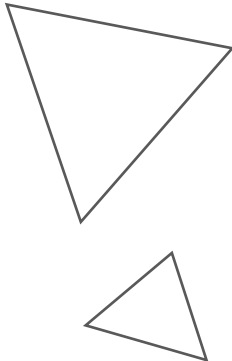




AI as a Service: Architecture Optimization and Development for Cybersecurity and Internet of Things

.....• **YING-DAR LIN**
YUAN-CHENG LAI

•.....

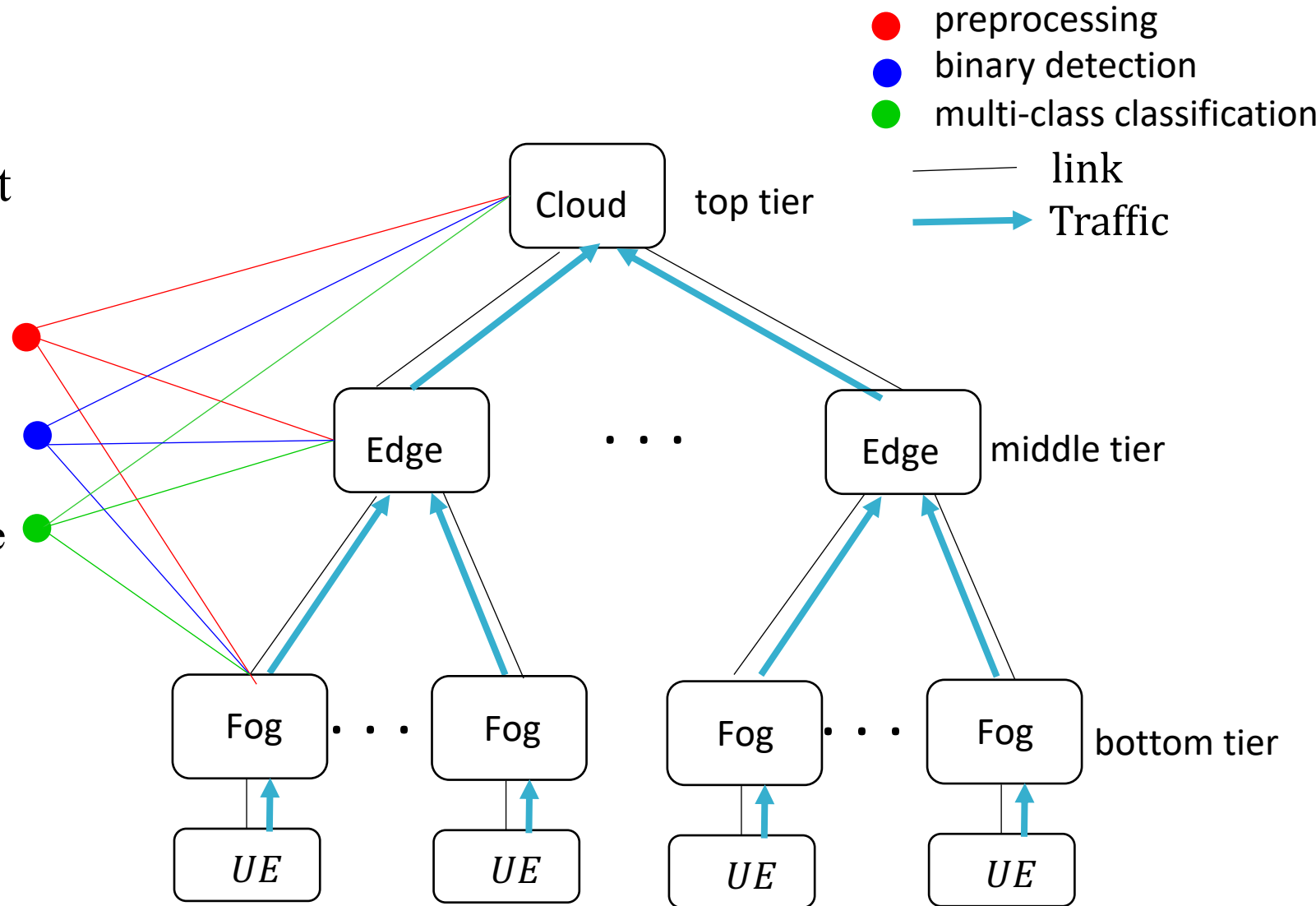


Outline

- Motivation
- Background
- Research Issues
 - IDaS
 - IoTaS
- Development Issues
- Survey
 - Survey on IDaS
 - Survey on AIoT
- Roadmap
 - Research topics
 - IDaS topics
 - IoTaS topics
 - Development topics
- Problem statements
 - Research track
 - IDaS
 - IoTaS
 - Development track
 - Service deployment
 - AISF for IDaS
 - AISF for IoTaS
- Solutions
 - Research track
 - IDaS
 - IoTaS
 - Development track
 - AISF
 - AISF for IDaS
 - AISF for IoTaS
- Implementation
- Evaluation
- References

Motivation

- AI as a Service from service provider perspective
- Utilize distributed architecture that can be provided by different configurations
 - Federation among fog, edge, cloud
 - Mapping tasks to servers
 - Resource allocation in each servers
 - Auto-deployment and reconfigurable system for services and network architectures



Background

AI as a Service (AIaaS)

- The global market of AI continues to grow
 - 90 percent of new enterprise apps will embed AI by 2025
- Challenges for enterprises in integrating and monitoring the AI system
 - Complex and demanding process
 - A lack of abilities to set up and maintain
 - Limited knowledge to deploy and configure
- Enterprises outsource their AI-based system to service providers (SP)
- SP can offer AIaaS with on-demand, portable, and pay-per-use cost model
 - Handle the process of developing, deploying, maintaining, updating the system
- Factors need to be considered in offering AIaaS
 - Architecture
 - Algorithm

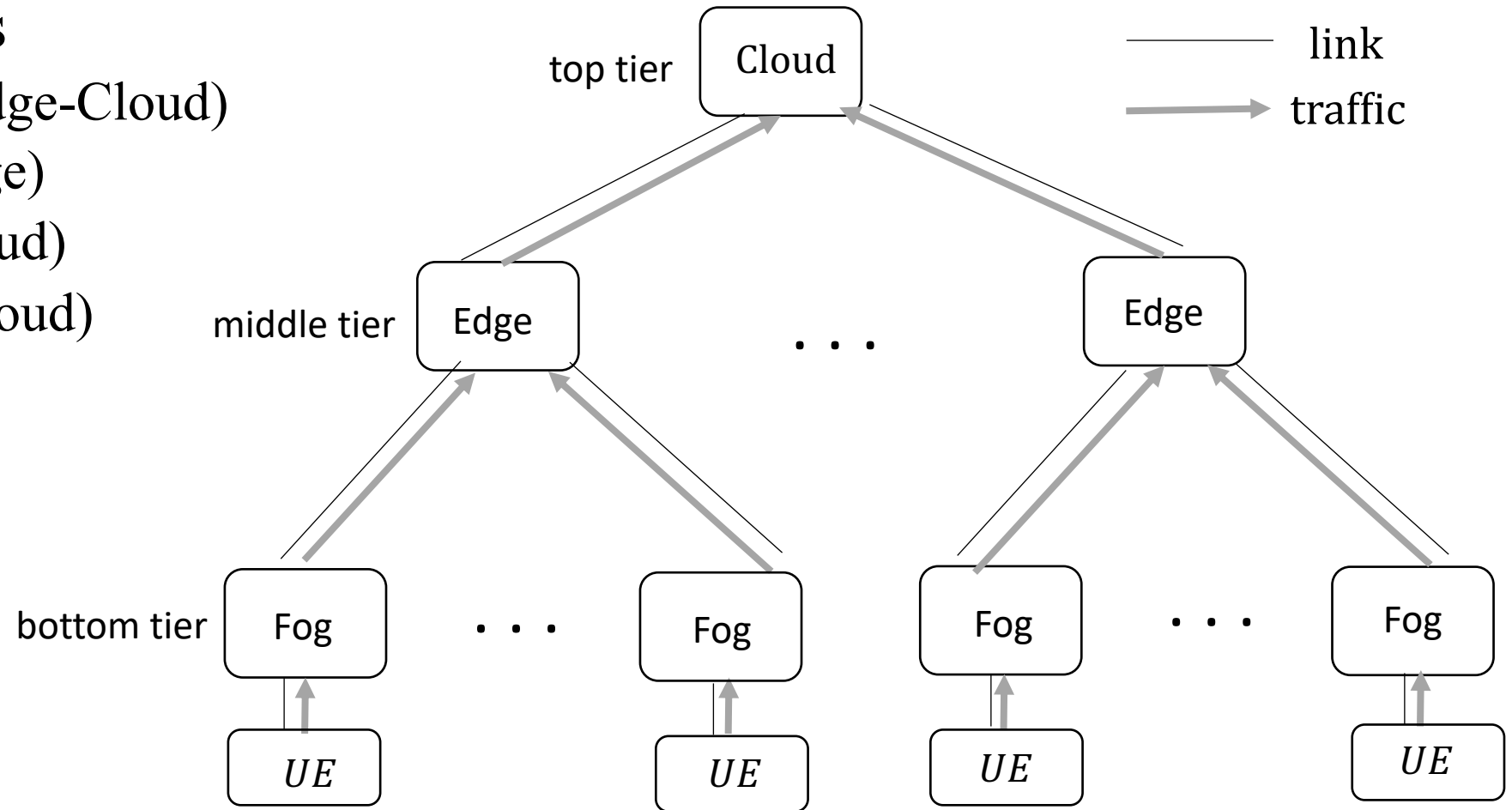
Background

Three-Tier Network Structure

Example: Fog-Edge-Cloud

- Seven architectures

- Three-tier (Fog-Edge-Cloud)
- Two-tier (Fog-Edge)
- Two-tier (Fog-Cloud)
- Two-tier (Edge-Cloud)
- One-tier (Fog)
- One-tier (Edge)
- One-tier (Cloud)



Three-tier (Fog-Edge-Cloud)

Background

AIaaS - Services

- Intrusion detection as a service (IDaaS)
 - Provide the machine learning based intrusion detection system
- IoT as a service
 - Learning and inference with machine learning for IoT services

Background

Machine Learning

□ Learning and Testing

□ Learning

- Centralized Learning
 - One location
- Distributed Learning
 - Many locations
 - Load reducing
 - Model parallelism
- Federated Learning
 - Many locations + interaction
 - Privacy
 - Data parallelism
 - Local model and global model

Background

Intrusion Detection System (IDS) (1/2)

- IDS types:
 - Signature based
 - Often fails to detect unknown new attacks
 - Anomaly based
 - High false positive rates
 - Machine learning for address above problems
 - Achieve a satisfactory level of detection
 - Ability to detect more attack variants
- Intrusion Detection as a Service (IDaS)
 - Time consuming and costly to monitor the IDS in 24/7
 - Enterprises outsource their IDS system to service providers (SP)
 - Architecture and algorithm are the factors to consider in offering IDaS

