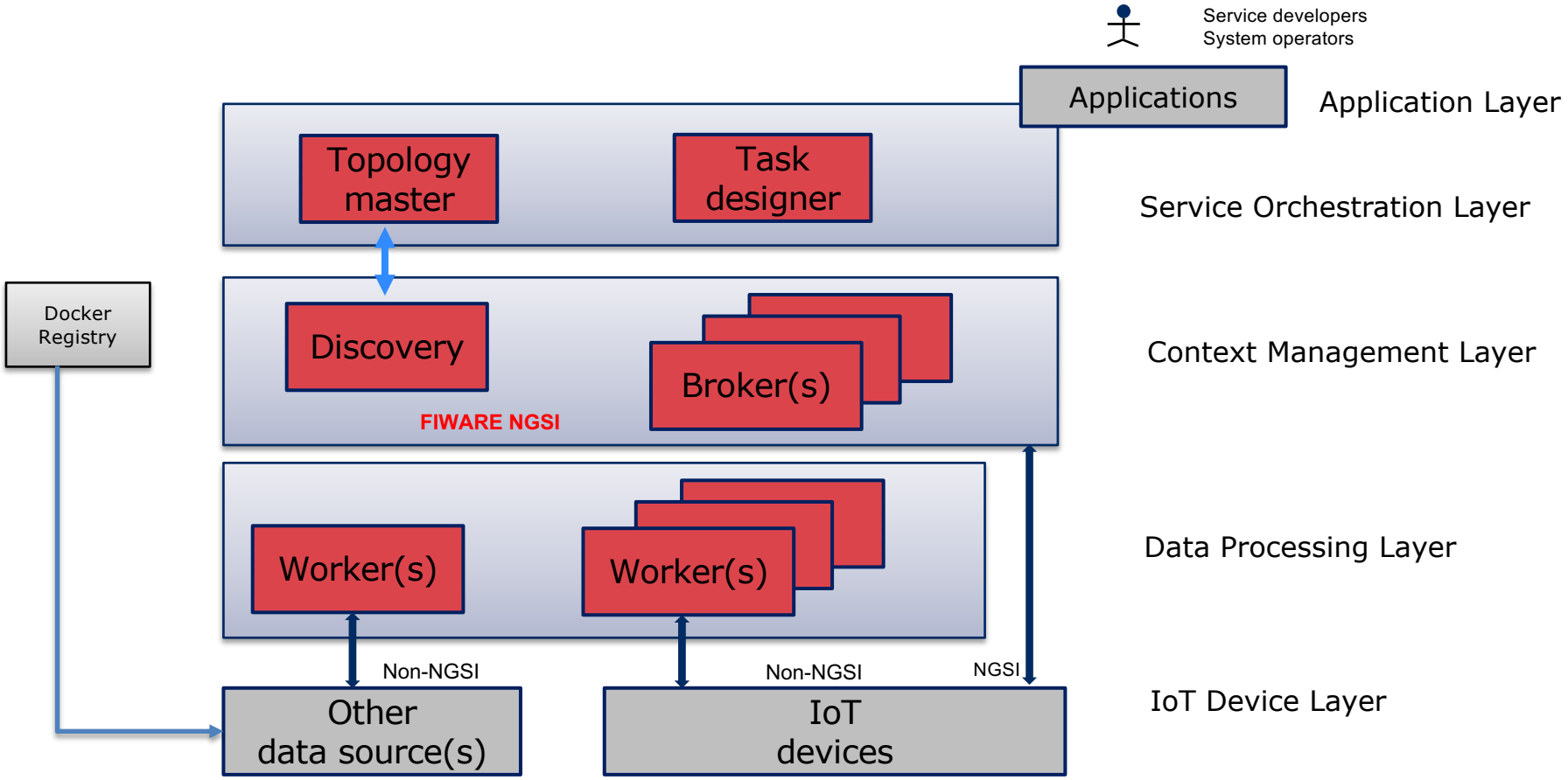
Cloud -> Edge

Edge -> Cloud

Edge -> Edge