Background

Intrusion Detection System (IDS) (2/2)

- Intrusion detection: architectures

- Centralized: bottleneck
- Distributed
 - Full job
 - Partial job
 - Benefits:
 - Reduce the traffic volume
 - More scalable
 - Enhance the IDS performance

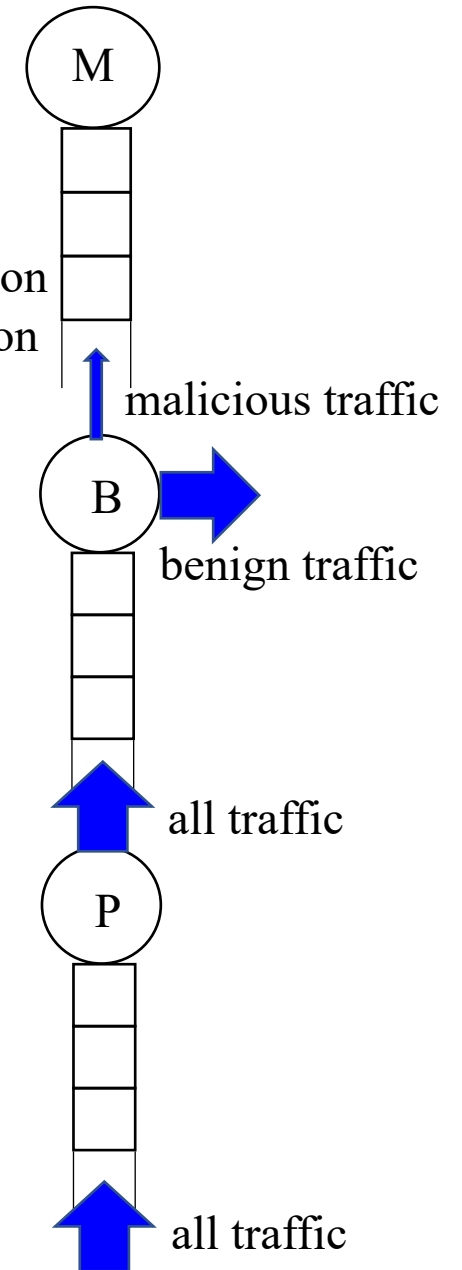
- 2-stage tasks (binary classification + multi-classification) [1]

- Better accuracy

- 3-stage tasks

- Preprocessing (all traffic)
- Binary classification (all traffic)
- Multi-classification (malicious traffic)

P: Preprocessing
B: Binary classification
M: Multi-classification



Background

Mapping Tasks to Servers (Combinations) in IDaS

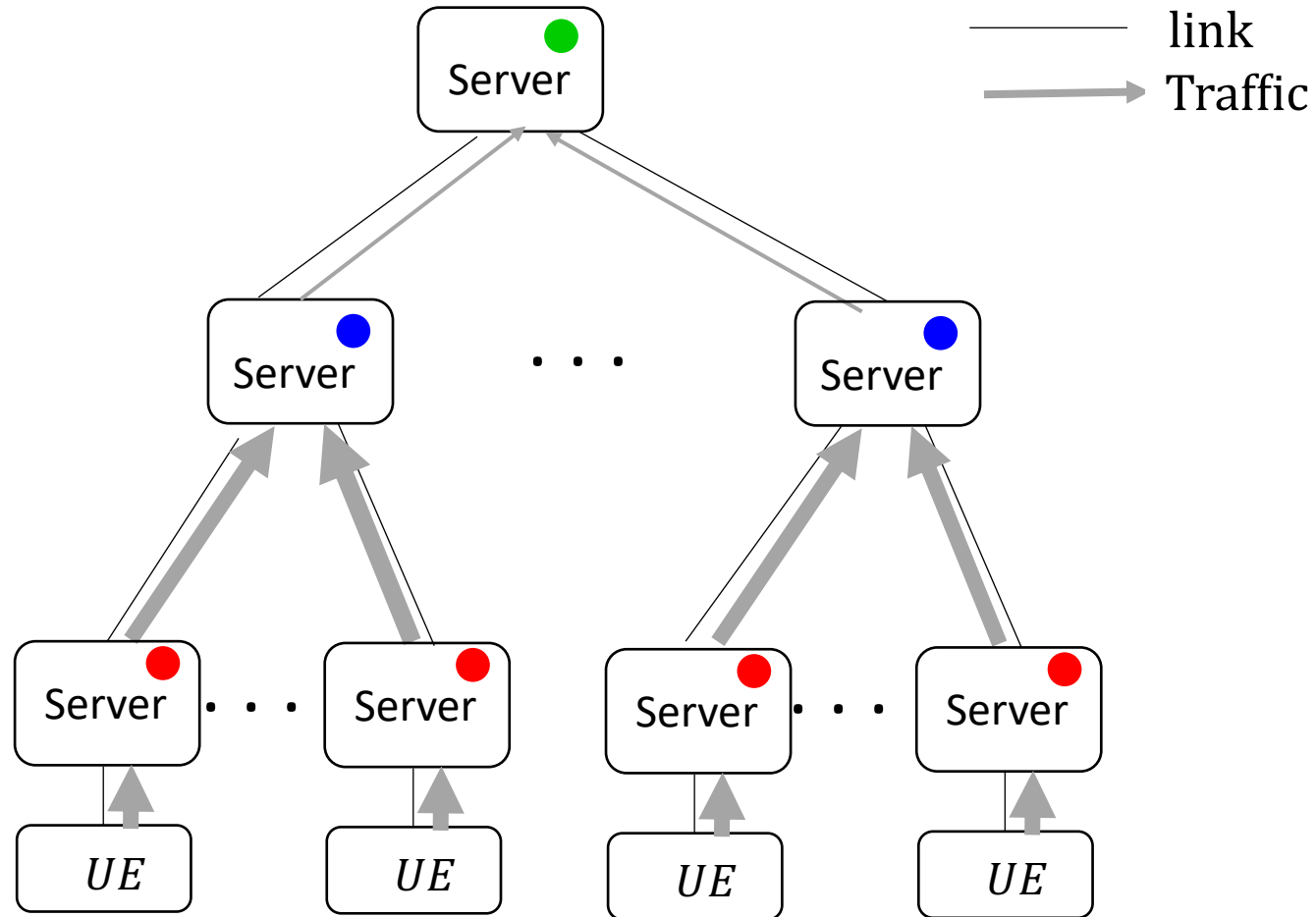
# of tiers	Architecture	Fog	Edge	Cloud	Traffic Path ID	Abbreviation
1	Fog	p, b, m			1	pbm/-/-
	Edge		p, b, m		2	-/pbm/-
	Cloud			p, b, m	3	-/-/pbm
2	Fog-Edge	p, b	m		4	pb/m/-
	Fog-Cloud	p, b		m	5	pb/-/m
	Edge-Cloud		p, b	m	6	-/pb/m
	Fog-Edge	p	b, m		7	p/bm/-
	Fog-Cloud	p		b, m	8	p/-/bm
	Edge-Cloud		p	b, m	9	-/p/bm
3	Fog-Edge-Cloud	p	b	m	10	p/b/m

Background

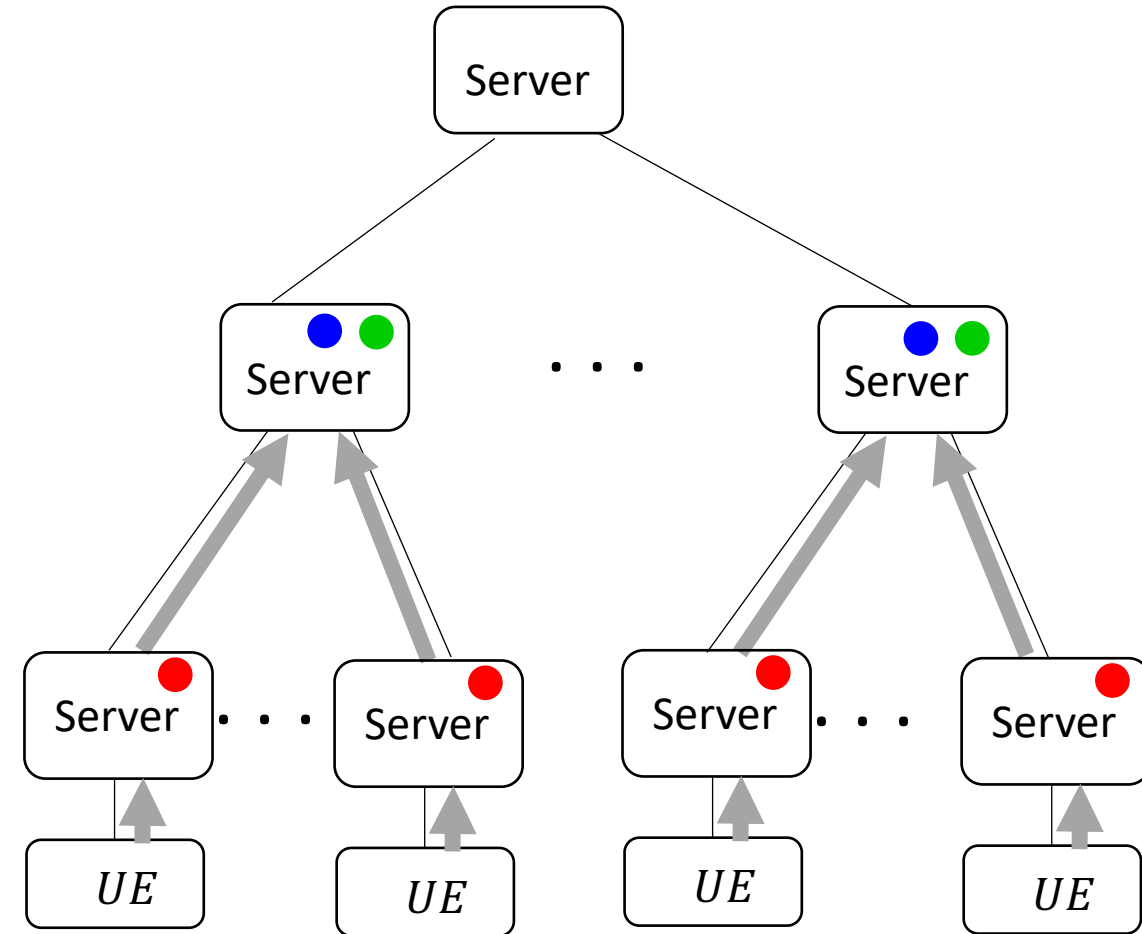
Combinations: Examples

- preprocessing
- binary classification
- multi-classification

Combination 10



Combination 7



Background

IoT as a Service (1/2)

❑ IoT

- Less capacity and less memory
- Heavy ML tasks
- Offloading ML

❑ Service provider's view

❑ IoT as a service: tenant

❑ Different dimensions

- Data
 - Big: All tenants' data
 - Small: one tenant's data
- ML model
 - Public: only one model
 - Private: one model for each tenant

Background

IoT as a Service – Task Assignment

- Small data – Private model
 - Centralized learning
 - Train only with a single tenant data
 - Federated learning
 - Global model for a single tenant aggregated from local tenant models
- Big data – Public model
 - Centralized learning
 - Train by using all tenants data
 - Federated learning
 - Global model from aggregating all local models from all tenants
- Small data – Public model
 - Federated learning
 - Each tenant will generate a global model
 - Then will be aggregated with all tenants' global models.

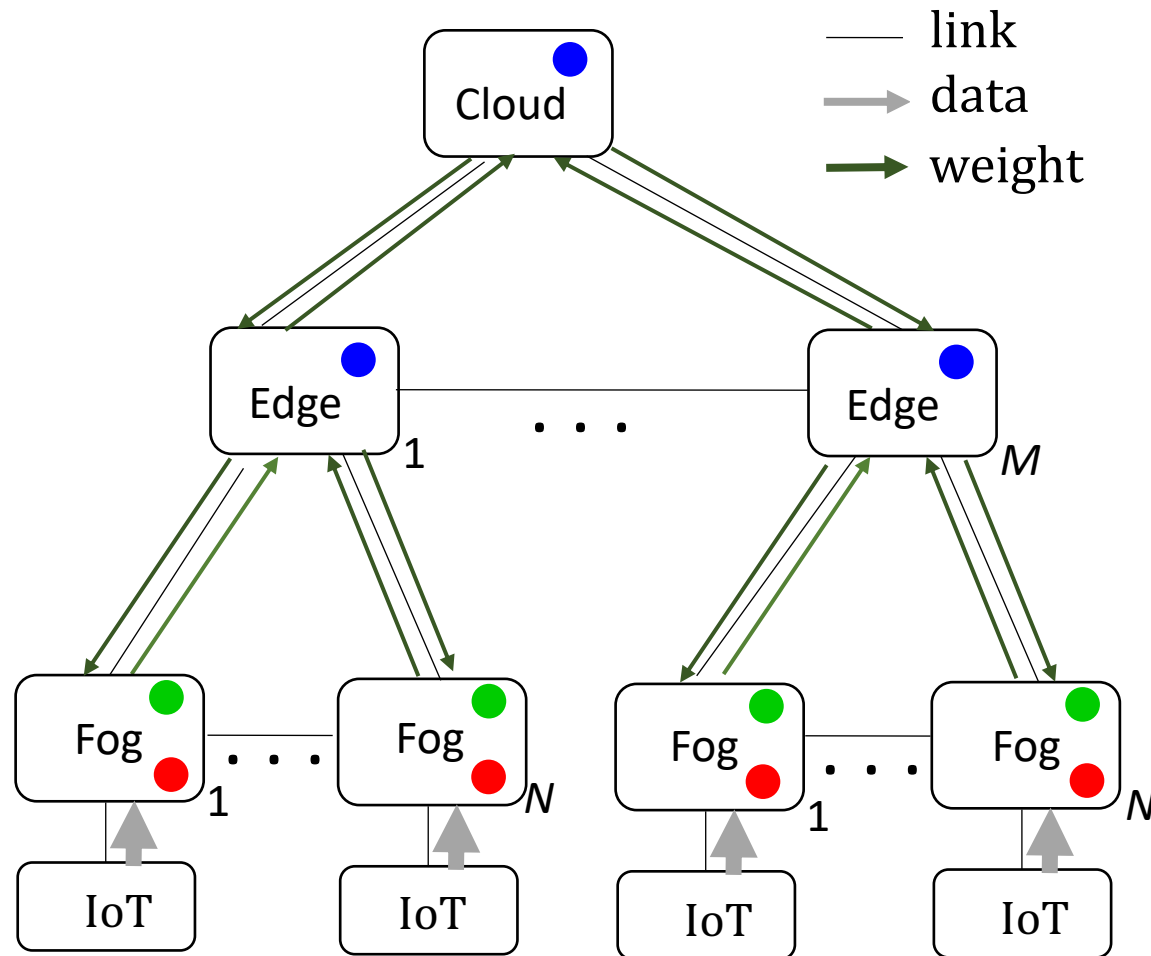
Background

IoTaS Task Assignment example

- federation
- training
- inference

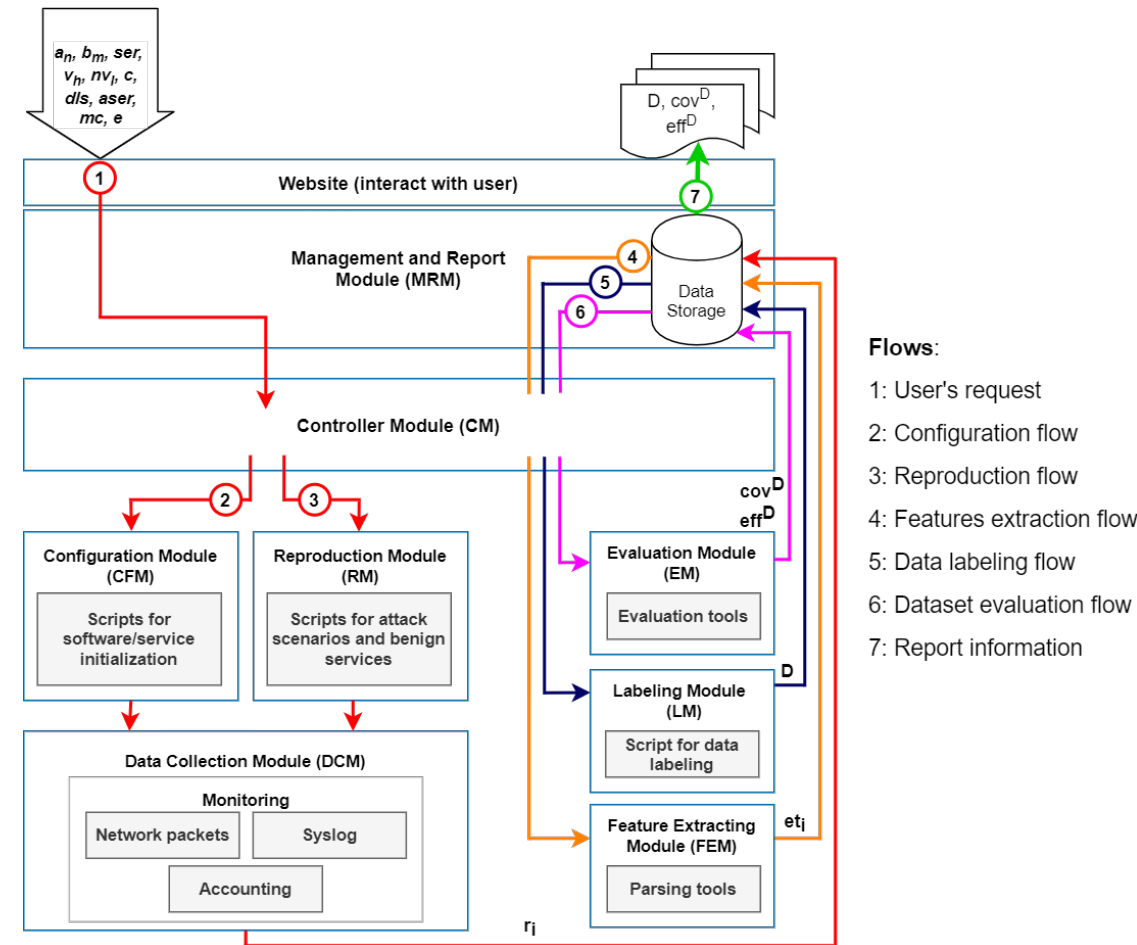
Small data – Private model

- Learning and inference at Fog
- 1st aggregation in Edge to aggregate local models from Fog
- 2nd aggregation in Cloud to aggregate models from Edge



Background CREME

- A toolchain of automatic dataset collection
 - Attack and benign behavior reproduction
 - Data collection
 - Feature extraction
 - Data labeling
 - Evaluation
- Collect and generate a dataset from multiple sources
 - Accounting
 - Network traffic
 - System logs
- Re-architect CREME as a traffic control in AI Service Framework

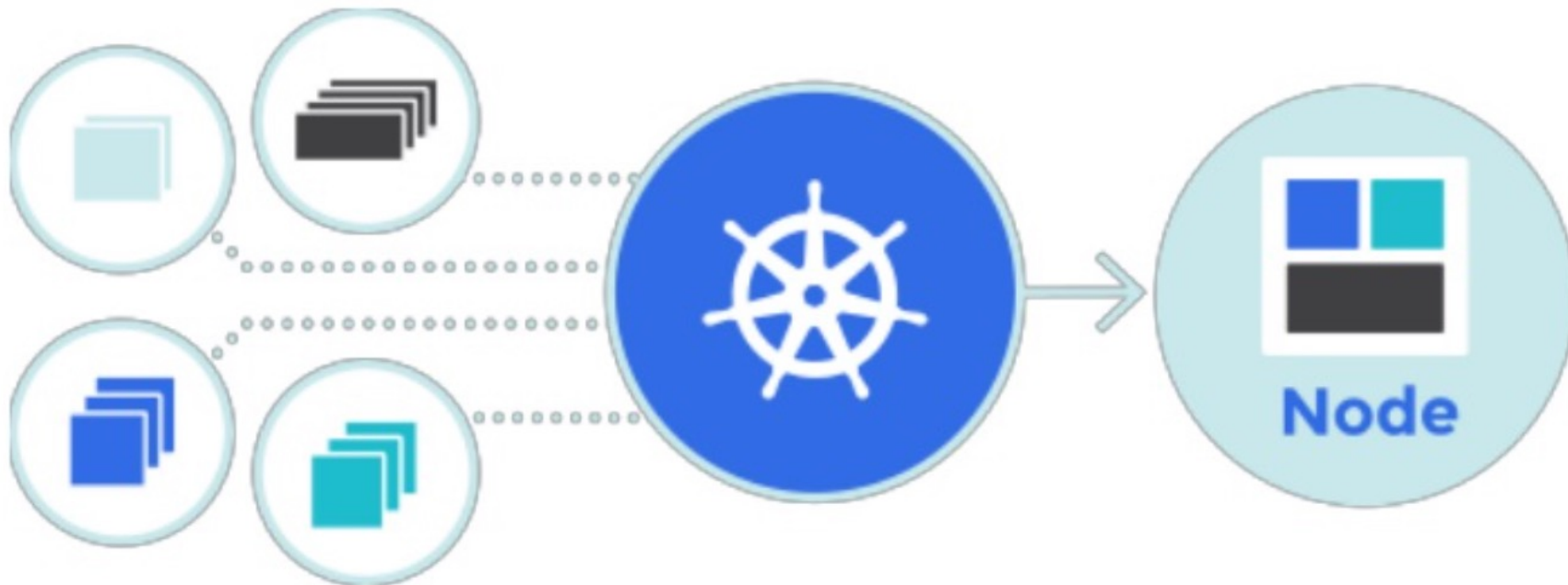


Background Kubernetes



Kubernetes is an open-source **Container Management** tool which automates *container deployment, container (de)scaling & container load balancing*.