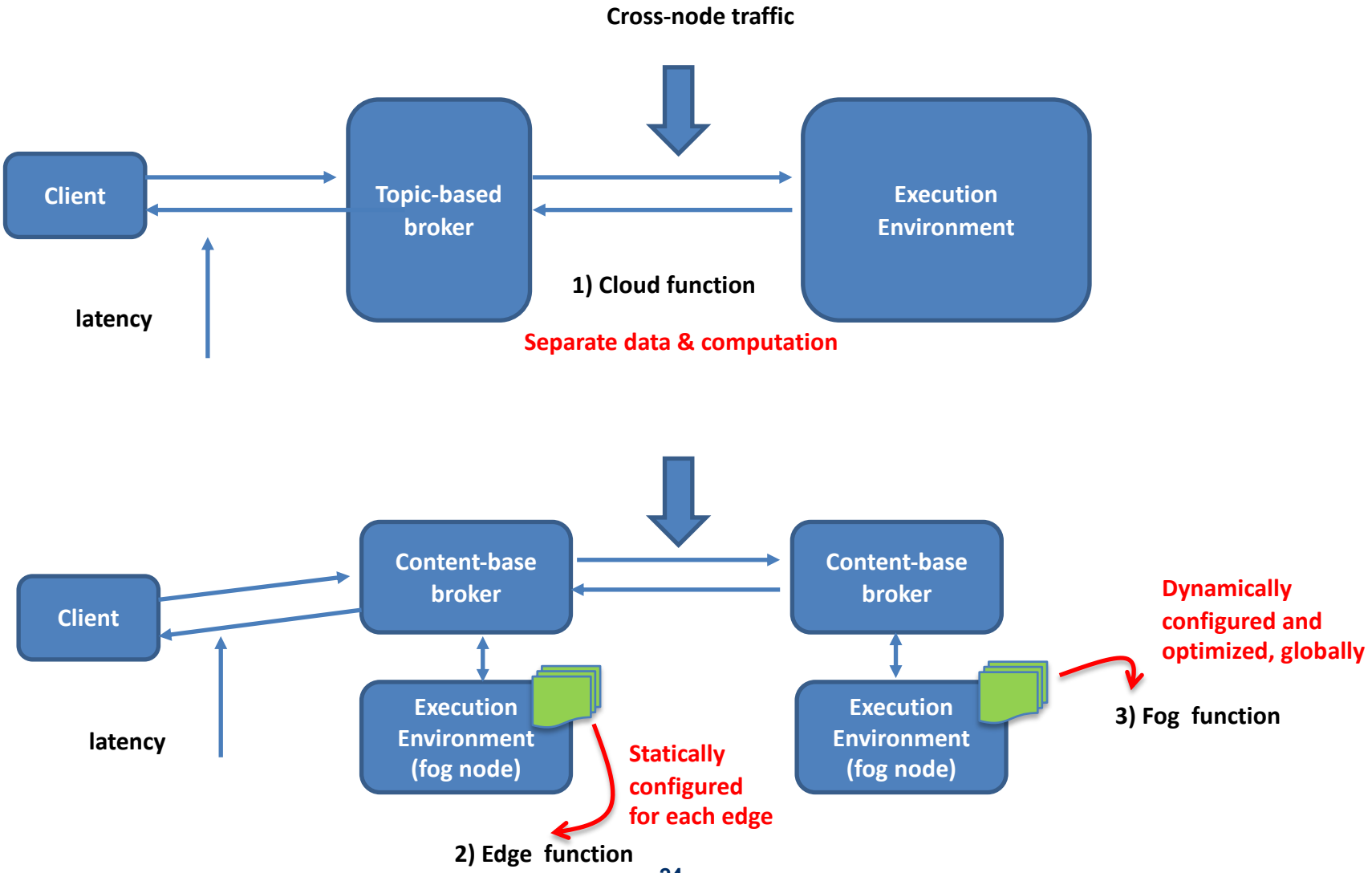
Three types of use cases for task migration