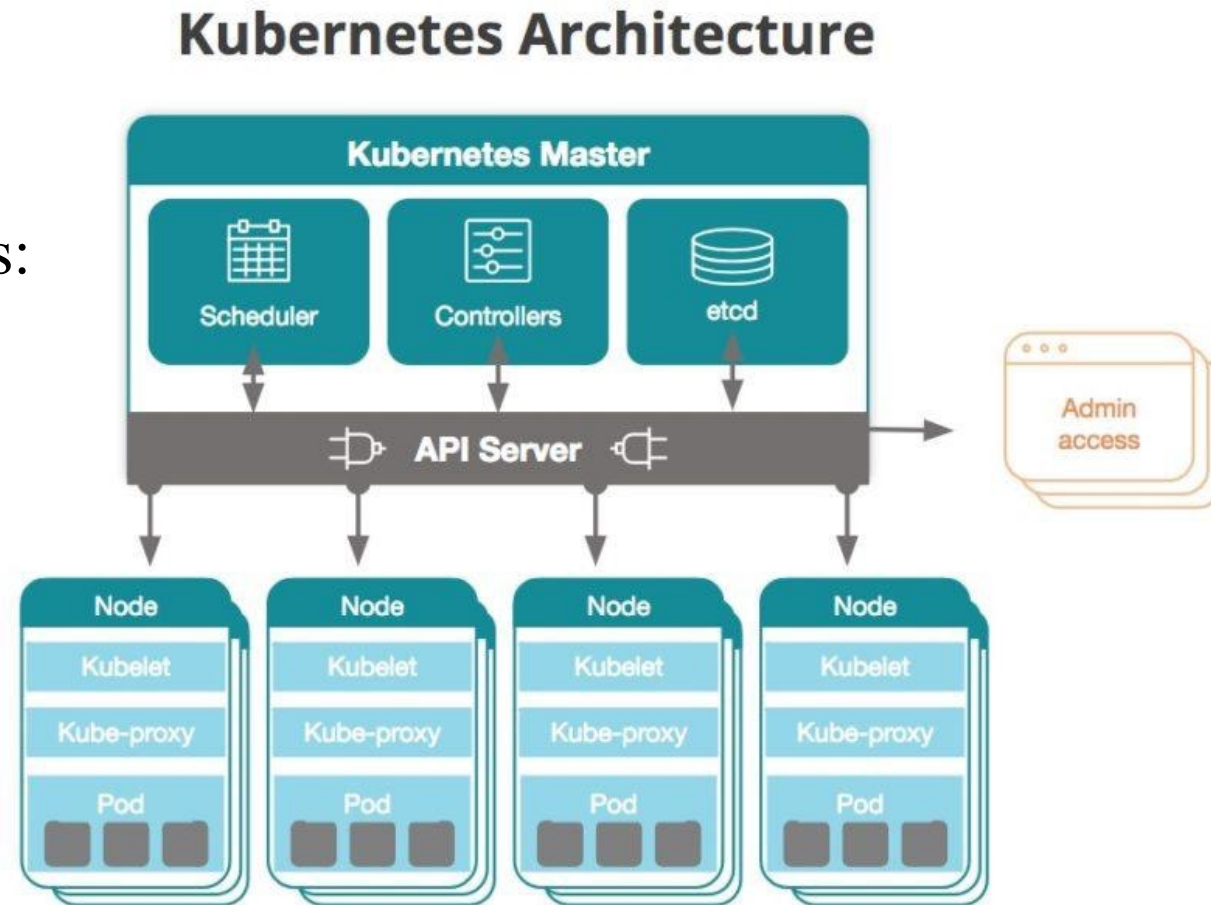
*Benefit: Works brilliantly with all cloud vendors: **Public, Hybrid & On-Premises**.*



Background

Kubernetes – Master node (1/2)

- Kubernetes has the following main components:
 - Master node
 - Worker nodes
 - Distributed key-value store, like etcd.
- A master node has the following components:
 - API server
 - front-end of the control plane
 - validates requests to update objects and acts as the unified interface
 - etcd.
 - a database Kubernetes uses to back-up all cluster data
 - Scheduler
 - watches for new requests coming from the API Server
 - assigns pod to nodes
 - Controller manager
 - responsible for registering a node and monitoring its health throughout its lifecycle



Proposal Tracks

□ Research track

- Focus in architecture optimization for IDaS and IoTaS
- To get the basic results in:
 - Architecture
 - Task mapping
 - Capacity allocation

□ Development track

- Develop the prototype of an AISF system
 - With auto-deployment, re-configurability, auto-scaling, and auto-offloading
 - Integrate IDaS and IoTaS to AISF

Roadmap

Development Track

Year	Topic	Objective	Auto-deployment			Master Proxy	Server Proxy	Traffic Generator
			Network Architecture	Service Mapping	Capacity Allocation			
1	Service deployment with AISF	Generic Service Framework	1/2/3-tier	<ul style="list-style-type: none"> • Preprocessing • Training-centralized • Training-local • Training-global • Detection-1st line • Detection-2nd line 	6 service elements	Scaling & offloading for all service elements	Handling all types of traffic for offloading	N-types of traffic
2	IDaS on AISF	Offer IDaS by AISF	1/2/3-tier	<ul style="list-style-type: none"> • Preprocessing • Detection-Binary • Detection-Multiclass 	3 service elements	Scaling & offloading for 3 service elements	Handling raw traffic for offloading	2-type of traffic (attack & benign)
3	IoTaS on AISF	Offer IoTaS by AISF	1/2/3-tier	<ul style="list-style-type: none"> • Training-centralized • Training-local • Training-global • Detection-1st line 	4 service elements	Scaling & offloading for detection-1 st line	Handling IoT apps traffic for offloading	N-types of traffic

Development Track Problem Statement

Problem Statement – Service Deployment

□ Given:

- Network topology: 1-tier, 2-tier, or 3-tier
- Service mapping
- Arrival traffic rate for the service
- Service level agreement

□ Output:

- Configurations of network architecture
- Configurations of service mapping
- Resource allocations for the service

□ Objective:

- Deploy the services to the nodes automatically
- Orchestrating the resources and operational management

Problem Statement – AISF for IDaS

□ Given:

- Network topology: 1-tier, 2-tier, or 3-tier
- 10 IDaS services mapping
- Benign and attack traffic
- Delay threshold for IDaS

□ Output:

- Configurations of network architecture
- Configurations of 10 IDaS services mapping
- Resource allocations for the IDaS services mapping

□ Objective:

- Deploy the IDaS services to the nodes automatically
- Orchestrating the resources and operational management for IDaS

Problem Statement – AISF for IoTaS

□ Given:

- Network topology: 1-tier, 2-tier, or 3-tier
- AIoT services mapping
- IoT traffic
- Delay threshold for inference task

□ Output:

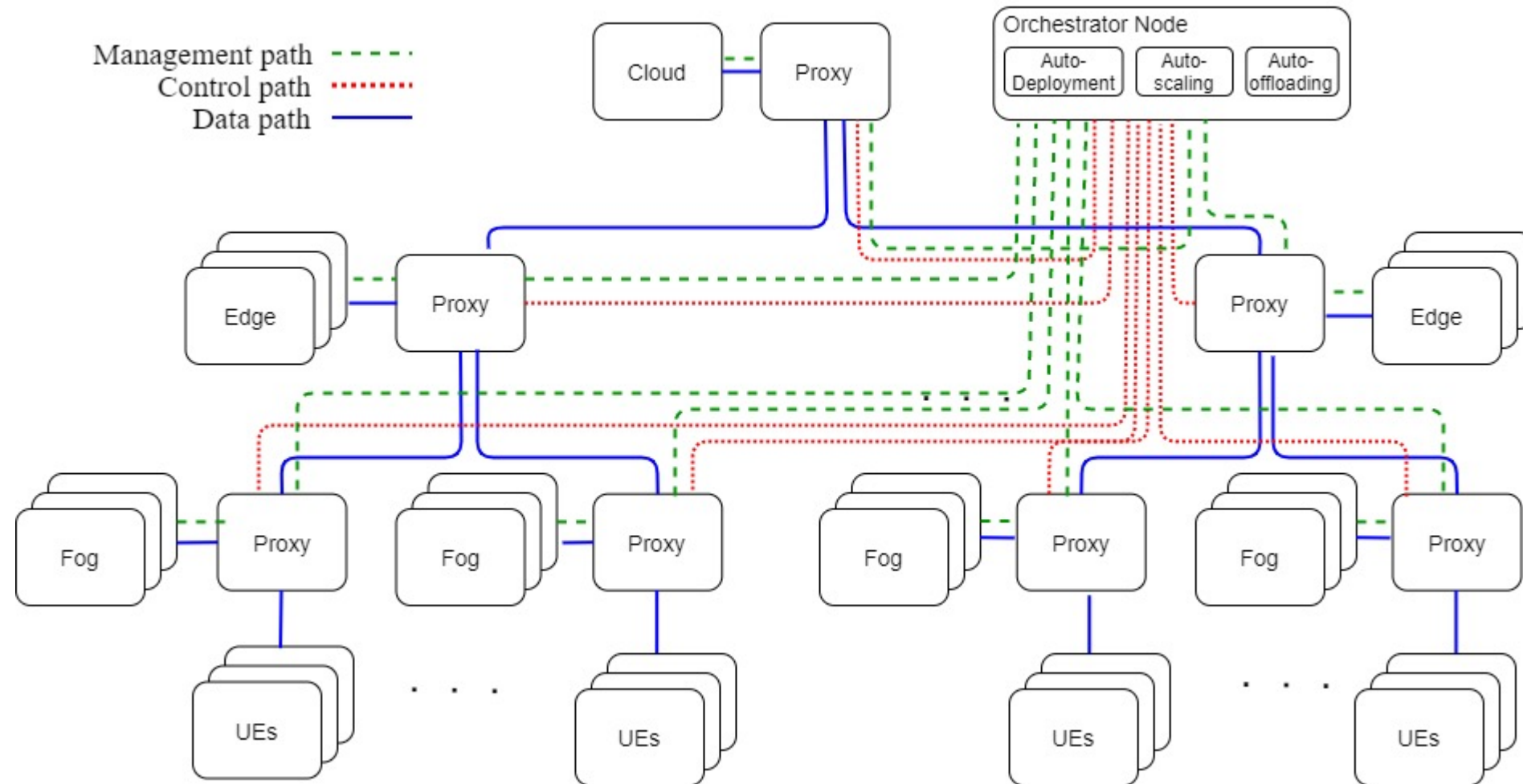
- Configurations of network architecture
- Configurations of AIoT service mapping
- Resource allocations for the AIoT services mapping

□ Objective:

- Deploy the AIoT services to the nodes automatically
- Orchestrating the resources and operational management for IoTaS

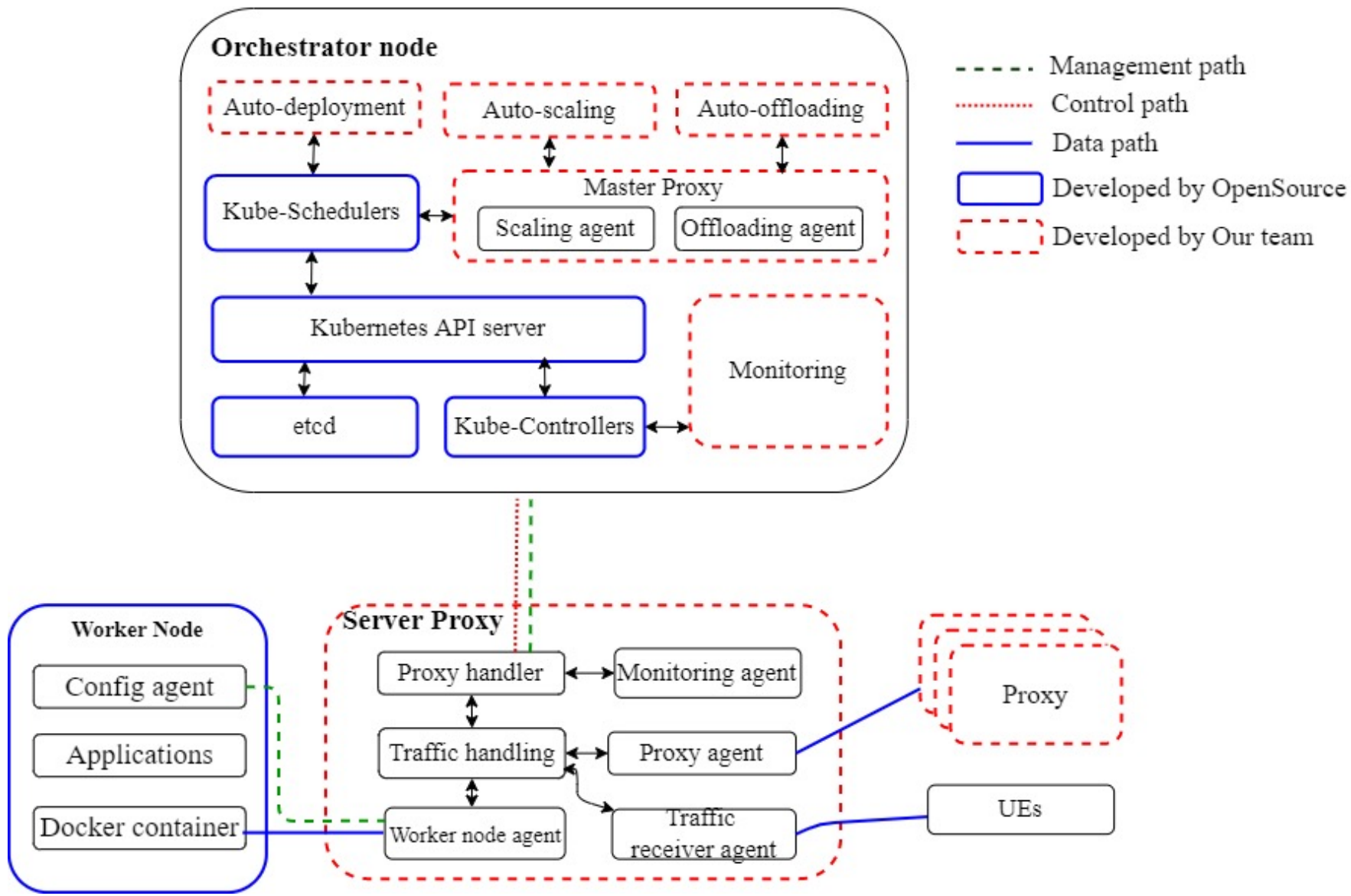
Development Track Solutions

Solution - Multi-tier Architecture for AIaaS



Solution – AISF System Design

Orchestrator (1/2)



Solution – AISF System Design

Orchestrator (2/2)

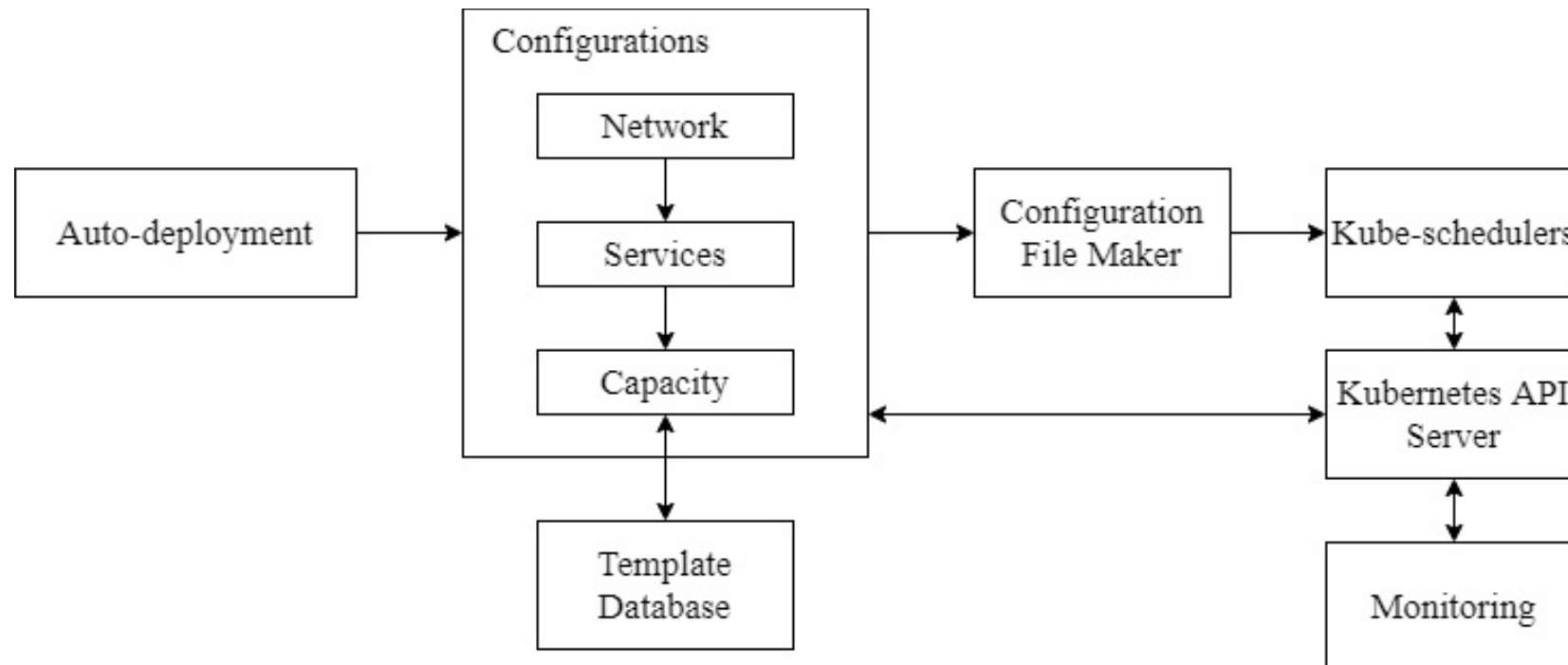
- ❑ Main plugins in orchestrator:
 - ❑ Auto-deployment → to automate the deployment
 - ❑ Auto-scaling → to automate the resource scaling of the pods
 - ❑ Auto-offloading → to offload the traffic into other paths
 - ❑ Master-proxy → the executor of auto-scaling and auto-offloading plugin
 - ❑ Monitoring → to monitor an entire system

- ❑ Master-proxy and monitoring have an agent in each worker node
 - ❑ Server proxy is an agent for master-proxy
 - ❑ Monitoring agent will be put inside a server proxy
 - ❑ Monitor the worker node and provide data to the orchestrator's monitoring plugin

Solution – AISF System Design

Auto-deployment

- ❑ Several configurations in auto-deployment:
 - Network: 1-tier, 2-tier, 3-tier
 - Services: IDaS, IoTaS
 - Capacity: specify the capability requirements of the services
 - Interfaced with a template database
 - Set of default mappings
 - Based on latency constraint and a number of users



Solution – AISF System Design

Proxy

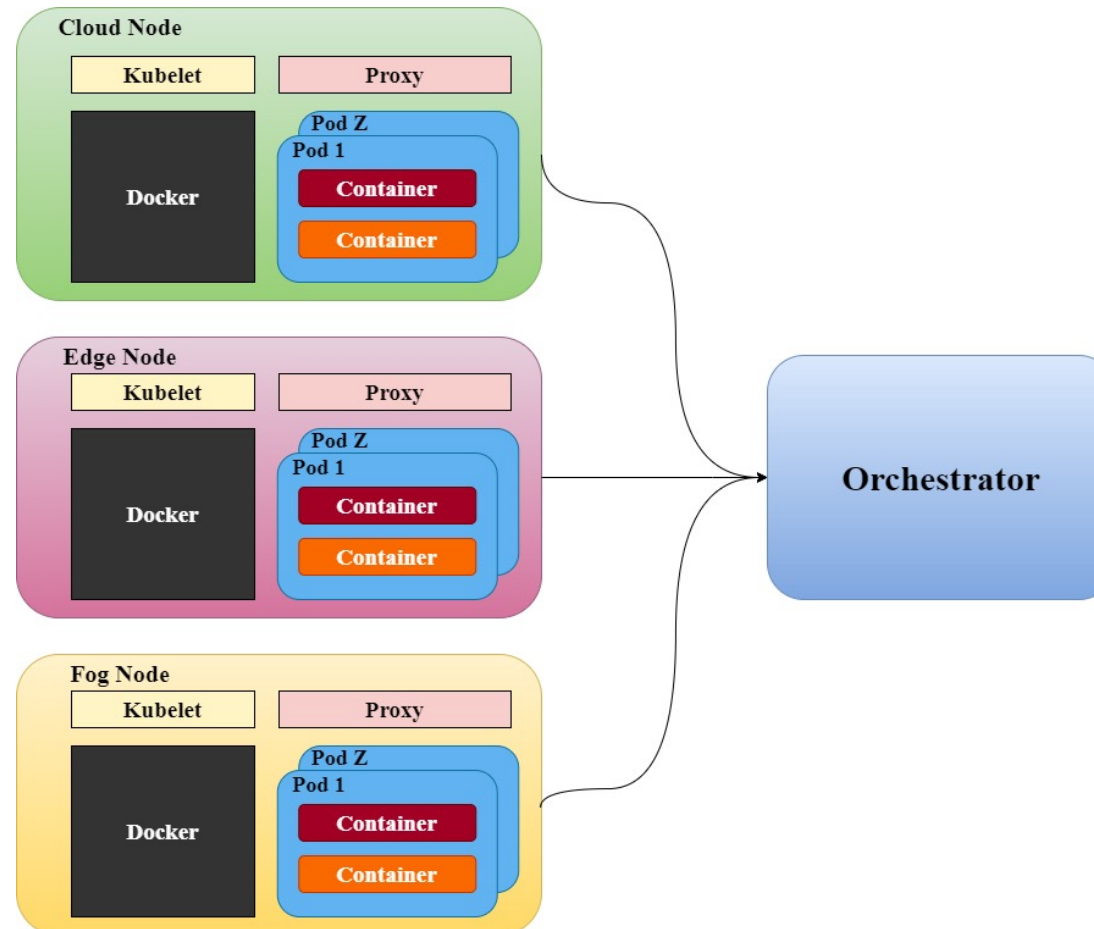
- ❑ Master proxy has two components:
 - Scaling agent → to receive the scaling table from auto-scaling
 - Interact with kube-schedulers to execute the scaling process
 - Replicate the new pods required and their destination
 - Schedulers will instruct API servers to perform replication
 - Offloading agent → to receive the offloading table from auto-offloading
 - Push the offloading table into server proxy
 - Traffic handling receives it and executes it in a server proxy