FogFlow System



Open source, available at github https://github.com/smartfog/fogflow

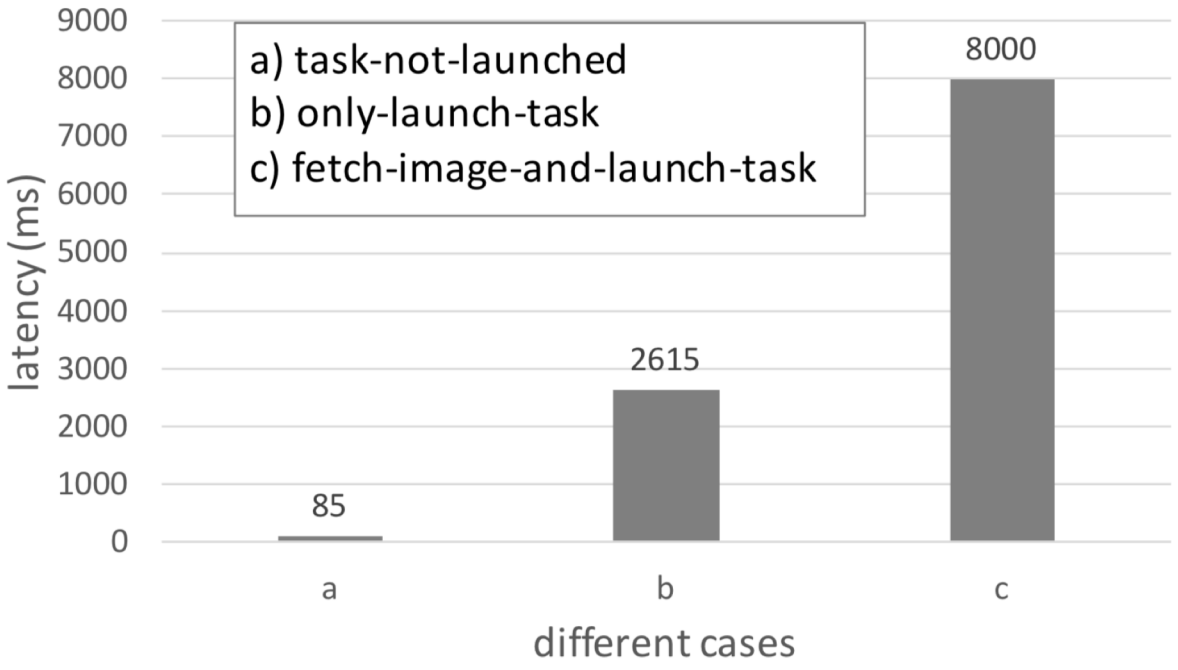
Comparison with Cloud Function and Edge Function



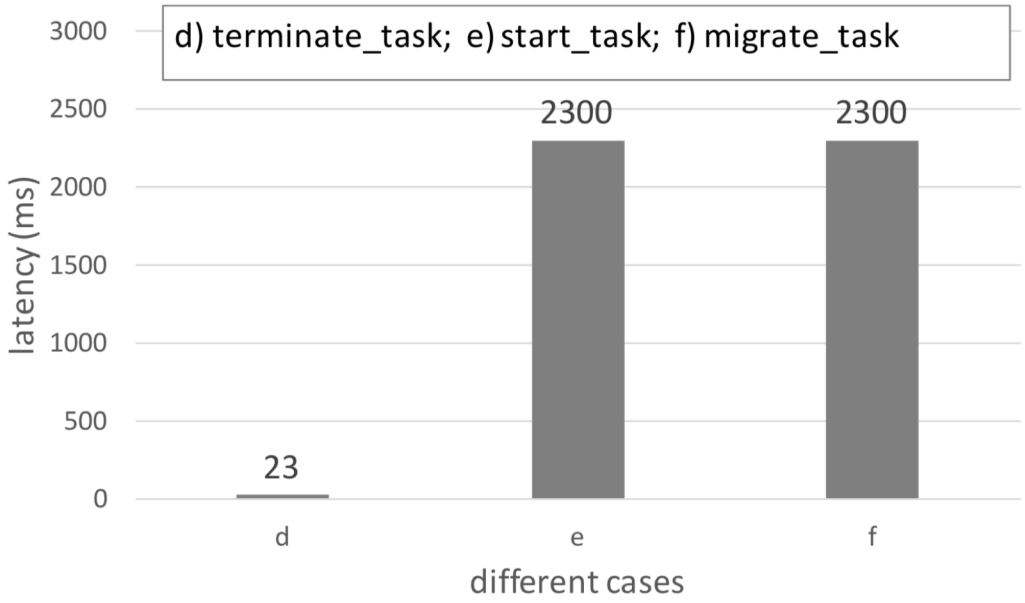
Performance Evaluation

- Latency
- Scalability
- Benefits as compared to the existing solutions

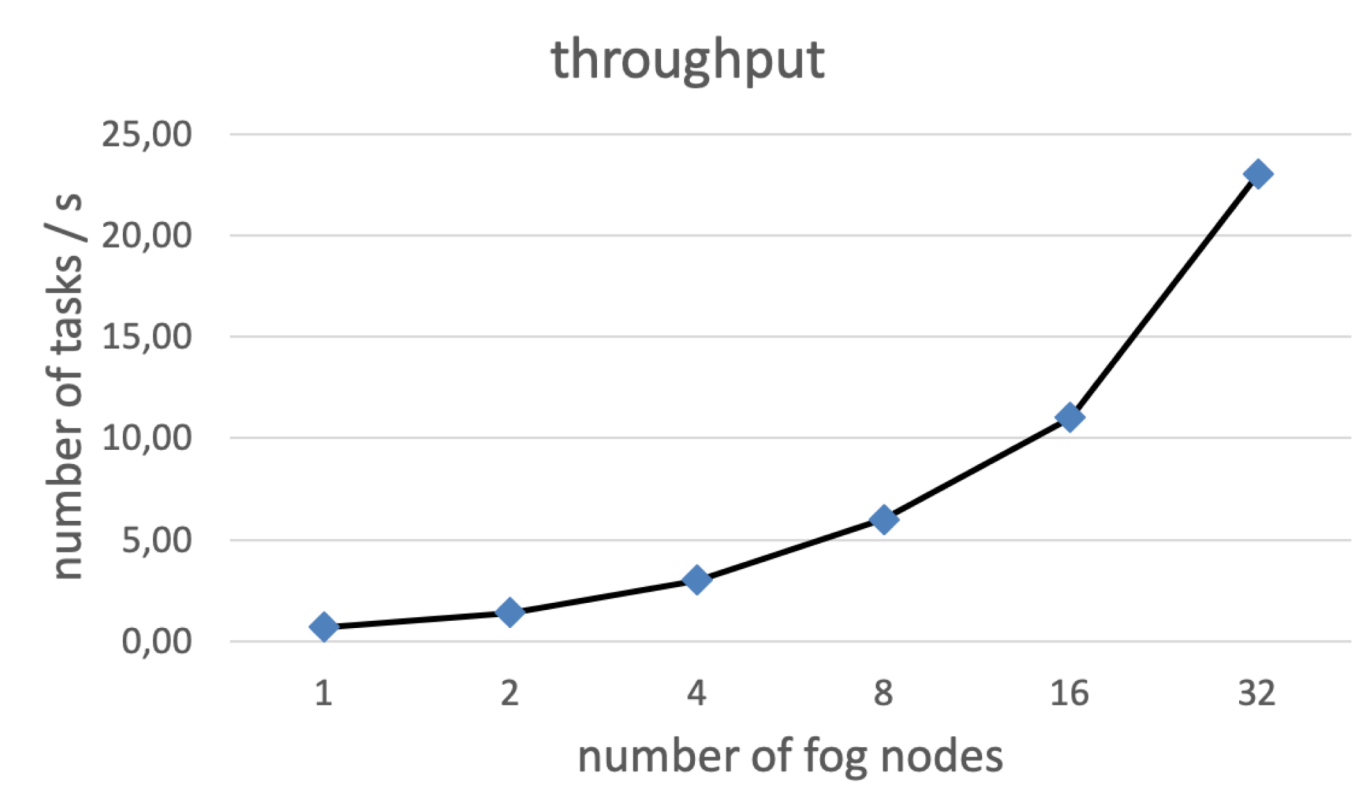
Evaluation Result (1): Startup Latency



Evaluation Result (2): Migration Latency

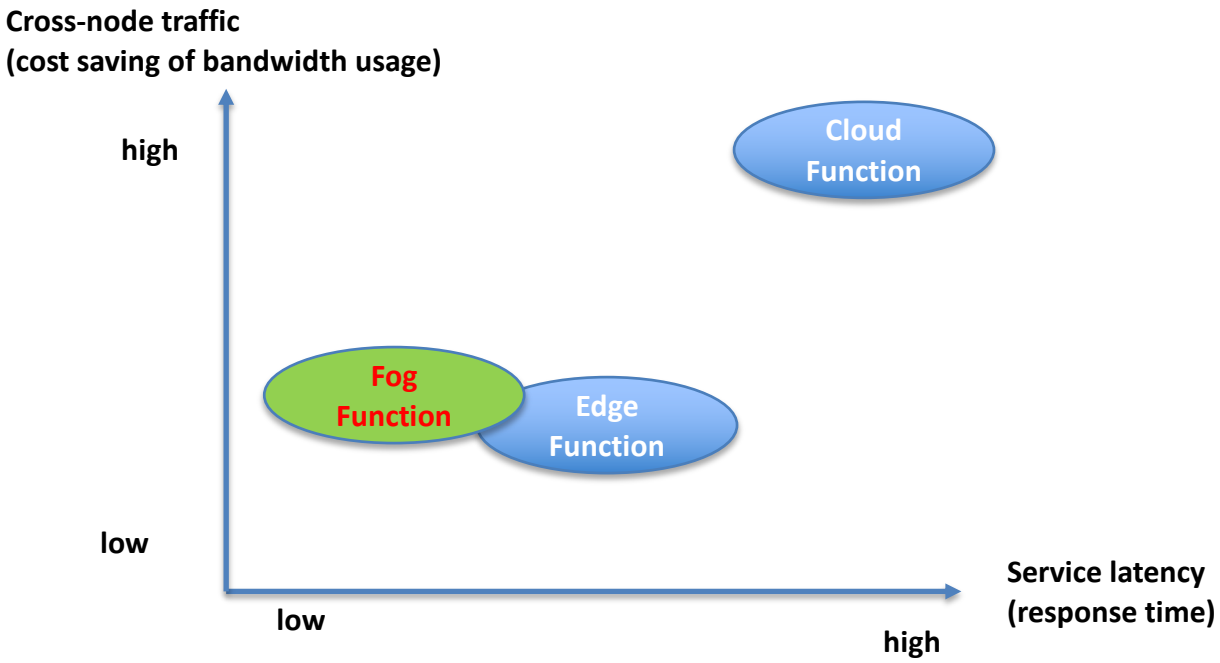


Evaluation Result (3): Scalability

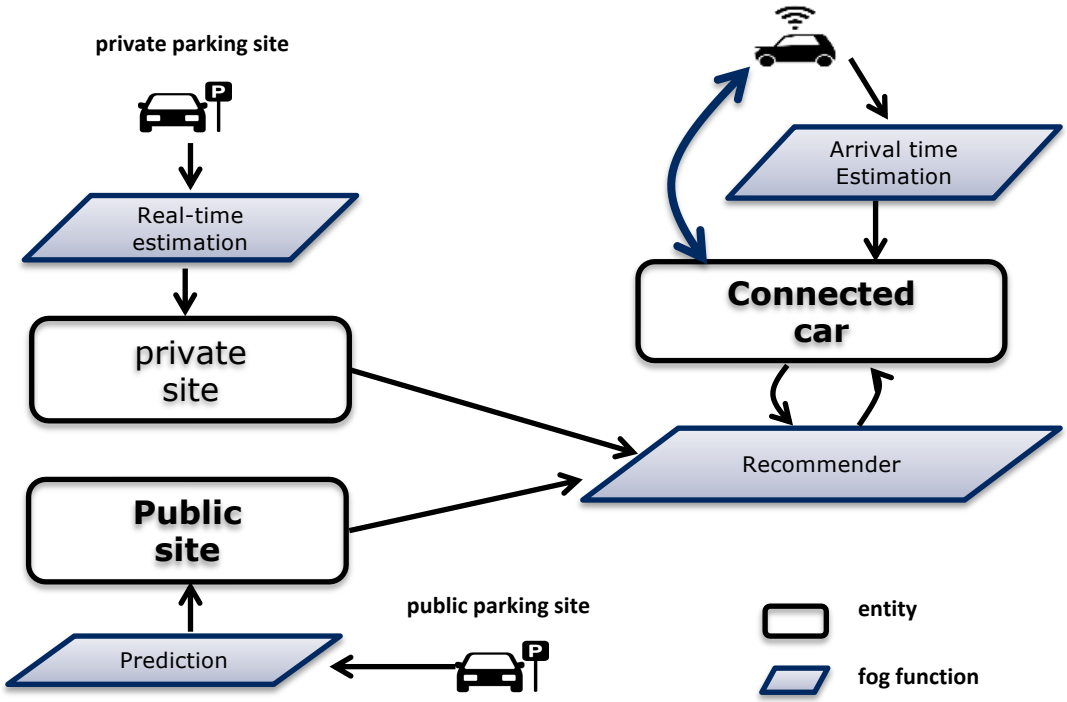


Comparison Result

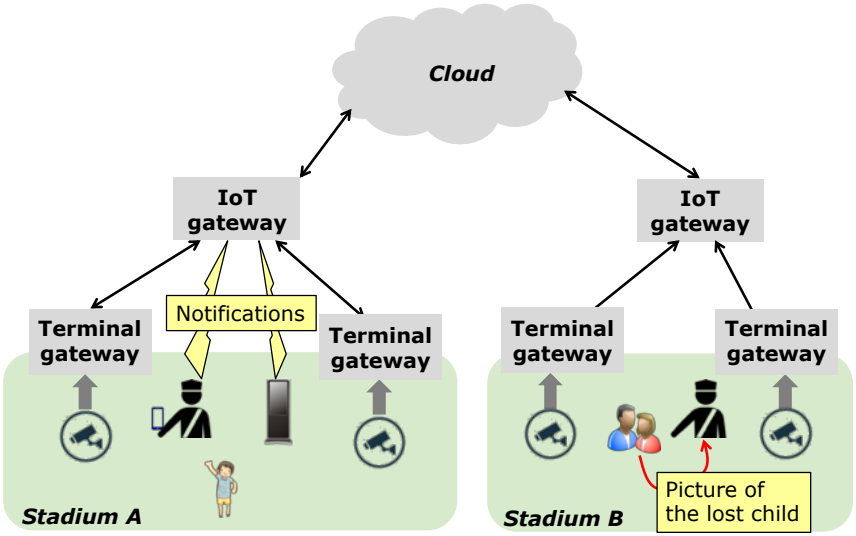
Approaches	cross-node traffic (MB)		avg. service latency (ms)	
	small	big	small	big
Cloud Function	86.6	987.3	262	610
Edge Function	3.5	10.8	68	150
Fog Function	3.8	11.4	59	102