- ❑ Server proxy in worker nodes has:
 - Proxy handler → to receive the instructions from orchestrator
 - Worker node agent → to automatically deploy the worker node
 - Monitoring agent → to monitor the worker node resource
 - Traffic handling → to determine where the traffic should be forwarded
 - Proxy agent → to communicate with other proxies
 - Traffic receiver agent → to receive the traffic from UE

Solution – AISF System Design

Worker node

- ❑ Utilize container-based virtualization
- ❑ Each AI service will be threatated as a pod
- ❑ ML applications for each AI service will be threatated as a container



Solution – Auto Scaling

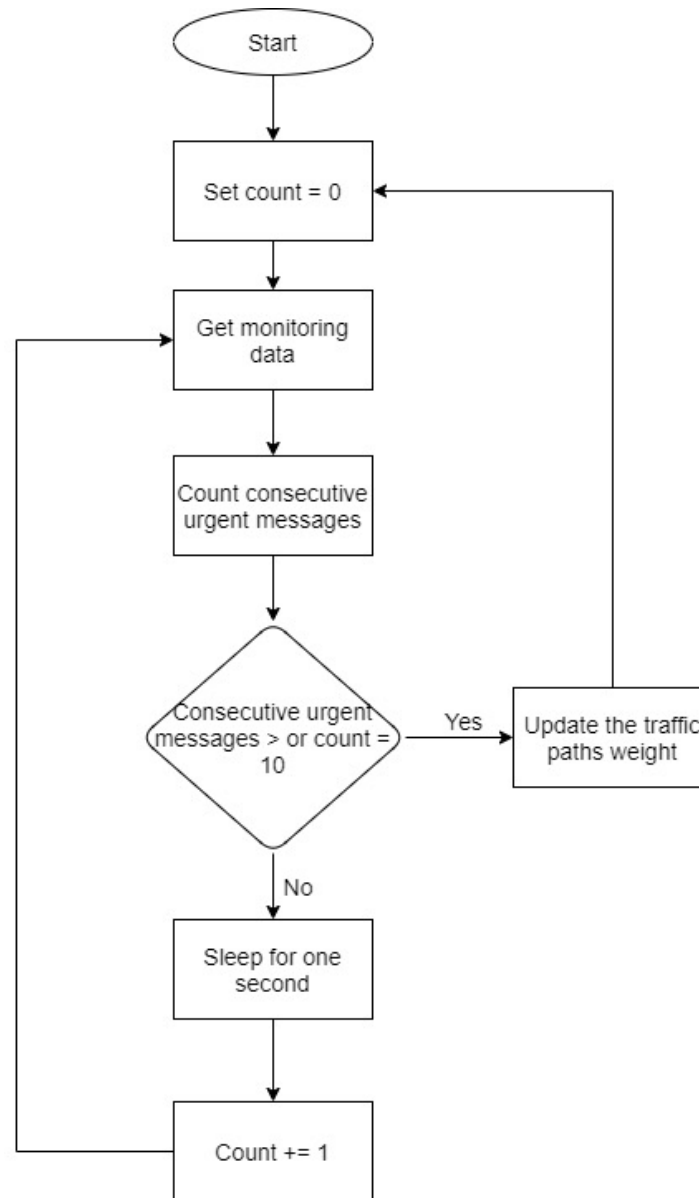
Horizontal Autoscaling (HAS)

- ❑ 2 phases of autoscaling process
 - 1st phase → add replication pods in that node
 - 2nd phase → add replication pods to other nodes
- ❑ 1st phase scaling:
 - Will be activated if:
 - Delay satisfaction percentage in pods < threshold
 - GPU or CPU utilization in pods > 90%
 - Replicate the high utilization pods
 - Load balancing the traffic into the original and replicate pods
- ❑ 2nd phase scaling:
 - Will be activated if:
 - 1st phase scaling reaches maximum pods and runs out of resources
 - Replication pods will be replicated to other nodes in the same tier
 - When the second phase is enabled, the offloading plugin calculates the offloading ratio

Solution – Auto Offloading (1/2)

- ❑ Auto-offloading runs in orchestrator node
 - ❑ adopted from [10]
- ❑ Set up an offloading table for all server proxies
 - ❑ Based on traffic path weight
 - ❑ It is calculated based on the ratio of its latency satisfaction
 - ❑ Distribute more traffic to paths with high satisfaction
- ❑ Executed in the order of seconds, or if the monitoring module receives three consecutive urgent messages
 - ❑ A sequence of latency violation traffic triggers urgent alerts

Solution – Auto Offloading (2/2)



Solution – AISF

AISF Procedures (1/2)

- ❑ Step 1: Auto-deployment
 - Select the network architecture
 - Select the service mapping
 - Specify the resource for the chosen service
 - Interfaced with a template database
 - Based on latency constraint and service constraint
 - Send the configuration into Configuration File Maker
 - Create the pods based on the configurations
 - Schedule the deployment in Kube-Schedulers
 - API Server checks the availability of worker nodes from monitoring plugin
 - Deploy the pods into the nodes via API Server

Solution – AISF

AISF Procedures (2/2)

❑ Step 2: Auto-scaling

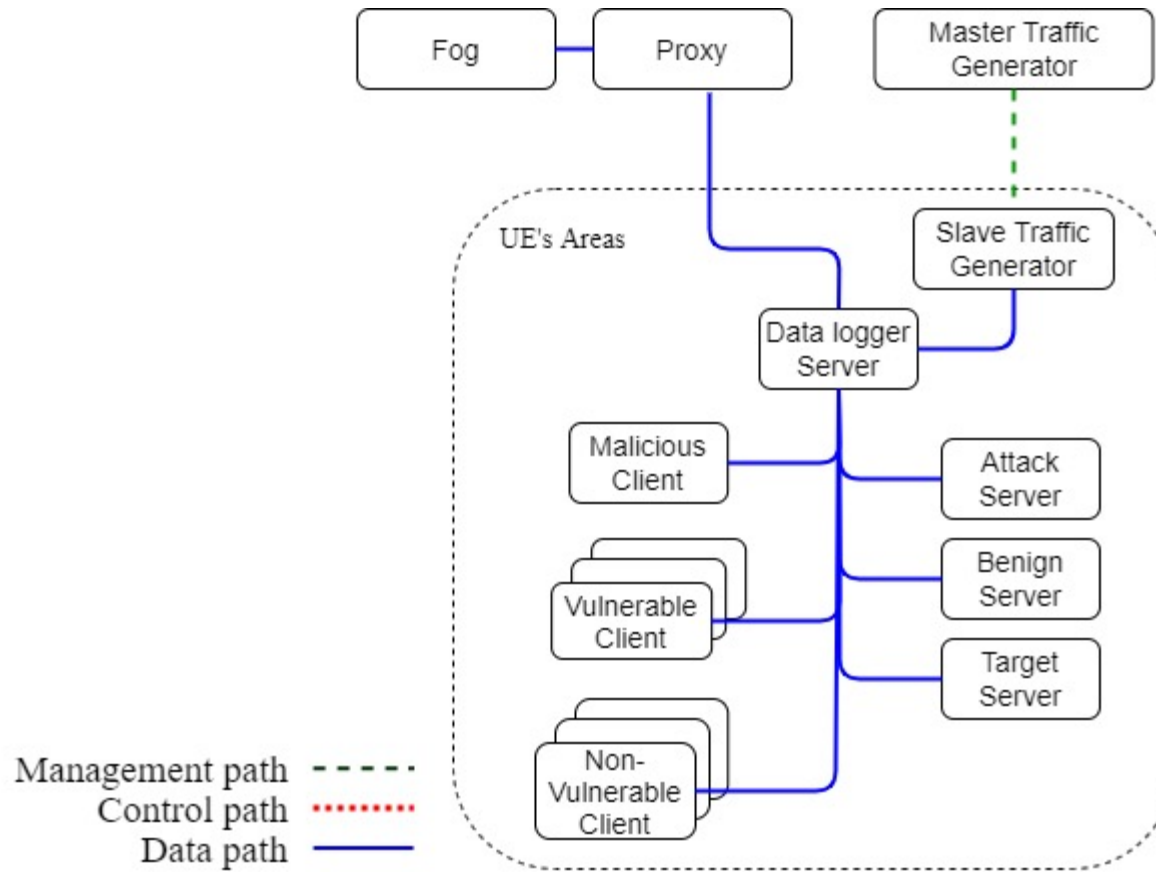
- Master proxy reports the performance $<$ threshold
- Auto-scaling plugin calculates the new replication pods required
- Send the number of new replication pods required to master proxy
- Master proxy replicate the pods
- Schedule the deployment in Kube-Schedulers
- API Server checks the availability of worker nodes from monitoring plugin
- Deploy the replication pods into the nodes via API Server

❑ Step 3: Auto-offloading

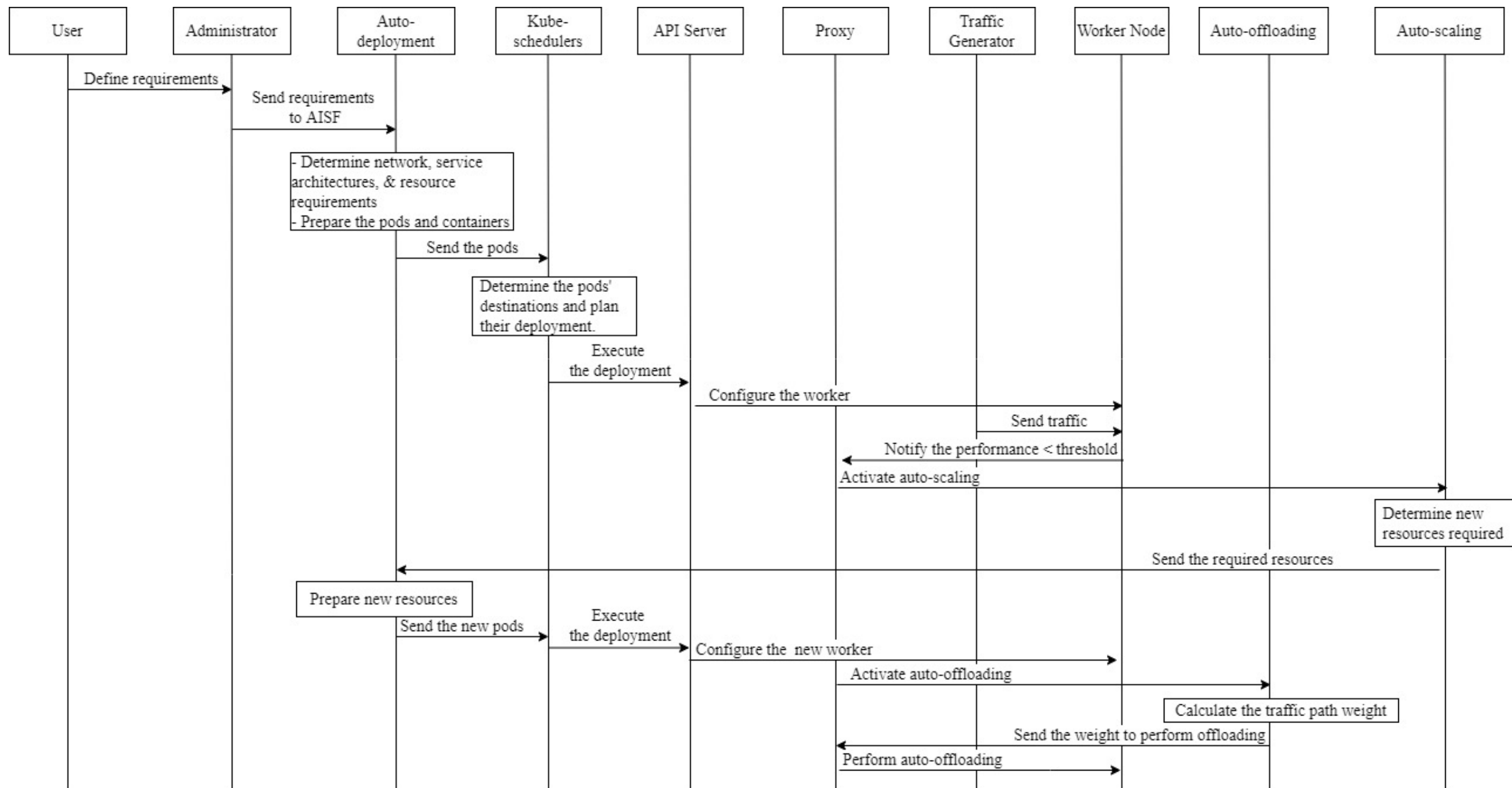
- Calculate the traffic path weight
- Send the weight to host proxy
- Server proxy performs the offloading in original pods and replication pods
- Monitoring plugin reports the nodes performance periodically
- Auto-offloading re-calculate the traffic path weight and update it to server proxy

Solution – Traffic Generator

- ❑ The traffic generator simulates traffic for system performance testing
- ❑ Traffic generator will consist with:
 - Master traffic generator → as the orchestrator for traffic generator
 - Slave traffic generator → receives the instructions from master
- ❑ This architecture will be used for the UE areas in each fog to generate traffic



Solution – AISF Process Flow



1

AISF for Intrusion Detection as a Service (IDaS)

1 AISF for IDaS Solution

- ❑ Offer IDaS by AISF for scalable ML-based IDaS (Intrusion Detection as a Service) with traffic generator
- ❑ Add IDaS into orchestrator to manage the system
- ❑ IDaS uses all plugins in AISF
 - ❑ Auto-deployment
 - ❑ Scaling
 - ❑ Offloading
- ❑ IDaS has 10 service mappings based on table in page 12

1

Solution – AISF

IDaS on AISF Procedures (1/2)

- ❑ Step 1: Auto-deployment
 - Select the network architecture: 1-tier, 2-tier, 3-tier
 - Select the **IDaS service mapping**
 - Specify the resource for the chosen service
 - Interfaced with a template database
 - Based on latency constraint, **arrival traffic rate, ratio of malicious traffic**
 - **Searches for an appropriate baseline configuration in the database**
 - **The template is created based on the algorithm and results from research track in IDaS**
 - Send the configuration into Configuration File Maker
 - Create the pods based on the configurations
 - Schedule the deployment in Kube-Schedulers
 - API Server checks the availability of worker nodes from monitoring plugin
 - Deploy the pods into the nodes via API Server

1

Solution – AISF

IDaS AISF Procedures (2/2)

□ Step 2: Auto-scaling

- Master proxy reports the **IDaS pods performance** < threshold
- Auto-scaling plugin calculates the new replication pods required
- Send the number of new replication pods required to master proxy
- Master proxy replicate the pods
- Schedule the deployment in Kube-Schedulers
- API Server checks the availability of worker nodes from monitoring plugin
- Deploy the replication pods into the nodes via API Server

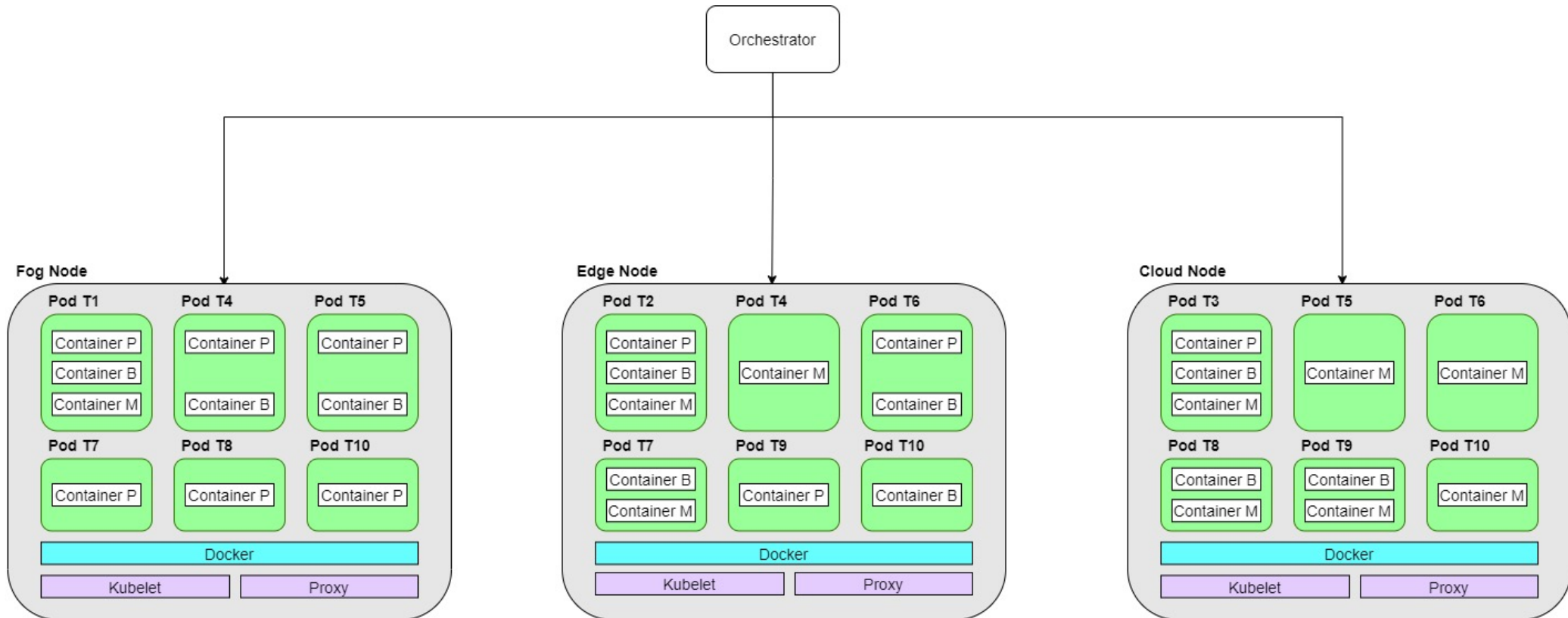
□ Step 3: Auto-offloading

- Calculate the traffic path weight
- Send the weight to host proxy
- Server proxy performs the offloading in original pods and replication pods
 - **Offload the raw traffic**
- Monitoring plugin reports the nodes performance periodically
- Auto-offloading re-calculate the traffic path weight and update it to server proxy

1 AISF for IDaS Solution

Worker Node Slicing Design (1/2)

- ❑ Utilize container-based virtualization
- ❑ Slice the server for 10 service architectures with static resource
- ❑ Each node can have 6 pods for IDaS tasks
- ❑ Each IDaS task will be threatened as a container