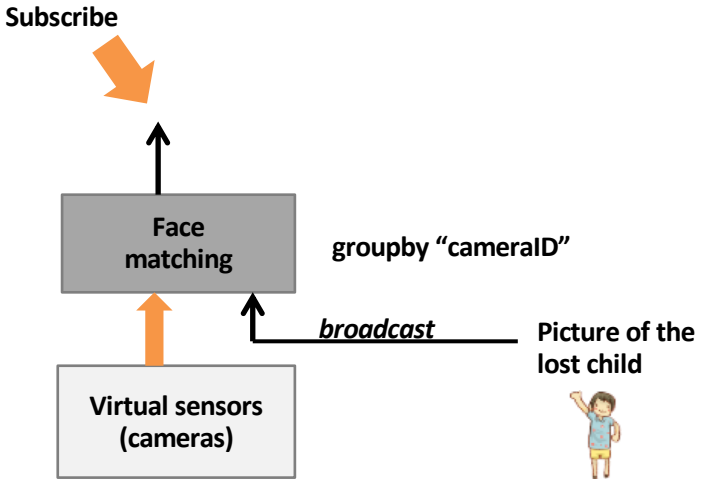


Use Case 1: Smart Parking



Use Case 2: Lost Child Finder





Conclusion and future work

- New *programming model*, namely *Fog Function*
  - Function-as-a-service
  - Hide all the details of how to manage the underlying infrastructure (cloud and edges)
  - *Data-centric*
- Efficient *fog computing framework, namely FogFlow*
  - *Utilize different context information*: data locality, available resource, usage context
- Various applications
  - Smart cities, smart industry, public safety

- Ongoing trend and future work: ICT infrastructure is becoming more distributed and complex, but for service developers and operators, ***the infrastructure must be transparent and intelligent***:
  - Controlling data processing flows dynamically to meet various requirements
  - Automatically matching data providers and situation consumers
  - Self-organized and optimized
  - Making its usage easier and easier

Thank you!

\Orchestrating a brighter world

NEC

BigDataStack

Holistic stack for big data applications and operations

The research leading to these results has received funding from the European Community's Horizon 2020 research and innovation programme under grant agreement nº 779747