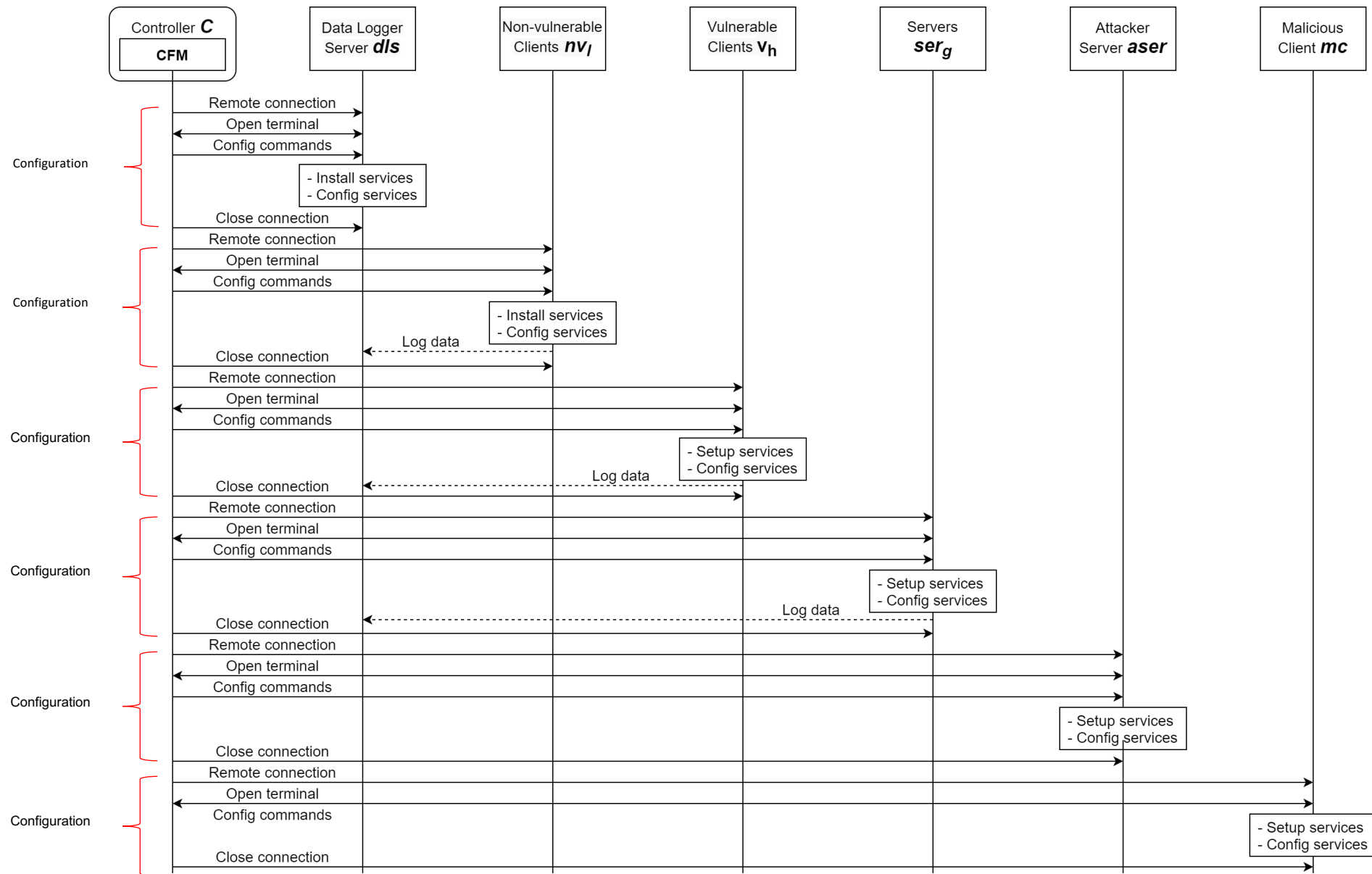
1 AISF for IDaS Solution

Traffic Generator

- ❑ Traffic generator will integrate CRÈME
 - It will be used for generating the attack and benign traffic
 - To test the system's performance
 - Master traffic generator will control slave generator in each UE's area

AISF for IDaS Solution

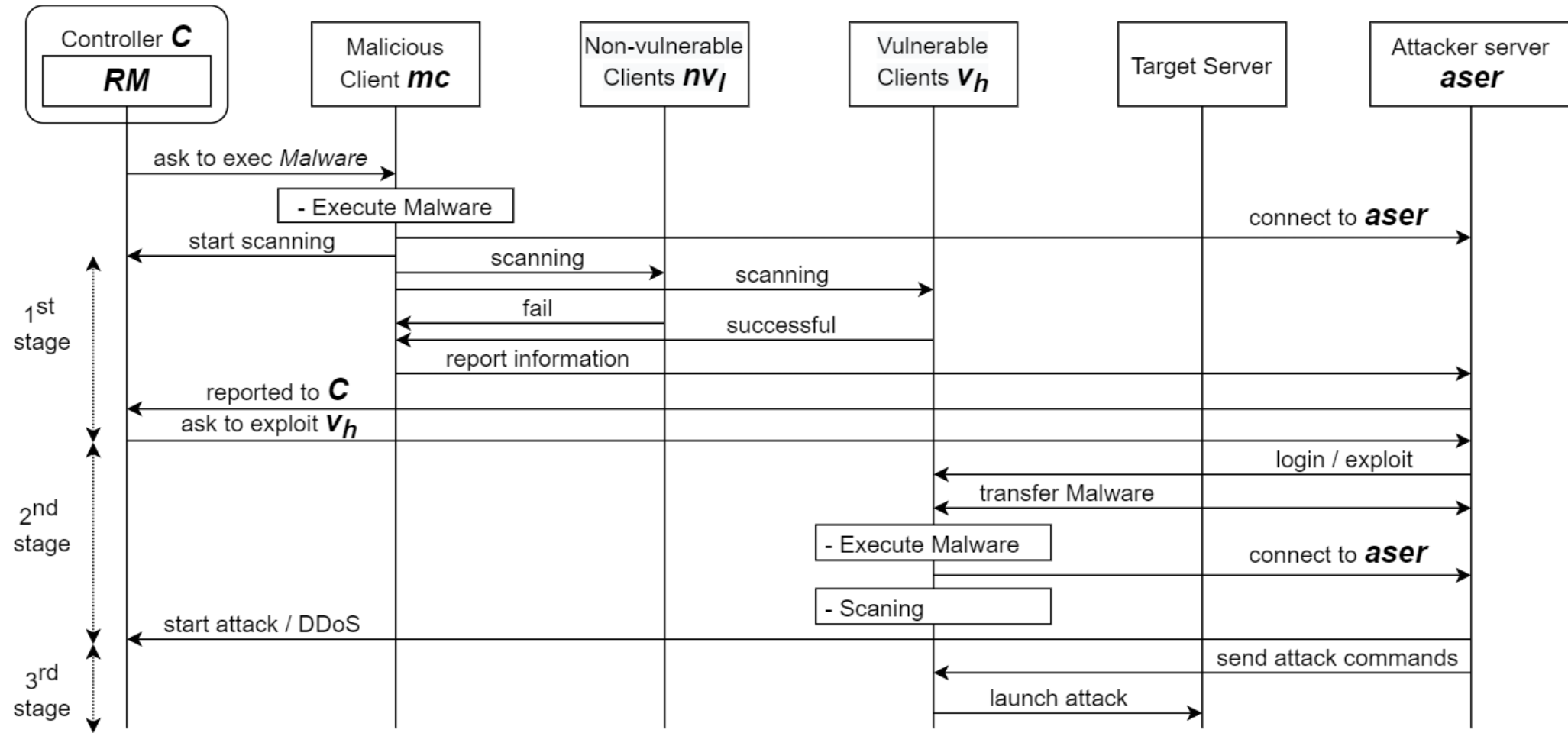
Traffic Generator - Process flow (1/2)



*Adopted from CREME

1 AISF for IDaS Solution

Traffic Generator - Process flow (2/2)



*Adopted from CREME

2

AISF for IoTaS Solution

2 AISF for IoTaS Solution

- ❑ Offer AIoT service into AISF for scalable IoTaS system
- ❑ Add IoTaS into orchestrator to manage the system
- ❑ IoTaS uses all plugins in AISF
 - ❑ Auto-deployment
 - ❑ Scaling
 - ❑ Offloading
- ❑ IoTaS utilize three pods (detection, learning, aggregation)
- ❑ Scaling and offloading will be used for inference pods
 - ❑ Detection must be processed quickly
 - ❑ When the performance of detection $<$ threshold:
 - ❑ Horizontal scaling
 - ❑ Offloading

2 Solution – AISF

IoTaS on AISF Procedures (1/2)

- ❑ Step 1: Auto-deployment
 - Select the network architecture: 1-tier, 2-tier, 3-tier
 - Select the **AIoT service mapping**
 - Specify the resource for the chosen service
 - Interfaced with a template database
 - Based on latency constraint, **arrival traffic rate**
 - **Searches for an appropriate baseline configuration in the database**
 - **The template is created based on the algorithm and results from research track in AIIoT**
 - Send the configuration into Configuration File Maker
 - Create the pods based on the configurations
 - Schedule the deployment in Kube-Schedulers
 - API Server checks the availability of worker nodes from monitoring plugin
 - Deploy the pods into the nodes via API Server

Solution – AISF

IoTaS AISF Procedures (2/2)

□ Step 2: Auto-scaling

- Master proxy reports the **detection pods performance** < threshold
- Auto-scaling plugin calculates the new replication pods required
- Send the number of new replication pods required to master proxy
- Master proxy replicate the pods
- Schedule the deployment in Kube-Schedulers
- API Server checks the availability of worker nodes from monitoring plugin
- Deploy the replication pods into the nodes via API Server

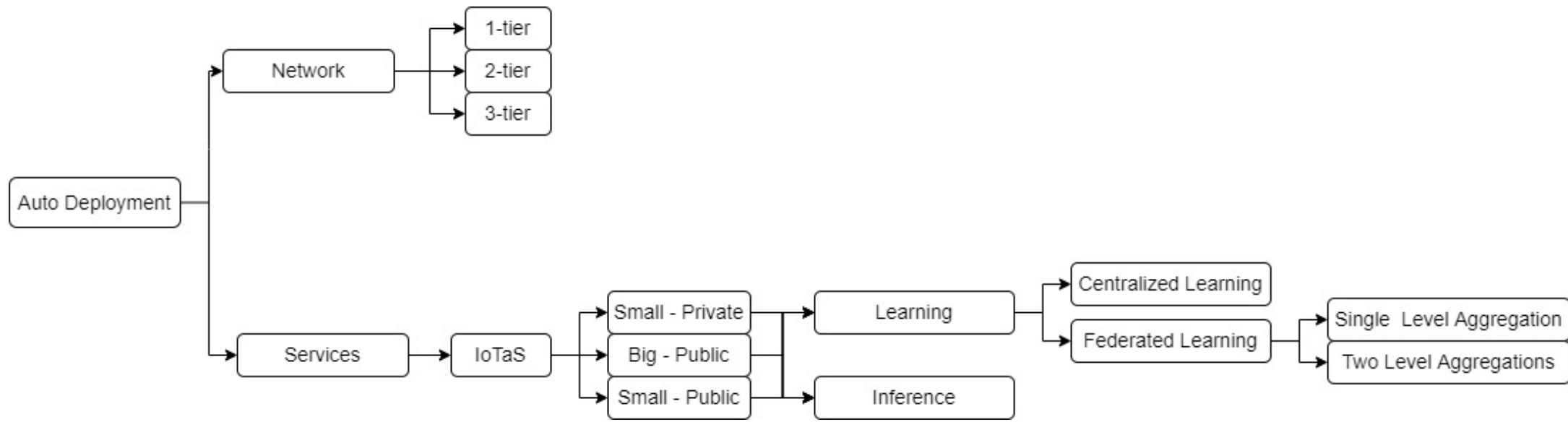
□ Step 3: Auto-offloading

- Calculate the traffic path weight
- Send the weight to host proxy
- Server proxy performs the offloading in original pods and replication pods
 - **Offload the AIoT traffic**
- Monitoring plugin reports the nodes performance periodically
- Auto-offloading re-calculate the traffic path weight and update it to server proxy

2 IoTaS on AISF Solution

Auto-deployment

- ❑ Auto deployment will be used to automate the deployment
 - Manage services and network architectures
 - Network architectures: 1-tier, 2-tier, 3-tier
 - Services: IoTaS



Implementation – Open sources and tools

Category	Name	Functionality
Orchestrator	Kubernetes Master	controls and manages a set of nodes
Worker Nodes	Kubernetes	manages containerized applications
Monitoring System	Prometheus	monitor resource usage
Data collection	Tcpdump	capturing network packets
Features extraction	Argus	extracting features for network packets
Attacks	Mirai	emulate Mirai botnet
	MySQL	database service
	Metasploit	hacking tool
	Metasploitable 3	vulnerable Linux virtual machine
Machine learning	Scikit-learn	machine learning library
	Tensorflow	deep learning and federated learning library
	Optuna	Hyperparameter optimization

Thank You